Redbooks
ibm.com/redbooks

IBM Cognos Dynamic Cubes

Dmitriy Beryoza

MaryAlice Campbell

Cesar Cardorelle

Tod Creasey

David Cushing

Vlaunir Da Silva

Sean David

Avery Hagleitner

Ian Henderson

Daniel Howell

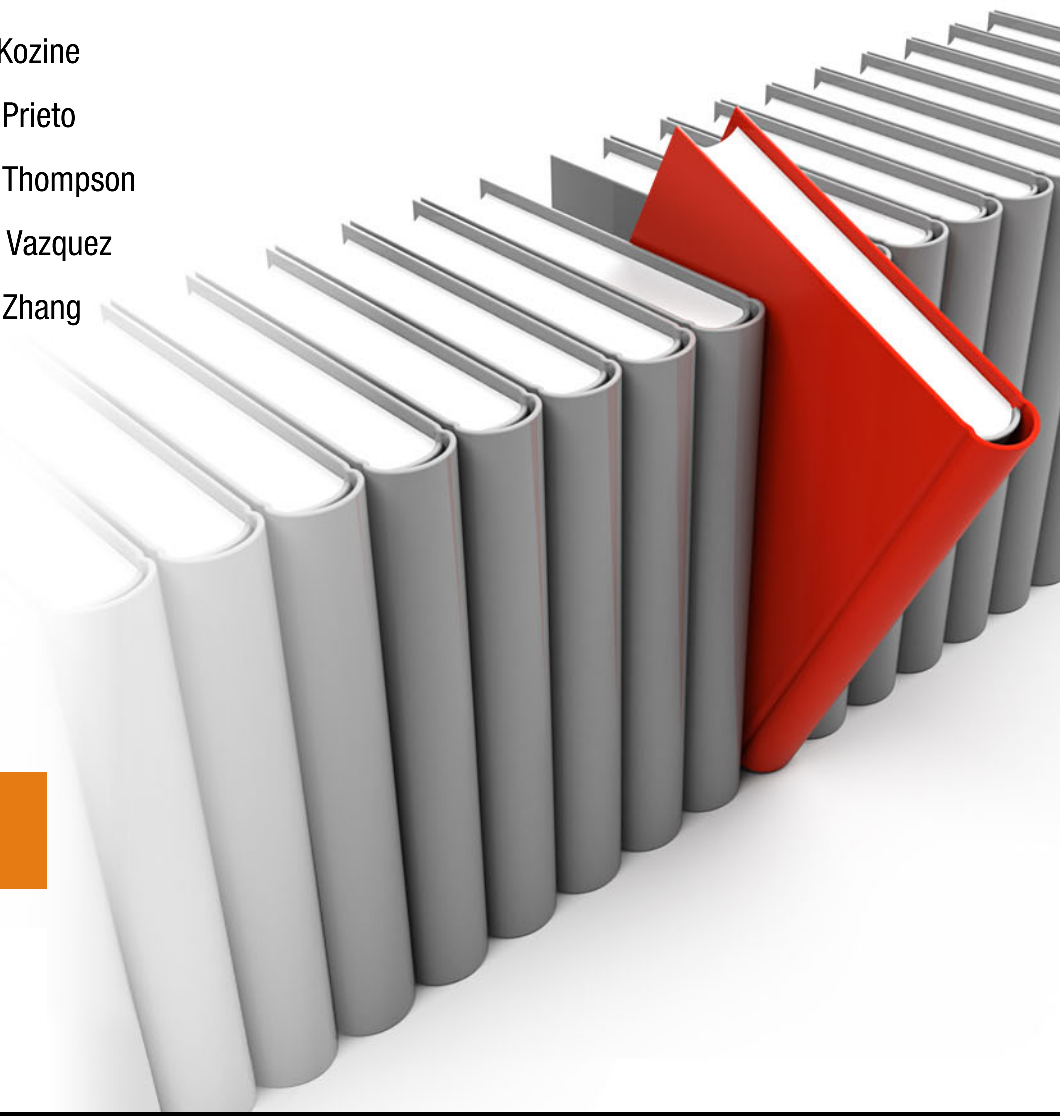
Igor Kozine

Paul Prieto

Paul Thompson

Jose Vazquez

Ying Zhang



Analytics

IBM

Redbooks

International Technical Support Organization

IBM Cognos Dynamic Cubes

July 2015

SG24-8064-01

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

Second Edition (July 2015)

This edition applies to Version 10, Release 2, Modification 2 of IBM Cognos Business Intelligence (product number 5724-W12).

Contents

Notices

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

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

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.

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

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

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

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

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

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

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.

COPYRIGHT LICENSE:

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

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at http://www.ibm.com/legal/copytrade.shtml

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

Cognos®
DB2®
developerWorks®
IBM®
Impromptu®
InfoSphere®
Insight™
PowerPlay®
Redbooks®
Redbooks (logo) ®
TM1®

The following terms are trademarks of other companies:

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

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

Java, and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

UNIX is a registered trademark of The Open Group in the United States and other countries.

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

Preface

IBM® Cognos® Business Intelligence (BI) provides a proven enterprise BI platform with an open data strategy. Cognos BI provides customers with the ability to use data from any source, package it into a business model, and make it available to consumers in various interfaces that are tailored to the task.

IBM Cognos Dynamic Cubes complements the existing Cognos BI capabilities and continues the tradition of an open data model. It focuses on extending the scalability of the IBM Cognos platform to enable speed-of-thought analytics over terabytes of enterprise data, without having to invest in a new data warehouse appliance. This capability adds a new level of query intelligence so you can unleash the power of your enterprise data warehouse.

This IBM Redbooks® publication addresses IBM Cognos Business Intelligence V10.2.2 and specifically, the IBM Cognos Dynamic Cubes capabilities. This book can help you in the following ways:

- Understand core features of the Cognos Dynamic Cubes capabilities of Cognos BI V10.2.2
- Learn by example with practical scenarios by using the IBM Cognos samples

This book uses fictional business scenarios to demonstrate the power and capabilities of IBM Cognos Dynamic Cubes. It primarily focuses on the roles of the modeler, administrator, and IT architect.

More information about IBM Cognos Business Intelligence V10.2.2 is in the IBM Knowledge Center:

http://ibm.co/1zZb7hP

Also, see the *Dynamic Cubes User Guide*, V10.2.2:

http://ibm.co/1QIQVoG

The team who wrote this book

This book was produced by a team of specialists from around the world working at the International Technical Support Organization, Raleigh Center.

Dmitriy Beryoza is an Advisory Software Developer IBM Analytics Solutions based in Ottawa, Canada. He has over 20 years of experience in software design and implementation. Dmitriy has worked at IBM since 2001, focusing on data access technologies (including Cognos Dynamic Query and Cognos Dynamic Cubes) in several generations of IBM Cognos products. Before joining IBM, Dmitriy worked on software projects for a number of companies and research labs as an architect, developer, and consultant. He holds a Ph.D. in Computer Science and has extensive experience in Java and C++ programming, and web and systems software development.

MaryAlice Campbell is a Sr. Consultant and Business Analytics Technical Practice Leader at ISW, Australia. She has over 20 years of experience as a business analytics specialist. MaryAlice is an IBM Cognos BI veteran having cut her teeth on the early, pre-web versions of IBM Cognos PowerPlay® and IBM Cognos Impromptu®; she contributed to beta and training programs, and has worked with all subsequent releases. MaryAlice is also an IBM certified solution developer, internationally recognized educator and a Master Instructor of the IBM Analytics curriculum.

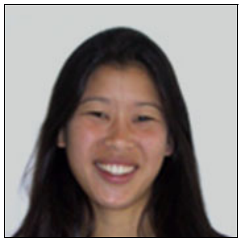
Cesar Cardorelle is a Principal Solution Architect in the IBM Business Analytics Platform products area and is currently a senior member of the IBM European Product and Technology Experts team and he is also a member of the IBM European RAVE Tiger team. César joined IBM with the Cognos acquisition and he is based in Paris, France. He has over 26 years of experience in development, implementation, and support of fourth-generation programming language (4GL) and several IBM Business Analytics solutions. His areas of expertise include relational databases, data warehousing, online analytical processing (OLAP), business intelligence applications design, modelling and reporting, business analytics system architecture and application tuning. César holds a Master's degree from Ecole Nationale Supérieure Maritime (ENSM), France and a 3rd Cycle Certificate in Computer Science from Centre Normand de Recherche en Informatique (CENORI), France.

Tod Creasey is a Software Development Manager in the IBM Cognos Business Intelligence organization responsible for the user interface of IBM Cognos Dynamic Cubes. Tod has worked at IBM since 1994 in a variety of positions including development environments, Eclipse, and business intelligence products since 2009. Tod is a frequent speaker at technical conferences, most recently presenting Cognos Dynamic Cubes sessions at IBM Insight™, the premier data and analytics conference.

David Cushing is the Product Manager for IBM Cognos Dynamic Cubes, based in Ottawa, Canada. David has been with IBM Cognos for 25 years and joined IBM as a result of the acquisition of Cognos. David has a Masters of Computer Science degree from Dalhousie University, Nova Scotia, Canada.

Vlaunir Da Silva is a Technology Solutions Architect in the IBM Analytics Product and Technology Experts Team in Miami, Florida, US. Vlaunir is originally from Brazil and holds a degree in Computer Sciences. Vlaunir has over 18 years of experience in business analytics, database support, application development, and software support. Vlaunir holds several professional certifications and has written several technical articles, taught classes, and presented at international conferences.

Sean David is a Software Quality Engineer in IBM Analytics Solutions based in the Ottawa, Canada Software Lab. For 12 years, Sean has been involved in the testing of IBM Cognos BI for a number of OLAP data sources including SAP NetWeaver Business Warehouse, IBM Cognos PowerCubes, Oracle Essbase, IBM InfoSphere® Warehouse Cubing Services, and most recently, IBM Cognos Dynamic Cubes. Sean is an Electrical Engineering graduate from the University of New Brunswick, Canada.



Avery Hagleitner is a Software Architect for IBM Cognos Dynamic Cubes in IBM Analytics Solutions at the IBM Silicon Valley Laboratory. Avery has over 14 years of software development experience at IBM. Her interests range from high-performance Java server applications to engaging graphical user interfaces. Her areas of expertise include business intelligence, data warehousing, and online analytical processing (OLAP). Avery holds a Master's degree in Software Engineering from San Jose State University, California, USA, a Bachelor of Science in Computer Science and a minor in Psychology from the University of California, San Diego.

Ian Henderson is the QE Technical Team Lead for IBM Cognos Business Analytics Modeling applications. He has nearly 25 years of experience in the area of business intelligence, modeling, and reporting.

Daniel Howell is a Technology Solutions Architect in the IBM Analytics Product and Technology Experts Team in Phoenix, Arizona, USA. Dan has over 20 years of experience in multidimensional databases and Business Intelligence software. He has worked for a number of BI software vendors and as an independent consultant. Over the years, he has implemented dozens of solutions in various industries. Dan joined IBM in 2008.

Igor Kozine is an Advisory Software Engineer at IBM. Igor is currently the Performance and Scalability Verification Leader for IBM Cognos Dynamic Cubes based in Ottawa, Canada. He has over 12 years of experience in the software industry. His areas of expertise include business intelligence, data access, modeling, data warehousing, and optimization of large scale applications.

Paul Prieto is an Executive IT Specialist in the European Analytics division at IBM. He has over 18 years of experience in the IT industry with a special focus on Analytics. During his career Paul took on various roles ranging from technical account management for large financial organizations to his current assignment leading the technical strategy and execution for Analytics SaaS solutions at IBM Europe. Paul is currently a Fellow of the British Computer Society (BCS) and Chartered IT Professional. Paul enjoys being a mentor, writing, and teaching.

Paul Thompson is an Advisory Software Engineer at the IBM Ottawa Lab and the technical lead for IBM Cognos Cube Designer. He has 20 years of experience building online analytical processing (OLAP) products including IBM Cognos Business Intelligence, IBM Cognos Framework Manager, IBM Cognos PowerPlay, and IBM Cognos Transformer. Paul learned to write software as a child using his family's Commodore 64 home computer.

Jose Vazquez is an Advisory Software Engineer in IBM Analytics Solutions. He has over 20 years of experience in software engineering. Jose's areas of expertise include Online Analytical Processing (OLAP), data warehousing, and Multidimensional Expression Language (MDX) engine technologies. Since joining IBM in 2008 Jose took several assignments as a member of the Data Access team, most recently in IBM Cognos Dynamic Cubes and the design and implementation of the MDX engine for IBM Cognos Dynamic Query. Prior to joining IBM Jose worked for over 12 years in the design and implementation of the IBM Cognos TM1® server engine. Jose holds a Bachelor's degree in Computer Science from the Havana University, Cuba.

Ying Zhang is an Advisory Software Engineer at the IBM Silicon Valley Laboratory in San Jose, California, USA. She has over 10 years of experience with Business Intelligence software. In her current assignment, Ying is working on IBM Cognos Dynamic Cubes with focus on dimensional security and virtual cubes. Ying holds a master's degree in computer science.

The project that produced this publication was managed by **Marcela Adan**, IBM Redbooks Project Leader - IBM International Technical Support Organization, Global Content Services.

Thanks to the following people for their contributions to this project:

Christopher Yao
IBM Cognos Dynamic Cubes development, IBM Analytics

Ralf Vierich
IBM Security

Now you can become a published author, too

Here's an opportunity to spotlight your skills, grow your career, and become a published author—all at the same time! Join an ITSO residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our books to be as helpful as possible. Send us your comments about this book or other IBM Redbooks publications in one of the following ways:

- Use the online **Contact us** review Redbooks form found at:

  ibm.com/redbooks

- Send your comments in an email to:

  redbooks@us.ibm.com

- Mail your comments to:

  IBM Corporation, International Technical Support Organization

  Dept. HYTD Mail Station P099

  2455 South Road

  Poughkeepsie, NY 12601-5400

Stay connected to IBM Redbooks

- Find us on Facebook:

  http://www.facebook.com/IBMRedbooks

- Follow us on Twitter:

  http://twitter.com/ibmredbooks

- Look for us on LinkedIn:

  http://www.linkedin.com/groups?home=&gid=2130806

- Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:

  https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm

- Stay current on recent Redbooks publications with RSS Feeds:

  http://www.redbooks.ibm.com/rss.html

Summary of changes

This section describes the technical changes made in this edition of the book and in previous editions. This edition might also include minor corrections and editorial changes that are not identified.

Summary of Changes
for SG24-8064-01
for IBM Cognos Dynamic Cubes
as created or updated on July 31, 2015.

July 2015, Second Edition

This revision reflects the addition, deletion, or modification of new and changed information described below.

New information

The following chapters are new in this edition:

- Chapter 4. Modeling with IBM Cognos Cube Designer
- Chapter 5. Basic modeling
- Chapter 6. Advanced topics in modeling
- Chapter 10. Securing the IBM Cognos BI environment
- Chapter 12. Near real-time updates
- Chapter 15. Query service memory monitoring
- Chapter 16. Using Framework Manager models
- Chapter 17. Hardware sizing
- Chapter 18. Dynamic cubes in a multi-dispatcher environment

Changed information

All chapters changed to reflect IBM Cognos BI V10.2.2 Fix Pack 1, IBM Cognos Dynamic Cubes V10.2.2 Fix Pack 1, and to add information captured from lessons learned in the field.

1

Overview of Cognos Dynamic Cubes

This chapter provides an overview of IBM Cognos Dynamic Cubes: how the solution extends the scalability of the IBM Cognos Business Intelligence platform and uses the core strengths of an enterprise data warehouse. This chapter also describes the architecture of Dynamic Cubes: the IBM Cognos Cube Designer, the IBM Cognos Dynamic Cubes Server, and the Aggregate Advisor, a part of IBM Cognos Dynamic Query Analyzer.

This chapter contains the following sections:

- Introduction to IBM Cognos Dynamic Cubes
- Skill set requirements for IBM Cognos Dynamic Cubes
- When to use Cognos Dynamic Cubes
- System requirements
- Comparison to other IBM cube technologies
- Overview of the remainder of the book

1.1 Introduction to IBM Cognos Dynamic Cubes

With the advent of new data types and data sources, data warehouses are more critical than ever. The information in a well-designed data warehouse contains the corporate memory of an organization and the context to make sense of other data. For example, the contextual value of your business products and locations will make social data from systems such as Apache Hadoop much more relevant.

In addition, when new data is collected that provides insight, either from new sources or from instrumented devices, the ability to store it for later reference is critical. Recording analyses and point-in-time reports in an organized system provides this ability.

It is no surprise, then, that the data warehousing market continues to increase in popularity, growing at double digits year after year. The volume of data being generated from multiplying sources makes these technologies a must-have and must-use for virtually all organizations.

IBM Cognos Business Intelligence provides a proven enterprise BI platform with an open-data access strategy. It provides customers with the ability to pull data from various data sources, package it into a business model, and make it available to consumers in a variety of interfaces that are suited to the task, using business language that is relevant to consumers.

Cognos Dynamic Cubes complements the existing query engine, and extends Cognos scalability to enable speed-of-thought analytics over terabytes of enterprise data, without being forced to rely on a new data-warehousing appliance. This capability adds a new level of query intelligence so you can unleash the power of your large enterprise data warehouse.

Figure 1-1 shows how Cognos Dynamic Cubes is tightly integrated into the Cognos BI stack, and how its data can be surfaced through any of the Cognos interfaces. This approach allows existing customers to integrate this technology into their application environment without affecting existing users who are already familiar with interfaces such as Report Studio, IBM Cognos Workspace, and IBM Cognos Workspace Advanced.

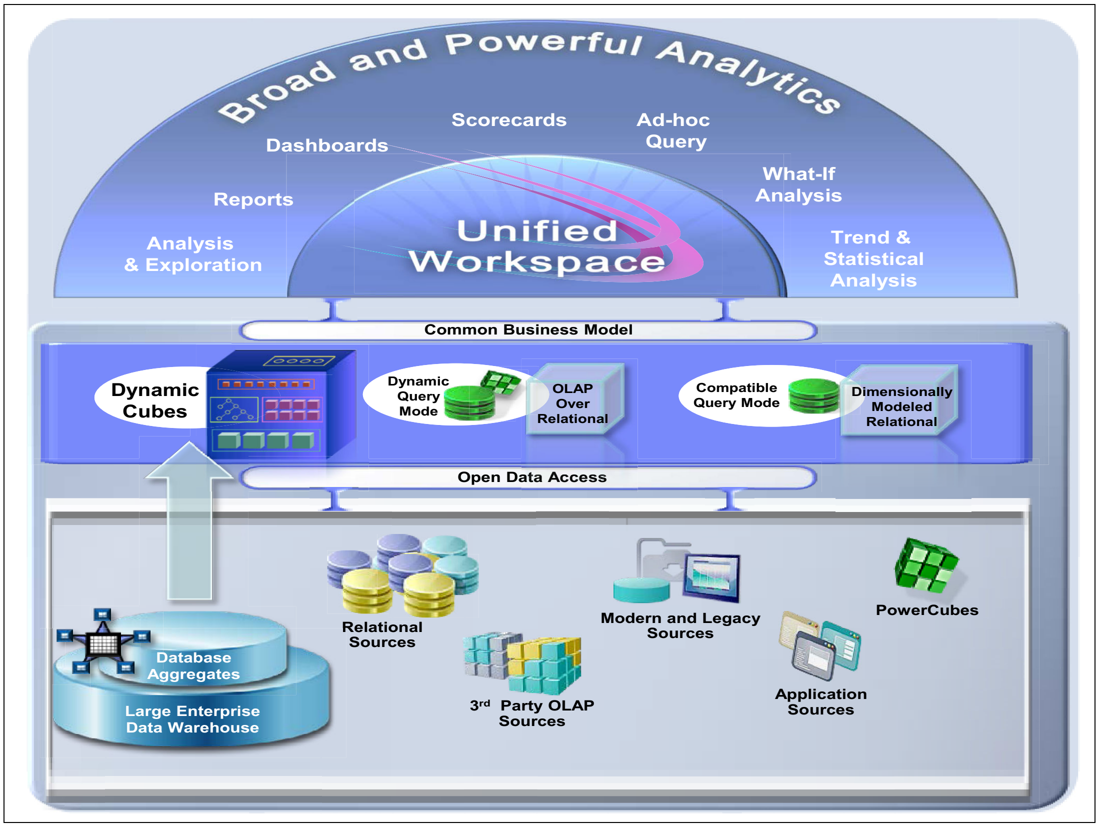


Figure 1-1 Cognos Dynamic Cubes integrated into the Cognos BI stack

Table 1-1 shows the Cognos Dynamic Cubes features introduced in the different releases of Cognos Business Intelligence.

Table 1-1 Cognos Dynamic Cubes features introduced in each release of Cognos BI

Cognos BI release	Cognos Dynamic Cubes features
10.2	► In-database and in-memory aggregates. ► Aggregate Advisor. ► Member and measure security. ► Cognos Administration support for Dynamic Cubes.
10.2.1	► Centralized Dynamic Cubes administration. ► Cube Designer Cubing Services metadata import. ► Cube Designer generation of dimensions and cubes from physical metadata. ► New `Performance Issues` tab added to Cube Designer. ► Support for FIRST, LAST, and CURRENT PERIOD aggregation rules. ► Aggregate Advisor suggest database aggregate tables to assist in loading of in-memory aggregates. ► Support for dimension and attribute security. ► Support for table-based member security filters and ability to refresh member security.

Cognos BI release	Cognos Dynamic Cubes features
10.2.1 FP1	► Dimension and measure filters. ► Measure sorting. ► Macros in expressions. ► Aggregate Advisor ability to run using workload information only, and ability to apply subset of the in-memory aggregate recommendation.
10.2.1 FP2	No features added to this release.
10.2.1 FP3	► Near real-time updates. ► Relative time improvements, including future time periods and custom relative time members. ► Named sets. ► Shared dimensions between cube member caches. ► Create, edit, and publish packages in Cube Designer. ► `dcadmin` command-line utility. ► Enable and disable workload logging without restarting a cube. ► Reload `xqe.diagnosticlogging.xml` changes without restarting Query Service. ► Aggregate Advisor results stored on server.
10.2.1 FP4	No features added to this release.
10.2.1 FP5	No features added to this release.
10.2.1 FP6	No features added to this release.
10.2.2	► Cube Designer Framework Manager metadata import. ► Cube Designer hardware sizing calculator. ► User-defined in-memory aggregates. ► Automatic in-memory aggregate optimization. ► Memory monitor. ► Post in-memory trigger. ► Cube Designer support for parameter maps. ► Cube Designer available as stand-alone installation and available as Windows 64-bit.
10.2.2.FP1	► Loading of in-memory aggregates from existing in-memory aggregates. ► Calculated measure priming queries.

1.1.1 High performance interactive analysis

Cognos Dynamic Cubes is a technology meant to solve a specific but growing business problem: enabling high-performance interactive analysis over terabytes of data contained in an enterprise data warehouse (EDW).

As data volumes grow, analyzing that data with speed-of-thought performance can be challenging. Even with modern data warehouse technology, some operations require significant computation or data movement. This computation or movement creates delays and reduces the satisfaction of business users who want to perform these analyses.

Various ways exist to accomplish performance over large volumes of data. From self-contained cubes to large in-memory appliances, different vendors are employing variations of similar methodologies to give business users timely response times.

The Cognos Dynamic Cubes technology aims to give maximum flexibility in how memory is leveraged to accelerate interactive analysis over terabytes of data, giving you the ability to evolve your deployments over time.

1.1.2 Enterprise data warehouses for analytics

Data warehouses are the recognized foundation for enterprise analytics. They enable an organization to bring together cleansed data from separate sources of input, both internal and external, such as from partners or suppliers. Instead of *garbage in, garbage out* information to support decision-making, a consistent and consolidated enterprise-wide view of data from a business provides the foundation to improve your business. Building upon a trusted information platform for analytics is a key contributor to long-term business health. Not only do data warehouses enable higher quality information, they are designed to enable high-performance data access for analytic style applications.

Cognos Dynamic Cubes technology helps to take advantage of the core strengths of an EDW and take the EDW to the next level of performance for analytics, while making the deploying and tuning easier and faster.

1.1.3 Data volumes and context in the Enterprise Data Warehouse

With social data generating petabytes per day, and instrumented devices becoming the norm, data volume growth is accelerating at an unprecedented pace.

Big data is a growing business trend, with data from unconventional sources having the potential to be business disrupters. However, before the power of these new sources can be fully used, we must understand what is happening within our own business. Understanding your own business is added value of a data warehouse, and why taking full advantage of these data holdings is a critical first step to using these new sources of data.

In addition, any organization that relies on instrumented infrastructures has an opportunity to maximize the efficiency of its operations. Analytics is key to accomplishing this type of optimization, leading to concrete business results.

With data volumes exploding, and growing urgency to make sense of this sea of data, using a data warehouse technology, and maintaining a connection to the data warehouse from your analysis tools helps to rationalize investments.

Easy visibility into enterprise data is a critical step on the journey to insight from various sources, traditional and emerging. Semi-structured or unstructured data will only make sense within the context of your business. Your data warehouse contains this context.

1.1.4 Architecture summary

The Cognos Dynamic Cubes technology is part of the IBM Cognos BI query stack, and is available with existing IBM Cognos entitlements. It provides a new and powerful tool to enable high performance analytics over large data warehouses.

The Cognos Dynamic Cubes solution consists of a modeling tool (IBM Cognos Cube Designer), a dynamic cube object in the administration environment that becomes the data source, and a wizard (Aggregate Advisor), started from IBM Cognos Dynamic Query Analyzer:

- IBM Cognos Cube Designer

  Cognos Cube Designer is a modeling tool that brings together best modeling principles from past successful modeling technology, with a modern and extensible architecture. The first step to deploying Cognos Dynamic Cubes is to model with the Cognos Cube Designer.

- IBM Cognos Dynamic Cubes server

  After a dynamic cube is designed and deployed, it becomes available in the Cognos environment and acts as the data source to the interface layer for dynamic cube applications. It manages all aspects of data retrieval, and leverages memory to maximize responsiveness while giving you full flexibility to manage what is contained in memory and when you want to refresh in-memory data. You manage dynamic cubes in the Cognos administration environment.

  A dynamic cube contains a number of in-memory elements to drive performance:

  – Metadata members

    When you start a cube, all members are loaded in memory. This approach provides a fast experience as users explore the metadata tree. This approach also helps the server to maximize its ability to run the most efficient queries possible.

  – Aggregates

    With Cognos Dynamic Cubes, you have the option to create in-memory aggregates. This way allows summaries or aggregates to be returned immediately when users navigate down from top-level summaries, such as sales per country, and start to drill down to lower levels, such as states. Each level requires a new total to be returned. When these totals or other aggregates are loaded in memory, they are returned instantly.

  – Data

    When users run the system and run queries, individual data items are loaded in memory. Because security is applied on top of the dynamic cube, the reuse of data is maximized, allowing as many users and reports as possible to benefit from previous queries.

  – Results sets

    When a report is run, it is saved in memory for later reuse. For commonly used reports, this way allows the engine to benefit from previous runs, giving users the fastest response time possible.

  – Expressions

    To accelerate query planning, the engine saves expressions that are generated when queries are run. In this way, query planning can be accelerated for future queries when expressions can be reused.

  With in-memory assets that can include aggregates, expressions, results, and data, Cognos Dynamic Cubes technology provides the tools to minimize user wait-times, ensuring the best possible user experience.

- Aggregate Advisor (part of IBM Cognos Dynamic Query Analyzer)

  After a dynamic cube is deployed in the environment, it becomes available for reporting and analysis. The server maintains logs to understand how the system is being used, and can use these logs to help optimize aggregates, both in-memory and in-database.

To create aggregates, you use the Aggregate Advisor, launched from Dynamic Query Analyzer (DQA), which is a diagnostic tool to help you better understand performance for Dynamic Query and Cognos Dynamic Cubes. The Aggregate Advisor scans cube definitions and usage logs, and recommends aggregates to improve performance. This approach helps address specific performance problems.

Figure 1-2 shows the lifecycle of a dynamic cube.

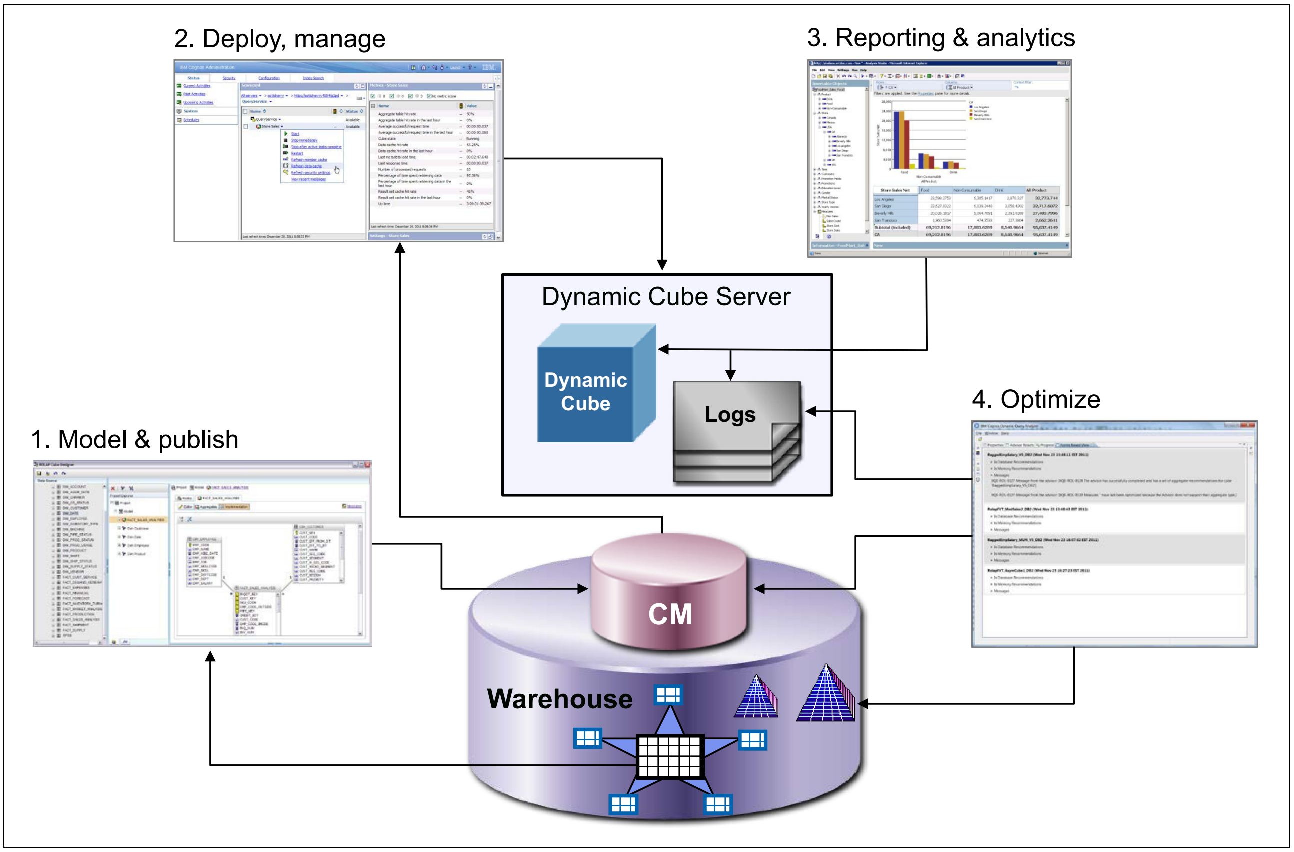


Figure 1-2 Dynamic cube lifecycle

1.2 Skill set requirements for IBM Cognos Dynamic Cubes

Developing and deploying Cognos Dynamic Cubes typically requires a team of individuals that collectively encompasses a range of knowledge and skills, including:

- ► Business intelligence modeler

  A dynamic cube needs to be modeled to meet the needs of the users and, in the case of a managed reporting and analysis application, the report authors. A knowledge of dimensional and data warehouse modeling is especially useful when modeling a dynamic cube, including concepts such as star and snowflake schemas.

- ► Business intelligence report author

  When developing a managed reporting and analysis application, it is important to understand the requirements of the reports and analyses being developed as the requirements affect the dimensional structures created in a cube model.

- Cognos system administrator

  The dynamic cube technology is included with IBM Cognos BI. Therefore, you must know how to administer the Cognos platform and its multitiered architecture to properly design and deploy a Cognos solution that includes Cognos Dynamic Cubes. For example, to set up a large Cognos Dynamic Cubes application along with other large relational applications, advanced routing rules must be set up, which requires advanced knowledge of the Cognos platform.

- Cognos security administrator

  In many environments, access to the data within dynamic cubes must be controlled based on the identity of users or their roles in an organization. This control requires the involvement of the Cognos security administrator. In addition, the dynamic cubes-related tasks available to users can also be secured, with different capabilities granted to users depending upon which environment they are working in, such as development, test, or production.

- Database analyst

  A dynamic cube is ultimately dependent on the performance of the underlying relational database and the availability of aggregate tables. As a result, it is important to have access to a DBA who is able to tune the database that underpins a dynamic cube, as well as building and maintaining any available aggregate tables. The BI modeler and the DBA must work closely together to produce a dynamic cube that delivers the required performance.

- System administrator

  During the development phase of dynamic cubes in a development environment, it is important for people in different roles to be able to access logs created in the file system of the server. The configurations of the Cognos server and the host server might need to be modified to accommodate the needs of dynamic cubes. System administrators perform these tasks.

1.3 When to use Cognos Dynamic Cubes

Cognos Dynamic Cubes was originally designed to enable interactive analysis with Cognos Business Intelligence on large volumes of data stored within a relational data warehouse. While that goal is still core to the design of dynamic cubes, over time it has become apparent that dynamic cubes are appropriate for a wide range of data volumes and situations.

Regardless of the situation in which dynamic cubes are employed, the underlying data must be stored in a star or snowflake schema within a relational database. This restriction is enforced by IBM Cognos Cube Designer. Although it is possible to model views within a relational database to make an arbitrary database schema appear as a star or snowflake, avoid this because many views mask complex SQL constructs that affect runtime query performance.

The list of relational database vendors supported by IBM Cognos Dynamic Cubes can be found on the web page *Cognos Business Intelligence 10.2.2 Supported Software Environments* at `http://www.ibm.com/support/docview.wss?uid=swg27042164`.

It is important that the underlying data warehouse be well tuned and make use of the facilities that are available to optimize query performance because at various points during the operation of a dynamic cube it must obtain data from the relational database.

Various capabilities of dynamic cubes individually or in combination address a wide range of use cases:

- Querying data warehouses with large volumes of dimension data, fact data, or both.
- Performing analysis on a combination of individual dynamic cubes.
- Support for relative time members, including custom relative time members that represent the time to date.
- Support for first and last semi-aggregation of measures.
- Controlling the refresh of members and data within individual dynamic cubes.
- Updating fact data within an active dynamic cube.
- Applying member, dimension, measure, and attribute security to cubes.
- Restricting fact data to what is related to members in one or more dimensions.
- Restricting dimension members to what is related to the available fact data.

1.4 System requirements

Cognos Dynamic Cubes attempts to take full advantage of available hardware, notably memory and CPU cores, to maximize performance over large volumes of data. Memory is used to cache the metadata and data of a cube, reducing the frequency with which a cube must retrieve data from the underlying relational database. Cognos Dynamic Cubes takes advantage of available cores by performing operations in parallel, breaking large tasks into smaller, parallel operations, and by creating execution pipelines to reduce memory consumption and execution time.

Therefore, it is common within a Cognos environment to explicitly dedicate servers to host one or more dynamic cubes, ensuring the cubes can take full advantage of the memory and cores available on the servers. Dedicating servers requires the use of routing rules to ensure that reports are routed to the appropriate servers.

IBM Cognos Cube Designer 10.2.2 includes a hardware sizing calculator that can be used to estimate the hardware required to host a dynamic cube. This tool is described in detail in Chapter 17, “Hardware sizing” on page 533. To estimate hardware requirements for more advanced use cases that are not covered by the hardware sizing wizard, see *IBM Business Analytics Proven Practices: Dynamic Cubes Hardware Sizing Recommendations* at:

`http://www.ibm.com/developerworks/library/ba-pp-infrastructure-cognos_specific-page659/index.html`

1.5 Comparison to other IBM cube technologies

Different data requirements require different data solutions. One data path cannot be proficient at solving all data problems. Therefore, IBM Cognos has technologies that are built to suit specific application requirements.

Table 1-2 lists the primary use case for each technology. However, carefully consider your individual application requirements when you make a decision.

Table 1-2 Use cases for Cognos Dynamic Cubes

Cube technology	Primary use cases
TM1 In-memory cube technology with write-back support	► Is optimal for write-back, what-if analysis, planning and budgeting, or other specialized applications. ► Can handle medium data volumes. Cube is run 100% in memory. ► Aggregation occurs on demand, which can affect performance with high data and high user volumes.
Dynamic Cubes In-memory accelerator for dimensional analysis	► Is optimal for read-only reporting and analytics over large data volumes. ► Provides extensive in-memory caching for performance, backed by aggregate awareness to use the power and scalability of a relational database. ► Star or snowflake schema is required in underlying database (use to maximize performance).
PowerCubes File-based cube with pre-aggregation	► Is optimal to provide consistent interactive analysis experience to large number of users when the data source is an operational or transactional system, and a star or snowflake data structure cannot be achieved. ► Pre-aggregated cube architecture requires careful management using cube groups to achieve scalability. ► Data latency is inherent with a pre-aggregated cube technology, where data movement into the cube is required.
Dimensionally Modeled Relational (DMR) Dimensional view of a relational database	► Is optimal for creating a dimensional data exploration experience over low data volumes in an operational or transactional system, and where latency must be carefully managed. ► Caching on the dynamic query mode server helps performance. ► Processing associated with operational or transactional system affects performance.

1.6 Overview of the remainder of the book

This book is intended as a one stop shop for everything you need to know to be successful with IBM Cognos Dynamic Cubes.

Chapter 2, “IBM Cognos Dynamic Cubes architecture” on page 13 provides an in-depth analysis of the architecture of Dynamic Cubes that helps you understand how the topics in the remainder of the book tie together.

Chapter 3, “Installation and configuration of IBM Cognos Cube Designer and IBM Cognos Dynamic Query Analyzer” on page 65 describes how to install and configure Cognos Dynamic Cubes.

Chapter 4, “Modeling with IBM Cognos Cube Designer” on page 89, Chapter 5, “Basic modeling” on page 105, and Chapter 6, “Advanced topics in modeling” on page 159 cover a variety of topics that are related to modeling dynamic cubes in IBM Cube Designer.

Chapter 7, “Administering dynamic cubes” on page 199 describes how dynamic cubes are administered.

Chapter 8, “Virtual cubes” on page 223 describes how virtual cubes are modeled and common use cases for which virtual cubes are applicable.

Chapter 9, “Dimensional security” on page 255 provides a thorough description of topics that are related to security, including member security rules, table-based security filters, and dimension, measure, and attribute security.

Chapter 10, “Securing the IBM Cognos BI environment” on page 321 describes how to apply security to user accounts within the Cognos environment to control the dynamic cubes operations that individuals can perform. These include publishing a dynamic cube to the content store and running the IBM Cognos Dynamic Cubes Aggregate Advisor.

Chapter 11, “Optimization and performance tuning” on page 341 discusses various topics that are related to optimizing and tuning dynamic cubes.

Chapter 12, “Near real-time updates” on page 427 describes the near-real time updates feature of dynamic cubes. It describes how updates to a fact table can be applied to an active dynamic cube.

Chapter 13, “Dynamic cube lifecycle” on page 445 provides a description of the lifecycle of a dynamic cube.

Chapter 14, “Troubleshooting” on page 461 provides an in-depth set of techniques for troubleshooting problems that can arise with a dynamic cube.

Chapter 15, “Query service memory monitoring” on page 509 describes the memory management facilities within the dynamic query mode server and how they relate to dynamic cubes.

Chapter 16, “Using Framework Manager models” on page 519 describes how Framework Manager models can be imported into IBM Cognos Cube Designer as the basis for a dynamic cube model.

Chapter 17, “Hardware sizing” on page 533 describes how to estimate hardware sizing for dynamic cubes, including the use of the IBM Cognos Cube Designer hardware wizard.

Chapter 18, “Dynamic cubes in a multi-dispatcher environment” on page 543 describes how Cognos Dynamic Cubes operates within a multi-dispatcher environment.

2

IBM Cognos Dynamic Cubes architecture

IBM Cognos Dynamic Cubes adds relational online analytical processing (ROLAP) capabilities to IBM Cognos dynamic query mode (DQM) in version 10.2.2 of IBM Cognos BI, enabling reporting and ad hoc analysis on multi-terabyte data volumes.

Cognos Dynamic Cubes uses its shared in-memory member and data caches, and its awareness of in-memory and in-database aggregates, to provide fast query performance for a large number of users.

This chapter contains the following sections:

- Dynamic cubes in the IBM Cognos BI environment
- Dynamic cubes and the query service
- Cognos Dynamic Cubes caching
- Dynamic cubes query processing
- Using the database to evaluate set expressions
- Management of Cognos Dynamic Cubes

2.1 Dynamic cubes in the IBM Cognos BI environment

Dynamic cubes are in-memory OLAP containers that are inside the JVM of a query service.

Dynamic cubes are sourced from a star or snowflake data warehouse and the associated aggregate tables are stored in any one of the supported relational databases. For the list of data sources that support Cognos Dynamic Cubes, see:

https://www-01.ibm.com/support/docview.wss?uid=swg27042164

Unlike other OLAP sources available to IBM Cognos BI, dynamic cubes are entirely inside the Cognos BI environment.

To understand the architecture of Cognos Dynamic Cubes, first consider the IBM Cognos BI environment, which consists of a content store (CS) database and the following installed components:

- A gateway
- One or more Application Tier Components (ATCs)
- A Content Manager (CM)

The Application Tier Components of interest here are the dispatcher (often referred to as a server), the report server, and the query service.

Figure 2-1 shows multiple computing nodes, each hosting Cognos report servers and a query service. Each query service supports one or more dynamic cubes which in turn access a relational data warehouse.

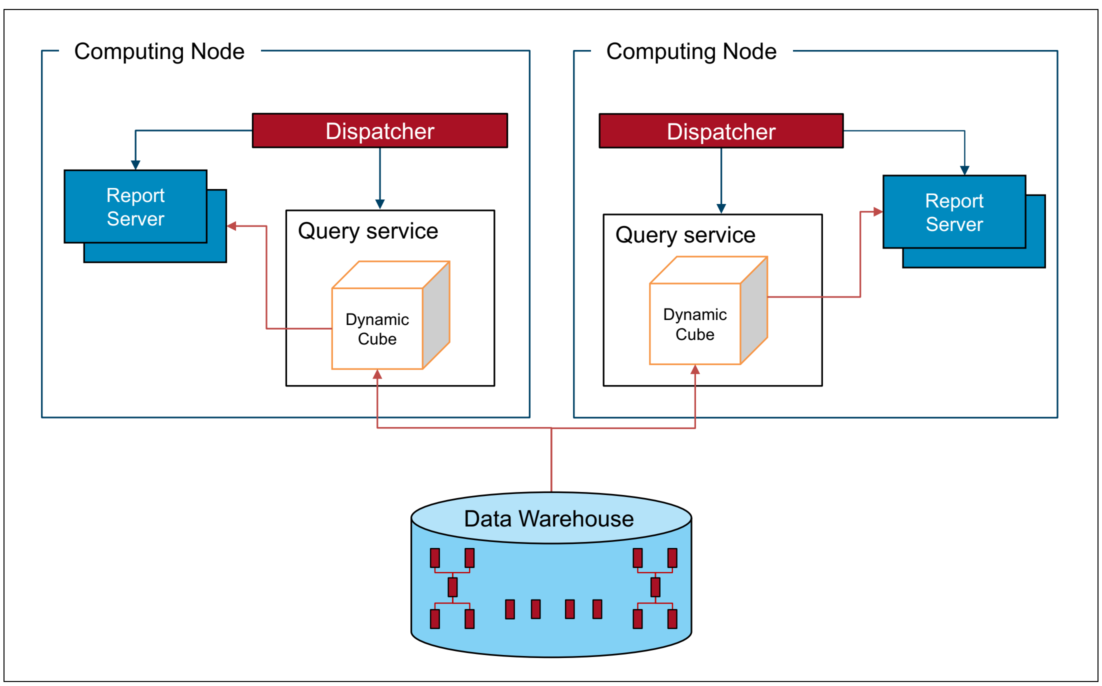


Figure 2-1 High-level representation of the Cognos Dynamic Cubes architecture

The dynamic cube itself can be considered from two perspectives. The first perspective encompasses all the objects that comprise a published cube, where those objects are, and how they are accessed. The second perspective is how the instances of a cube are configured and managed.

Consider the first perspective and look at a published dynamic cube. Each dynamic cube in the Cognos BI environment has an infrastructure that is composed of the following CS objects:

- A dynamic cube data source connection
- Credentials to access the underlying source relational database (access account property of a dynamic cube data source connection)
- A model
- Security views
- In-memory aggregate specifications
- Dynamic cube configurations

Figure 2-2 shows the components of a dynamic cube in the content store and associated packages and reports that are also in the content store.

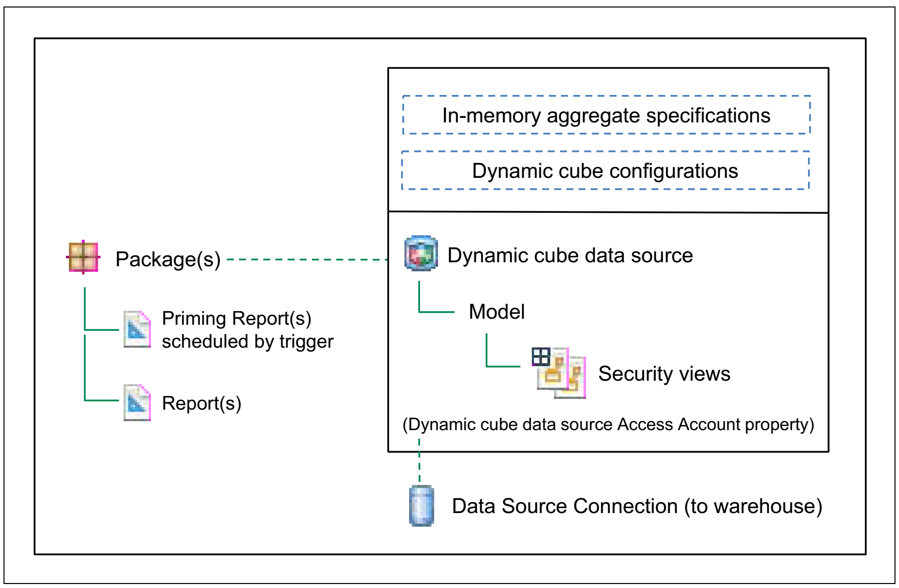


Figure 2-2 Components of base dynamic cube in content store with associated package and reports

> **Note:** There are two types of dynamic cube: Base cubes and virtual cubes. Each type of cube has a different icon and infrastructure. These differences are discussed in 2.1.4, "Dynamic cubes as Cognos BI data sources" on page 30 and in Chapter 8, "Virtual cubes" on page 223. The rest of this section focuses primarily on base cubes.

A published cube is presented in Cognos Administration as a dynamic cube data source in the list of CS data source connections. The cube model is visible as a child of the dynamic cube data source, and any security views defined in the cube are visible below the model.

The access account property of the dynamic cube data source provides the model with valid credentials for retrieving data from the source data warehouse.

Two other CS components that are associated with the dynamic cube are the cube configuration properties for each cube instance and the in-memory aggregate specifications that have been applied to the cube instances.

Packages make the dynamic cube available to the Cognos BI studios and include a reference to the dynamic cube data source object. A dynamic cube can be presented in its own package or be presented as part of a multi-data source package.

There is also a reliance on a data source connection to the source relational data warehouse of the dynamic cube. This component provides connectivity to the underlying cube source. It is used by Cognos Dynamic Cubes throughout the dynamic cube lifecycle, from modeling the cube to populating the running cube instances and servicing data requests from the Cognos BI user interfaces.

Next, consider the second perspective of a dynamic cube, namely the configuration of its various instances and how they are managed. After the objects that make up the cube are published to the content store, the cube is added to one or more dispatchers. Adding a cube to a dispatcher involves adding the cube configuration information to the query service of the dispatcher that hosts the cube. The actual cube instance that is queried by the report server is in the memory (Java heap) of the associated query service. A single published cube can have multiple running instances. Multiple running instances of a single published cube are achieved by adding the cube configuration to more than one query service.

So, looking at the communication between the query service and the report server, the query service manages dynamic query requests and returns the results to the batch or report service that sent the request. Each dispatcher with the report service, the batch report service, or both services enabled can create a configurable number of instances of the report server. However, each dispatcher with the query service enabled can create only one query service. Therefore, requests against the cube instances on a single dispatcher are all processed by the same query service instance. Additionally, although there can be multiple instances of a dynamic cube available in the Cognos BI environment, there is only one instance of each cube per dispatcher.

Figure 2-3 shows two cubes, A and B, configured and running on the same dispatcher. An example of two instances of the same cube is shown in Figure 2-4 on page 17.

Note: Figure 2-3 shows a conceptual diagram only. The cube configurations, although added to the query service of a dispatcher, are not actually stored there. The cube configurations are stored in the CS.

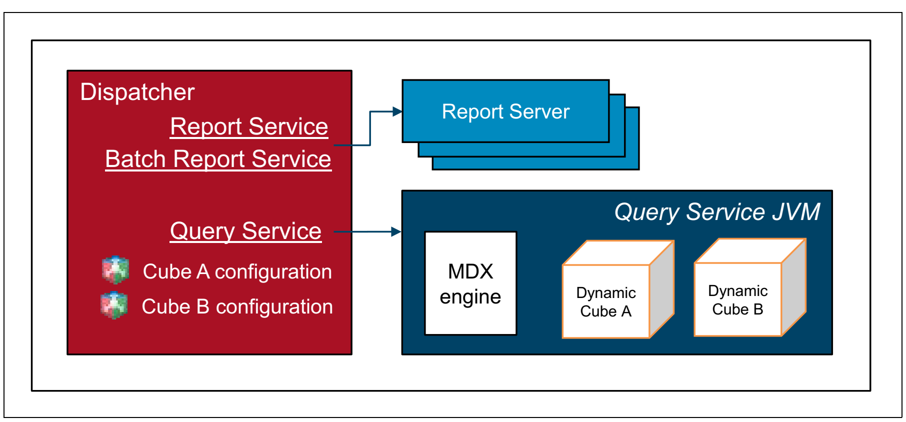


Figure 2-3 Dynamic cubes configured and running on the same dispatcher

Figure 2-4 shows a detailed representation of the Cognos Dynamic Cubes architecture including the cube components in the CS, the cube configurations, and two running instances of the same cube.

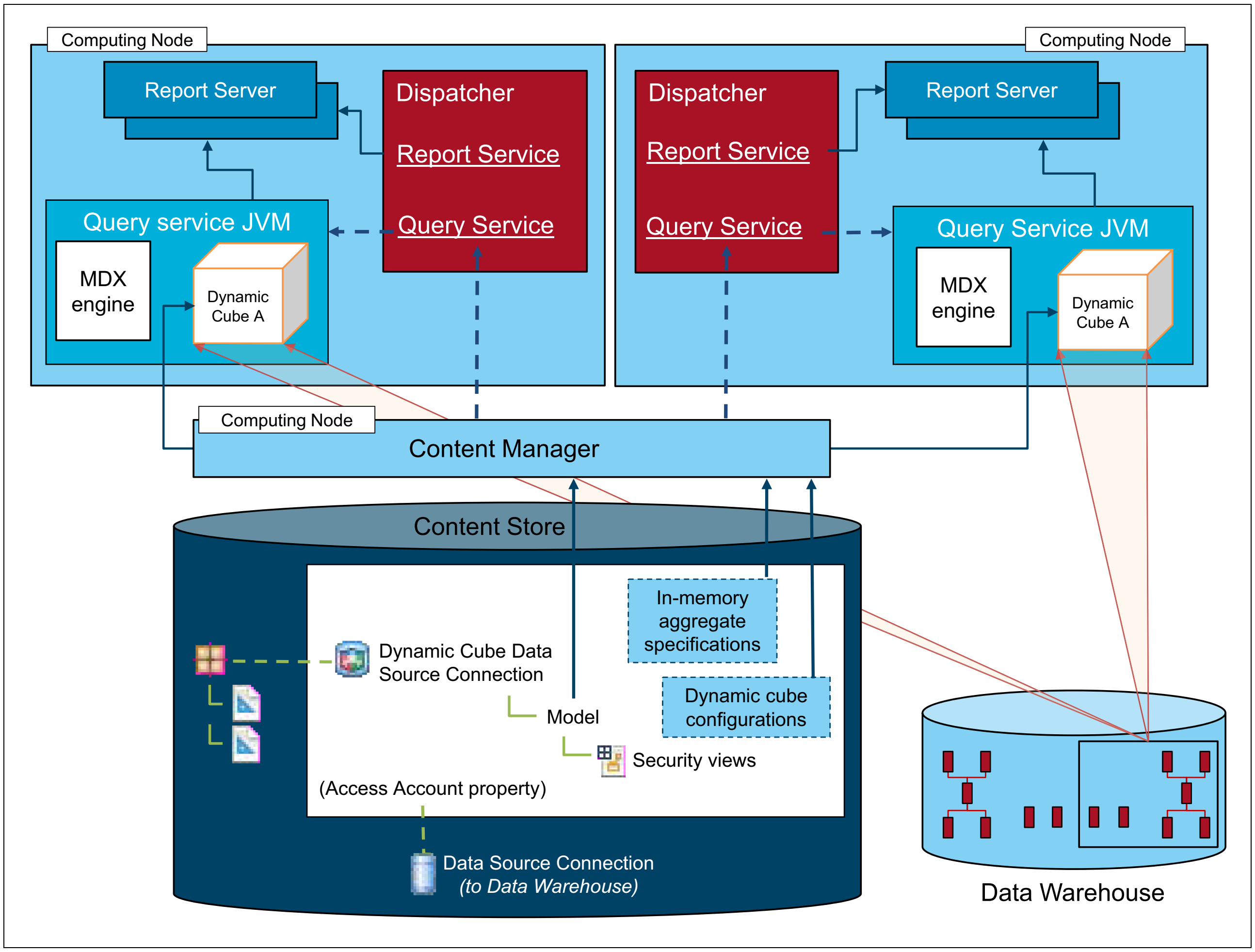


Figure 2-4 Detailed representation of the Cognos Dynamic Cubes architecture

The dispatcher communicates with the query service when the cube is started, as shown by the dashed arrows in Figure 2-4. From this point on communication is between the query service and the report server, as shown in Figure 2-1 on page 14. The running dynamic cube instance in the query service JVM is provided with information from the CS through the CM and populated with data from the data warehouse. Each cube instance is populated by using a separate set of queries issued in this manner to the data warehouse.

The lifecycle of a dynamic cube is described in Chapter 13, "Dynamic cube lifecycle" on page 445, but it can be briefly summarized as follows:

1. Dynamic cubes are modeled in IBM Cognos Cube Designer. After modeling is complete, the modeler publishes the cube to the CS as an IBM Cognos Dynamic Cubes data source. The result is visible as a new dynamic cube data source in the list of data source connections that are displayed in IBM Cognos Administration. The associated model and security views are visible below the dynamic cube data source object.
2. Each dynamic cube must be assigned to at least one dispatcher so it can be configured and managed. Cubes can be assigned to a dispatcher either from Cognos Cube Designer when publishing a cube or later from the Status or Configuration tabs in Cognos Administration.

From Cognos Administration, an administrator can assign a dynamic cube to one or more dispatchers and configure the properties of the cube on each dispatcher. An example of configuring a cube is to specify the amount of space that is allotted for in-memory aggregates available to an instance of a cube. The associated query service might also need to be configured to accommodate the cubes that it is hosting. An example of configuring the query service is to adjust tuning settings for the JVM heap and cube administration.

3. Security for Cognos Dynamic Cubes is defined during the modeling stage and implemented during the configuration stage for each published cube. The security views defined by the cube modeler are visible under the child model object of the data source connection in Cognos Administration. Implementing security involves the process of adding extra groups and roles to the access list of each security view and setting the permissions. Cognos Dynamic Cubes security can also include the management of database tables referenced by table-based security filters that are modeled in the cube.
4. As with any other data source to the Cognos BI environment, a package must also be created to make the cube visible to the reporting and analysis interfaces. Packages containing a dynamic cube can be created during the publish process from Cognos Cube Designer or later using Framework Manager.
5. After a cube is assigned to a dispatcher and configured, it can be started. On cube startup, the query service retrieves the cube metadata from the CS. The data that is required to populate all dimensions of the cube is retrieved from the relational database and the query service loads all dimensional members into a member cache. This process constructs an in-memory representation of the cube hierarchies. The cube is then available for processing of reports and analyses. If in-memory aggregates are defined, they start loading from the moment the cube is available.
6. Triggers can be defined to control the timing of data loads to a cube data cache. There are two types of triggers:
   - Post cube start

     This trigger causes associated priming reports to run after the cube is started.
   - Post loading of in-memory aggregates

     This trigger causes the associated priming reports to run after the in-memory aggregates are fully loaded, ensuring they are available to the priming reports.

   Both types of triggers are defined in the cube configuration and must be set for each instance of a cube.

Figure 2-5 shows a cube being started, becoming available, and loading in-memory aggregates.

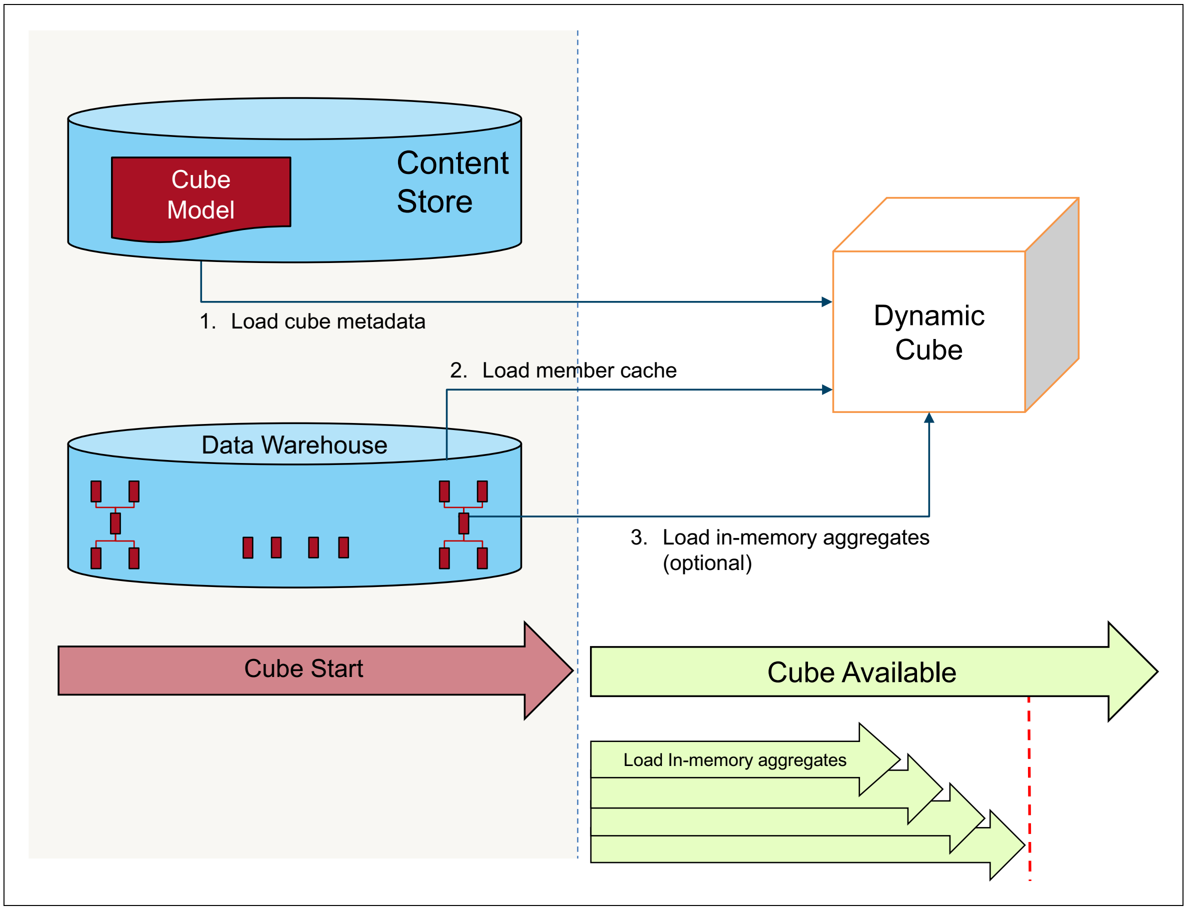


Figure 2-5 Starting a dynamic cube

Dynamic cubes have many important features and capabilities that are realized through the effective coordination of database administrators and designers, business intelligence modelers, Cognos administrators, and report authors.

This section provides an overview of the various aspects of IBM Cognos Dynamic Cubes in an attempt to show how dynamic cubes are designed and optimized, and the associated activities for ensuring optimal performance. The sections that follow focus specifically on dynamic cubes and the query service, caching, query processing, set expression evaluation, and dynamic cube management.

2.1.1 Dynamic cubes and the data warehouse

A dynamic cube provides Cognos BI users with OLAP access to a star or snowflake schema, and optionally to associated aggregate tables, in a relational database. Hence, the data source for a dynamic cube is a data warehouse. IBM Cognos Dynamic Cubes does not support data warehouses that do not conform to the star or snowflake topology, nor do they provide access to complex transactional databases.

Cognos Dynamic Cubes can take advantage of the security defined in star or snowflake data warehouses that implement row level security as part of their design. This is achieved by the process of the cube modeler adding security filters that translate this row level security into the cube. For more information about this approach, see 9.9, “Security lookup tables” on page 288.

The goal of IBM Cognos Dynamic Cubes is to provide quick response to reports and analyses on large volumes of data. To accomplish this goal, it is important that the queries that are posed to a relational database conform to a simple set of known patterns that are easily optimized by relational databases. Star and snowflake topologies are well documented as the best approach for modeling data warehouses, and are supported with specific query optimization by nearly all major database vendors.

As with any other relational data source accessed in the Cognos BI environment, a data source connection must be defined in Cognos Administration to enable access to the data warehouse that contains data for the cube. In addition, JDBC connectivity must be configured for the data source connection (required for all relational data sources that are accessed by the query service). This data source connection is used by both the Cognos Cube Designer tool to retrieve metadata and data during the modeling process, and by the query service to populate the published dynamic cube and satisfy query requests against it.

Figure 2-6 shows the data source connection with JDBC connectivity enabled.

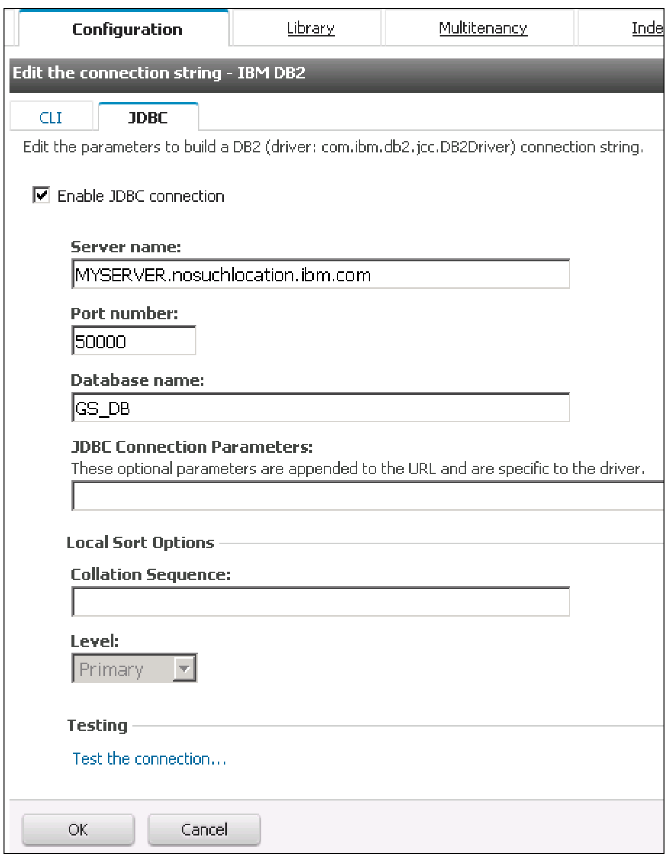


Figure 2-6 Data source connection with JDBC connectivity enabled

In addition to a star or snowflake schema, Cognos Dynamic Cubes is able to take advantage of aggregate tables in the relational database. Aggregate tables are any relational tables that contain pre-aggregated data representing the measure values from the fact table of a star or snowflake schema, summarized by members from one or more dimensions to a level of aggregation higher than that represented in the fact table.

If a dynamic cube has in-database aggregates defined, the query service can use aggregate tables to improve query performance, where appropriate, by routing queries for measure values to these tables rather than the detail fact tables.

Figure 2-7 shows routing of a query to a summary table.

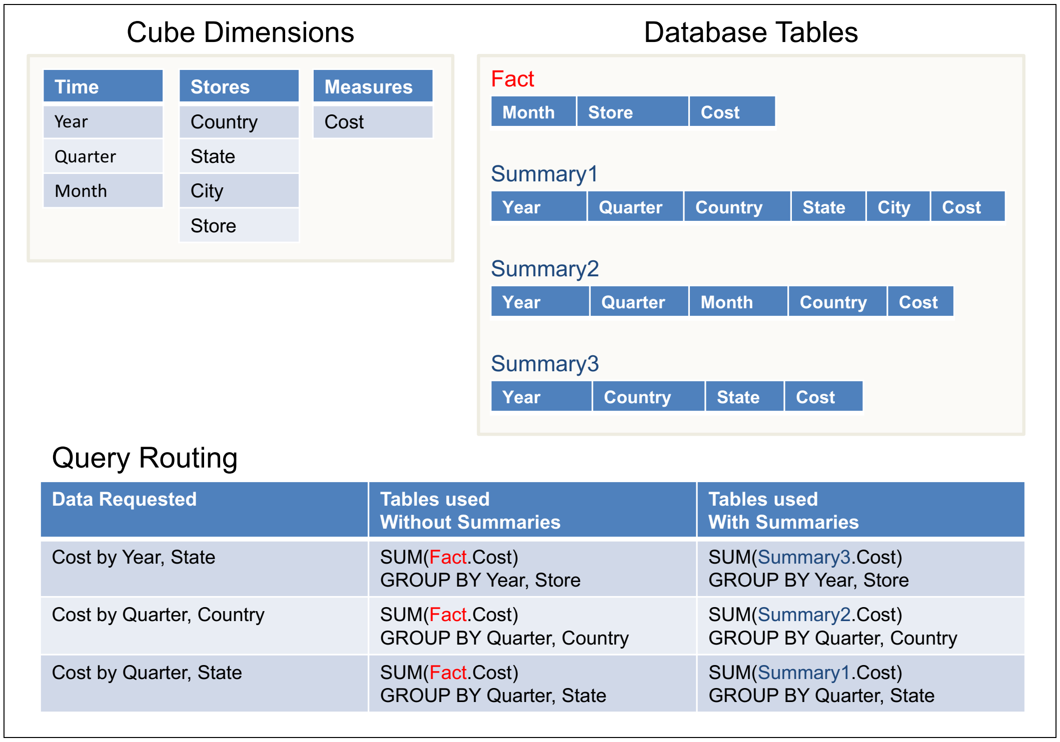


Figure 2-7 Routing a query to a summary table

In addition to this in-database aggregate approach, Cognos Dynamic Cubes can promote query performance by using in-memory aggregates. In-memory aggregates are discussed in more detail in 2.1.7, “IBM Cognos Dynamic Cubes Aggregate Advisor” on page 39 and 2.2.3, “Dynamic cubes in the query service JVM” on page 44.

2.1.2 Modeling dynamic cubes

Dynamic cubes are modeled using IBM Cognos Cube Designer. The tangible asset that is created by Cognos Cube Designer is a single project file, `*.fmd`. The project file stores all the information required to publish a dynamic cube to the Cognos BI environment and must be retained after publishing to modify and tune the cubes defined inside. Multiple cubes can be modeled in a single Cognos Cube Designer project.

Cognos Cube Designer uses the Get Metadata feature to dynamically browse the structure of the source database and present objects that can be used in the cube model. An important distinction between Cognos Cube Designer and Framework Manager (FM) is that with Framework Manager, the metadata of the physical source must be imported into the FM project file before it can be accessed by the FM modeler. The act of retrieving and browsing source metadata from within Cognos Cube Designer, however, does not change the Cognos Cube Designer project. The metadata of the physical source is not actually imported into the Cognos Cube Designer project until the modeler selects an object from the metadata pane and adds it to a model.

The Get Metadata feature can access the star or snowflake structure of the data warehouse and any associated aggregate tables required by a cube model either directly or indirectly:

- Directly by browsing the metadata of the physical source using a CM data source to view the structure of objects as they exist in the database.
- Indirectly by using the modeled structures available in a published Framework Manager (FM) package.

All objects in the FM package are presented in the Cognos Cube Designer metadata pane. However, objects that cannot be used by Cognos Cube Designer, such as stand-alone filters and calculations, cannot be imported into the Cognos Cube Designer project.

When the modeler selects an object presented directly in the Cognos Cube Designer metadata pane and adds it to the Cognos Cube Designer model, the physical source metadata for that object is imported into the Cognos Cube Designer project file. When an object is selected from an FM package (presented indirectly), Cognos Cube Designer references the FM package for the metadata of the selected object, resolves it down to the required physical source objects, and imports the relevant physical source metadata into the Cognos Cube Designer project file.

Figure 2-8 shows source objects presented directly and indirectly in the metadata pane in Cognos Cube Designer.

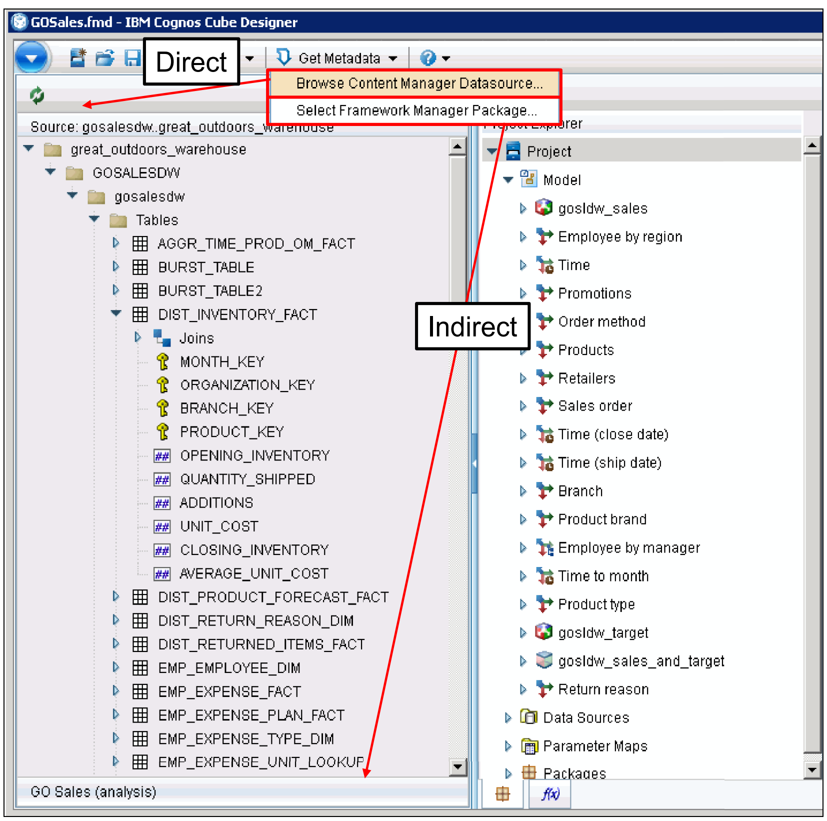


Figure 2-8 Get Metadata and the metadata pane in Cognos Cube Designer

IBM Cognos Dynamic Cubes supports the following features, all of which are modeled by using Cognos Cube Designer:

- Dimensions
- Multiple hierarchies per dimension

- Named Levels
- Level attributes
- Parent-child (recursive) hierarchies
- Ragged (unbalanced) hierarchies
- Time hierarchies, including automatic creation of relative time members and custom relative time members
- Calculated (dimensional) members and measures
- Member, measure, dimension, and attribute security
- In-database aggregates
- User-defined in-memory aggregates
- Virtual cubes
- Named sets
- Dimension and measure filters

A Cognos Cube Designer project, by default, has a single model namespace, but can also contain multiple model namespaces. The model namespace encapsulates all the metadata objects for a single model. The root level objects inside the model are the cubes and dimensions. Each dynamic cube is based on a set of measures, a collection of dimensions and hierarchies, and in-database aggregates specifying aggregate tables or user-defined in-memory aggregates. All objects that make up a cube are encapsulated inside a parent cube object.

A collection of dimensions can be defined once inside a specific model and then used in one or more of the cubes. Reused dimensions are presented as shortcuts under the cube objects. Alternatively, dimensions that are specific to a particular cube can be defined below the cube itself rather than at the model root where other dimensions available for reuse across the model are. Dimension folders can be used to group these objects and further reduce clutter.

The concept of reuse as seen with dimensions and hierarchies does not extend to measures. Measures cannot be reused across cubes because each cube must reference a single fact table and as such, each group of measures is specific to a particular cube.

Figure 2-9 shows metadata objects in Cognos Cube Designer.

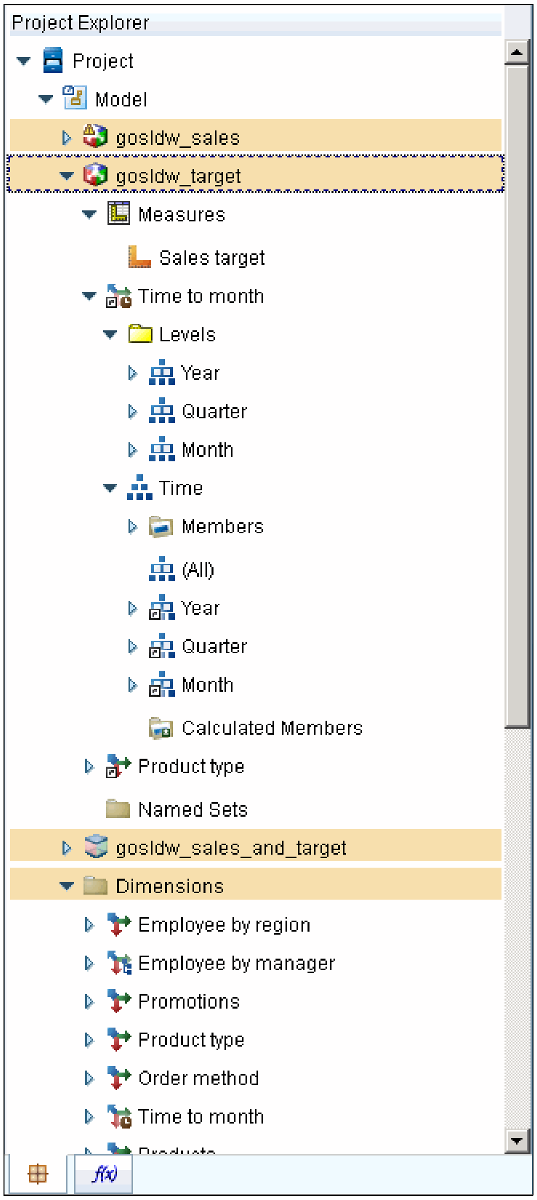


Figure 2-9 Metadata objects in Cognos Cube Designer

Cognos Cube Designer can also be used to model virtual cubes. Virtual cubes are defined in Cognos Cube Designer as the merger of two existing cubes into a new cube. The resulting virtual cube represents the merging of the members and data of these two source cubes, either of which can be a *base dynamic cube* or another virtual cube. This feature enables modelers to define a cube that is effectively sourced from two fact tables. In this manner, it is possible to model multi-fact cubes that span fact tables of differing granularity or different business areas of interest. For example, with a virtual cube, users can analyze a combination of daily sales and monthly inventory information.

Figure 2-10 shows the organization of a virtual cube, merging two source cubes into a new virtual cube. In this example, both source cubes are base cubes, each referencing a different fact table in the database.

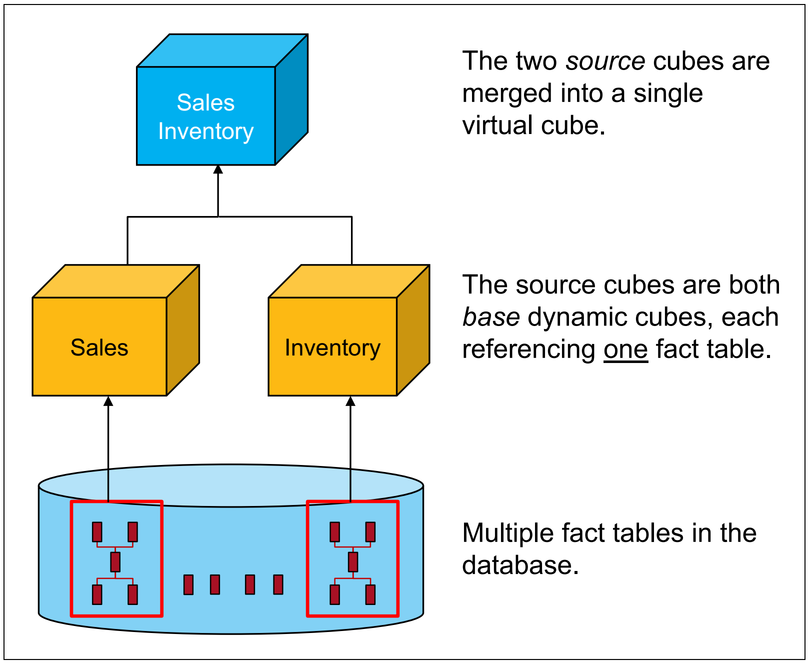


Figure 2-10 Organization of virtual cubes

When creating a virtual cube, the objects (dimensions, hierarchies, levels, members, properties, and measures) from two original cubes are merged. This merge can be performed automatically or on an object-by-object basis by using a manual process. A manual merge gives the modeler more control over the merge behavior and mappings between the original objects. A merge rule can be defined to control how measure values common to both base cubes are merged. For example, the quantity from the Sales cube can be merged with a negative value to subtract it from the quantity from the Inventory cube. This merge ruleallows the quantity in the resulting Sales Inventory cube to reflect the available inventory balances.

Cognos Cube Designer assists the cube modeler by analyzing the cube model as it is being constructed to ensure that the underlying data warehouse is a true star or snowflake schema. Cognos Cube Designer raises exceptions if it recognizes relationships in the data that contradicts constraints of either data topology. Cognos Cube Designer also includes various features to assist cube modelers in their task of developing and maintaining dynamic cubes.

These features include the following:

- Member browser
- Validate
- Estimate Hardware Requirements
- Publish and configure a cube in one step

Member browser is the feature that allows a modeler to view the members in a hierarchy from inside Cognos Cube Designer. This feature can be used to validate the structure of a hierarchy as it is being modeled. It is important to note that although the member browser parses calculated expressions for syntactic correctness, it does not actually run the expressions or evaluate them for runtime semantic correctness.

Figure 2-11 shows viewing members in the Cognos Cube Designer member browser.

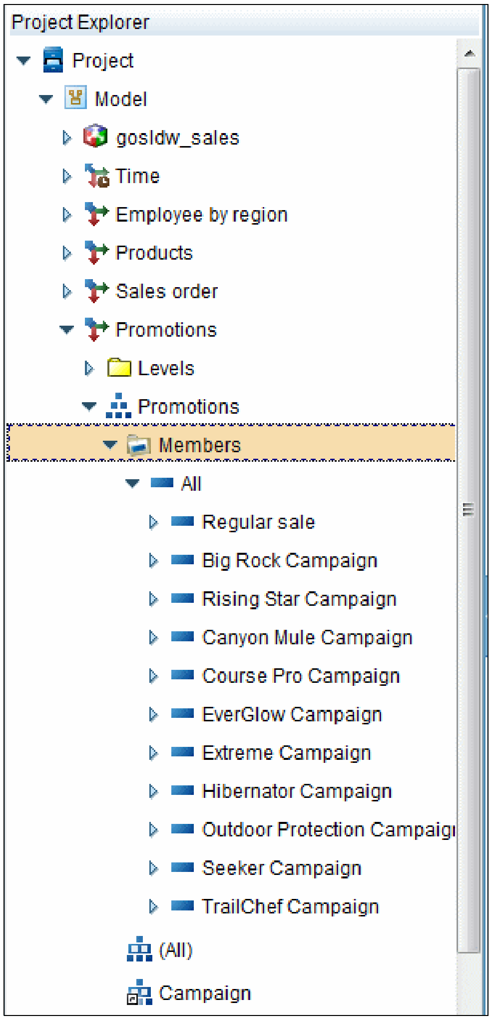


Figure 2-11 Viewing members in the Cognos Cube Designer member browser

Note: The execution of expressions defined in Cognos Cube Designer does not occur until after a cube is published. An example can be seen with a relative period expression that references the system date as the starting point for its calculation.

The member browser displays a member tree populated with relative time members based on the rightmost member in the lowest level of the hierarchy. The populated member browser does not indicate that the underlying database has records that correspond to the required point in time.

If the published cube is populated with dates that are all earlier than the system date, then no data matches when the relative time period expression is evaluated and the cube does not contain any relative time members. For more information, see 6.2, “Time dimensions and relative time” on page 168.

Validate is a feature that allows a modeler to validate all or part of a cube model to identify modeling problems as they occur. Resulting issues and performance warnings are displayed in the Properties pane when the model or a metadata object is validated. Major issues must be resolved before a cube can be published.

Figure 2-12 shows how to validate an object in Cognos Cube Designer.

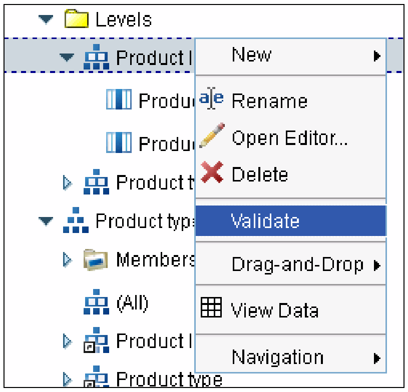


Figure 2-12 Validating an object in Cognos Cube Designer

Figure 2-13 shows a Cognos Cube Designer validation error message and a performance issue warning message.

> **Note:** Clicking an error message in the Issues window adds a link that can be used to open the editor of the object to which the error relates. Clicking the Performance Issues tab displays the list of performance issues.

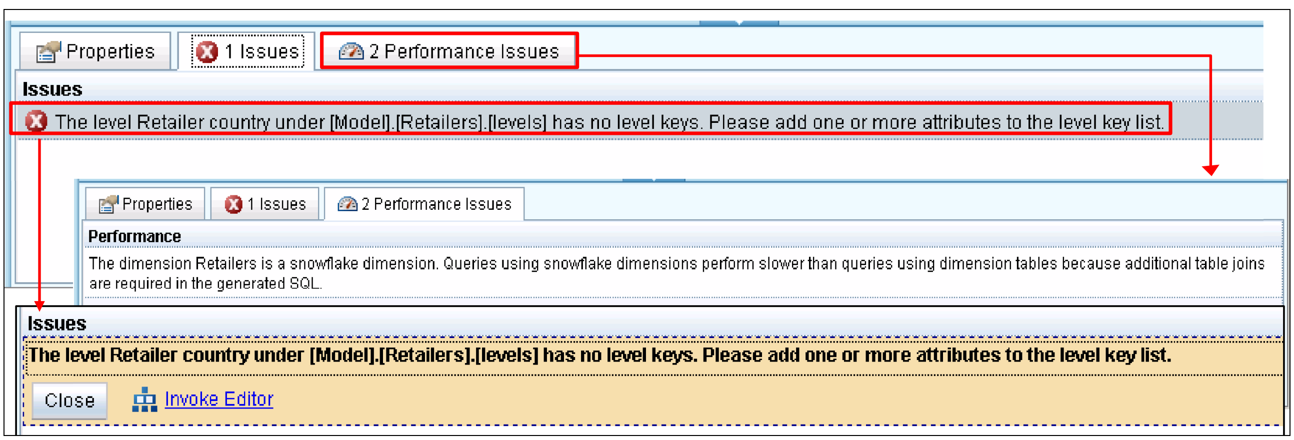


Figure 2-13 Cognos Cube Designer issues: Validation error message and performance warning message

Estimate Hardware Requirements, also known as *hardware sizing calculator*, is a feature that generates estimates for hardware resources that are required to support a dynamic cube. Based on the number of members in the selected cube and the number of concurrent queries per report, this tool produces estimates for memory size, number of processor cores, and hard disk space required to support a running instance of the selected cube and estimated values for the cube's configuration settings such as cache size limits. Estimates provided by the Estimate Hardware Requirements tool are for a single instance of a cube and do not consider multiple locales or shared dimensions. This feature is not available for virtual cubes. See Chapter 17, “Hardware sizing” on page 533 for more information about this feature.

Figure 2-14 shows the Estimate Hardware Requirements feature.

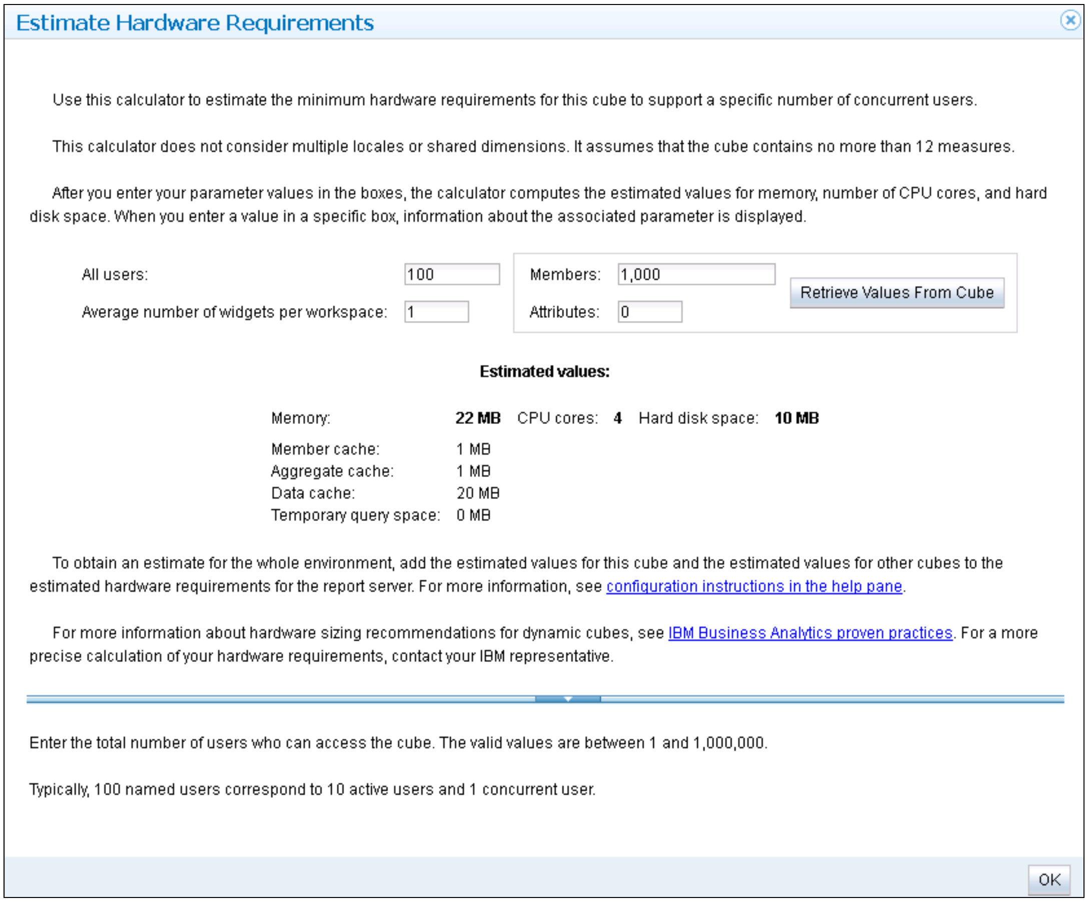


Figure 2-14 Hardware Sizing Calculator

The *Publish* dialog includes an option to publish a cube and perform extra required tasks in a single step.

Clicking the arrow to the left of Additional Options in the Publish window, as shown in Figure 2-15 on page 29, reveals the extended publishing options. The **Select all options** check box can then be used to select the check boxes for all the additional options. By selecting this check box, the modeler can publish a cube, set the access account property, add the cube to a dispatcher, create a package, and start the cube, all from a single click. This feature assists modelers working in a development environment where the expected (and frequently repeated) cycle is to model, publish, test, and remodel.

Figure 2-15 shows the Cognos Cube Designer Publish dialog with all additional options selected.

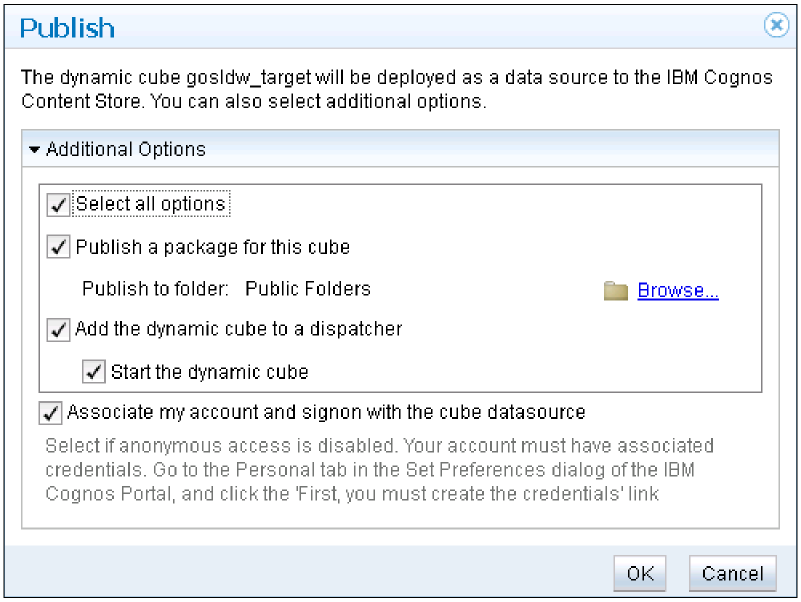


Figure 2-15 Cognos Cube Designer Publish options

See Chapter 3, “Installation and configuration of IBM Cognos Cube Designer and IBM Cognos Dynamic Query Analyzer” on page 65 for information about installing and configuring Cognos Cube Designer.

See Chapter 4, “Modeling with IBM Cognos Cube Designer” on page 89, Chapter 5, “Basic modeling” on page 105, and Chapter 6, “Advanced topics in modeling” on page 159 for detailed information about modeling with Cognos Cube Designer.

2.1.3 The published dynamic cube model

A published dynamic cube exists as two objects in the content store:

- A data source connection object

  The data source connection object contains the information required to connect to the cube to retrieve data from it.

- A model object.

  The model object contains the information required to create an instance of the cube in the memory space of the query service that is hosting it.

After a cube is published, it can be configured on one or more dispatchers to achieve multiple running instances for scalability and load-balancing purposes. In addition, multiple packages can be created to provide various audiences access to the cube. Therefore, the published cube can be referenced by multiple packages and have multiple running instances in the Cognos BI environment. All instances of a dynamic cube are potentially available to each package and access to a cube instance is determined by routing rules.

To understand this better, it is helpful to look at packages in general and then specifically at package interaction with a dynamic cube. A package is an object that combines metadata from the required data sources and a reference to the CM data source connections that provide connectivity to each source. In the case of a package that references an OLAP data source, this metadata is restricted to high-level information only such as the name of the required cube and how to identify specific features required by Cognos BI that are not natively provided by the cube model. For example, a package that contains a PowerCube includes the name of the cube and a reference to the relevant CM data source connection. A package that includes a TM1 cube also includes the name of the measures dimension and, optionally, the names of the time dimensions and the alias to be used for each dimension.

Cognos BI packages can include metadata from a range of data sources including relational databases and various OLAP data sources. For most OLAP data sources, it is easy to see how multiple packages can point to a single cube because most cube types are outside the Cognos BI environment. The possibility of multiple packages pointing to a single cube is less clear for dynamic cubes because these cubes and the associated cubing engine are entirely contained inside the Cognos BI environment. In the case of a package that includes a dynamic cube, all the information required by Cognos BI is supplied by the CM data source connection that references the published cube. The only difference is that the published cube is not an external entity and Cognos Dynamic Cubes updates all cubes included in the package.

When a package contains cube types other than a dynamic cube, the tool performing the publish action creates a package object in the content store or updates an existing one. In the case of a multi-cube package created in Cognos Cube Designer, the publish action also validates all the cubes that are referenced in the package and updates the published cube. Therefore, the action of publishing a package that includes references to existing published cubes requires you to update any cubes that have already been published.

The implication of a published cube being referenced by multiple packages is demonstrated in a Cognos Cube Designer warning message that can occur when publishing packages defined in Cognos Cube Designer.

Consider the following scenario:

1. Various cubes have been modeled in Cognos Cube Designer and each cube has been published previously.
2. A new cube is added to the model and a new package that includes the new cube and one of the existing cubes is defined. The new package might be used to satisfy a requirement to include data from both cubes in a single report.
3. When the new package is published, a warning message is displayed indicating that the older, previously published cube already exists and the modeler is prompted to update it.

This message demonstrates the following two points:

- There can be multiple packages in the content store, each with a reference to the same cube. However, there can only be one associated published cube.
- Publishing a package that includes references to multiple cubes requires you to republish all the cubes in the package even if they have been previously published.

2.1.4 Dynamic cubes as Cognos BI data sources

When a dynamic cube is published to the content store from Cognos Cube Designer, it appears in the list of data source connections in Cognos Administration. There are now three types of data source connections: connections to external data sources and two types of data source connections for dynamic cubes. These are *dynamic cube data sources* that reference base cubes and *dynamic virtual cube data sources* that reference virtual cubes. Each type has a specific icon to identify it and distinguish it from other data sources that are defined in the content store.

Figure 2-16 shows two dynamic cube data sources, `gosldw_sales` and `gosldw_target`, and a dynamic virtual cube data source connection, `gosldw_sales_and_target`, among other connections in Cognos Administration. Notice that the virtual cube data source has a different icon, and that the virtual cube data source name is in black, indicating that it is not a hyperlink.

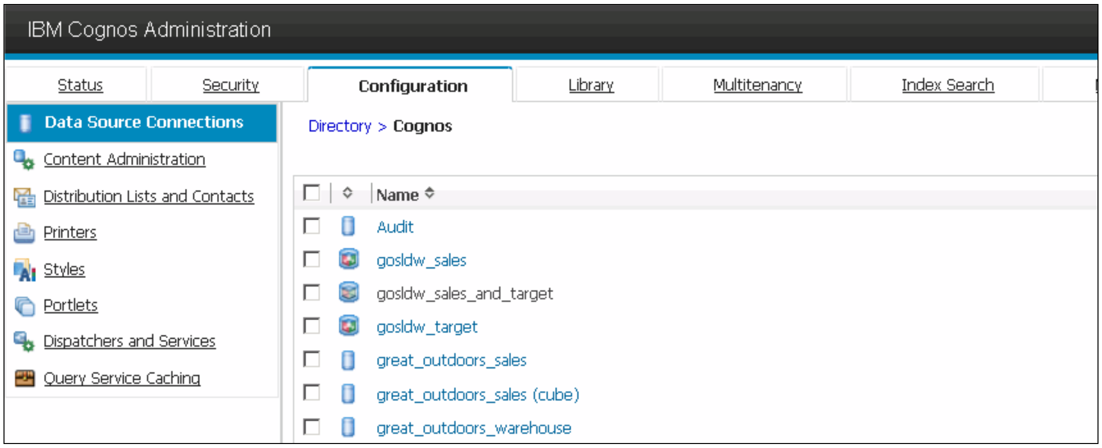


Figure 2-16 Dynamic cube and virtual cube data source connections among other connections

Dynamic and virtual cube data source connections not only differ visually from other CM data sources, but they also have their own specific set of properties and are composed of different objects. The differences in property settings for the configured cube instances are described in 2.1.5, "Configuring dynamic cubes" on page 33.

Table 2-1 compares the structure of data source connections to base dynamic cubes and virtual cubes against a data source connection to an external data source.

Table 2-1 Data source types and their component objects

Data source type	Objects comprising the Data Source Connection
Base dynamic cube	A dynamic cube data source object, a model object, security views (optional)
Virtual cube	A dynamic virtual cube data source object
External data source	A data source, one or more connections, one or more signons (there can be multiple signons for each connection)

For more information about the structure of a published base dynamic cube, see Figure 2-2 on page 15.

When managing and deploying dynamic cubes, note that there are two data source connections associated a dynamic cube.

The data source connections associated with a dynamic cube are created in the following order:

1. The first connection, to the underlying database, is required by both the cube model in Cognos Cube Designer, and the published dynamic cube to access the source data warehouse. After the cube is published, this connection is used by the query service to populate the dynamic cube and service requests.
2. The second connection, to the published cube, has a dual purpose in that it represents the cube in Cognos Administration and is used by the packages to present the published cube

to the Cognos BI reporting and analysis interfaces. This connection also references the first connection by using the access account property.

Despite being hosted in the Cognos BI environment, a published cube is simply a data source from the perspective of the Cognos BI interfaces. Hence, as with any other data source, the Cognos BI interfaces cannot access the published cube without access to a package and a CM data source connection to the cube.

Using the `Samples` and `Sample_Dynamic_Cubes` sample content, consider the Go Data Warehouse Sales package provided in the `Samples_Dynamic_Cubes` folder. This package uses the `gosldw_sales` data source connection that is shown in Figure 2-16 on page 31 to access the `gosldw_sales` dynamic cube. The published `gosldw_sales` cube in turn references the `great_outdoors_warehouse` data source connection to access the underlying relational database, namely, GOSALESDW.

The most important property of a base dynamic cube data source is the access account property. This property setting specifies the single account that is used to access the underlying relational database, which is the data source of the cube.

Figure 2-17 shows the dynamic cube access account property.

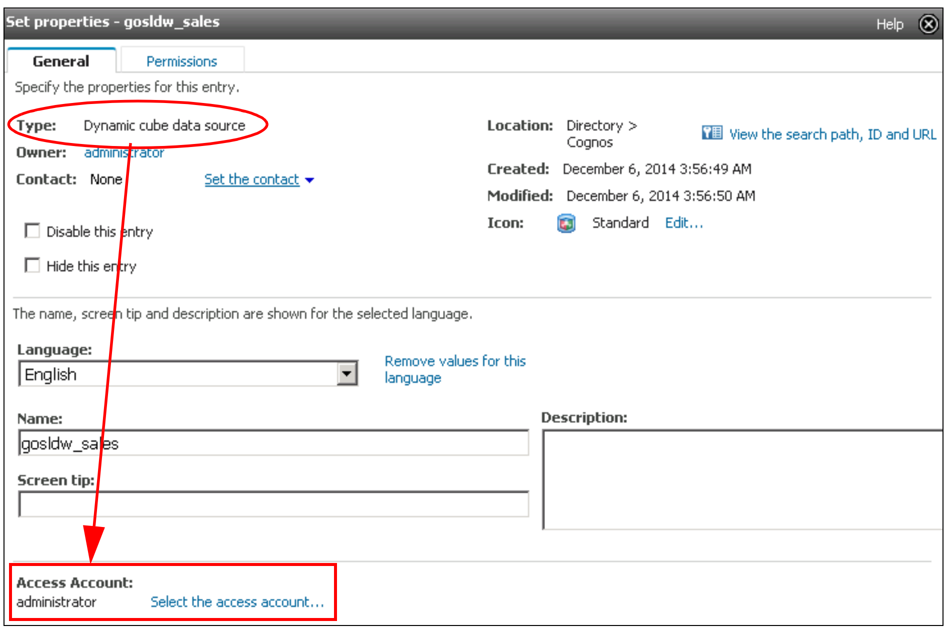


Figure 2-17   Dynamic cube access account property

Note: The data source connection for a virtual cube does not have an Access Account property as a virtual cube does not connect to the underlying source database. Virtual cubes are populated entirely from their two source cubes, each of which can be a base cube or another virtual cube.

Because a dynamic cube is published to the content store as a data source connection it can be included in a multi-source package. A Framework Manager (FM) project can connect to the cube as an OLAP data source, allowing it to be added to the packages in the FM project. If the required sources are all dynamic cubes and defined in a single project, then the multi-source package can be created in Cognos Cube Designer.

A dynamic cube data source connection also has a child model element. If security has been defined for a cube, the child model element contains a collection of security views to which users, groups, or roles can be assigned. These security views are defined by the cube modeler by using Cognos Cube Designer.

Figure 2-18 shows dynamic cube security views.

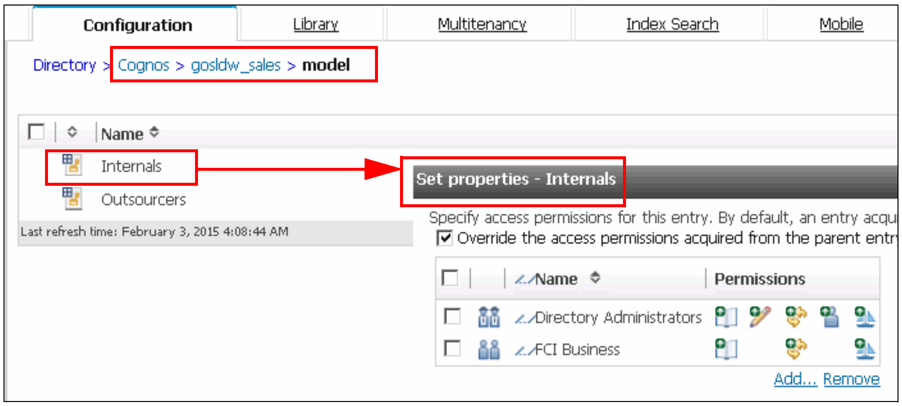


Figure 2-18 Dynamic Cube security views

2.1.5 Configuring dynamic cubes

Publishing a dynamic cube to the content store as a data source connection does not make a cube immediately accessible to users. The metadata definition of a cube is published so that it is accessible to other parts of Cognos and a data source connection is created to provide access to the published cube. The next step in making a cube available to users is to configure the published dynamic cube against one or more dispatchers inside a Cognos BI environment.

Configuring a dynamic cube against a dispatcher accomplishes two goals:

- Identifies a specific dispatcher on which a cube can be active, that is, has a running cube instance.
- Defines the operational characteristics of the cube instance hosted by the query service on that dispatcher.

The query service acts as both a host to dynamic cubes as data sources and an enabler of DQM requests against dimensionally modeled relational data (DMR) packages and dynamic cube packages. The running cube instance is hosted in the JVM of the query service. Requests against the dynamic cube are passed from the report service to the query service to access the Multidimensional Expression Language (MDX) engine inside. DQM requests against a DMR package are processed in the same way.

In most circumstances, dynamic cubes are assigned to specific dispatchers that are running on hardware allocated to host these cubes. This hardware selection is based on the memory and CPU requirements for hosting dynamic cubes, both of which are in excess of what is normally required for a query service alone. One consequence of this approach is the requirement to use advanced routing rules to route reports and analyses for dynamic cubes to the appropriate dispatchers.

Figure 2-19 shows multiple cubes that are deployed across multiple servers.

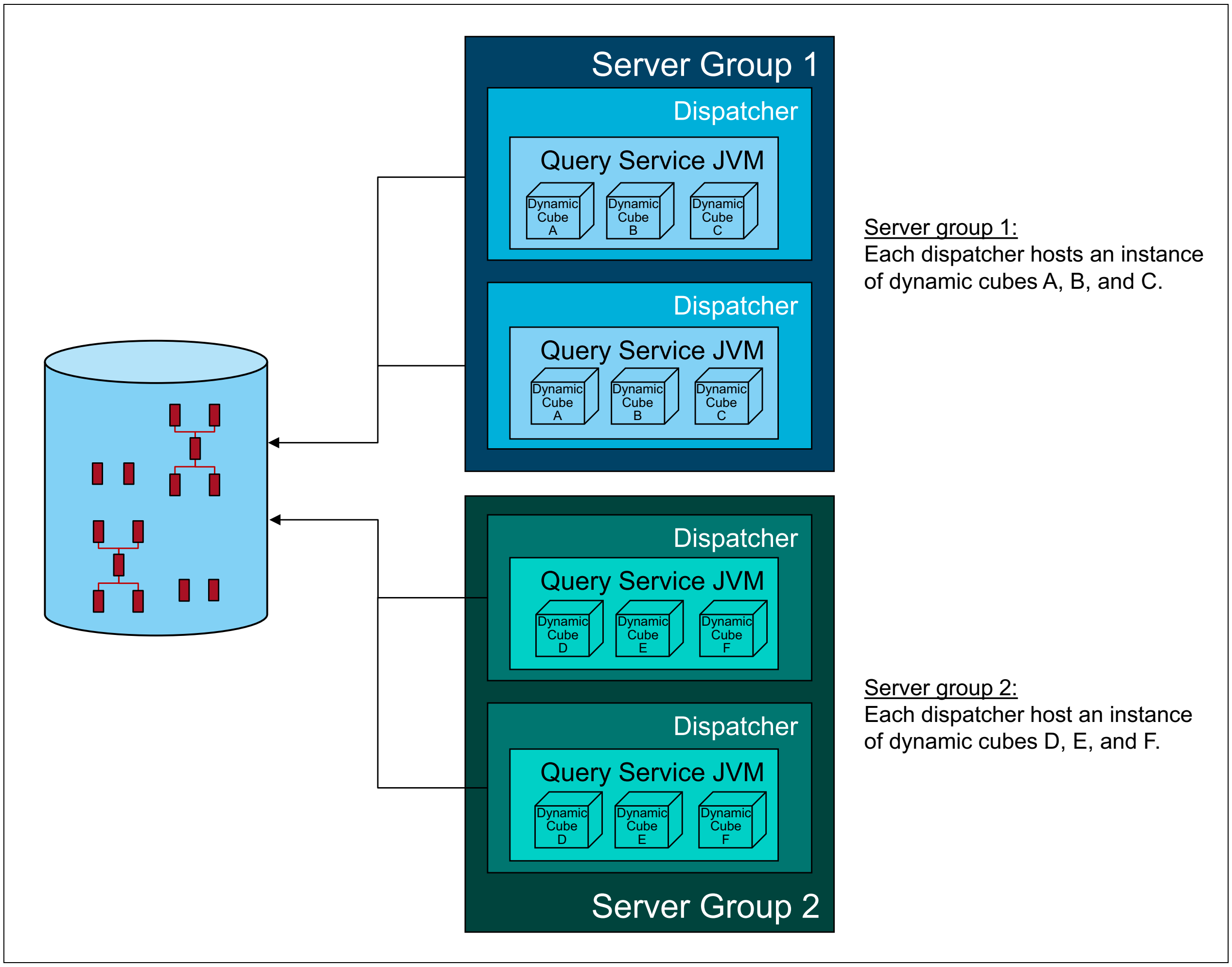
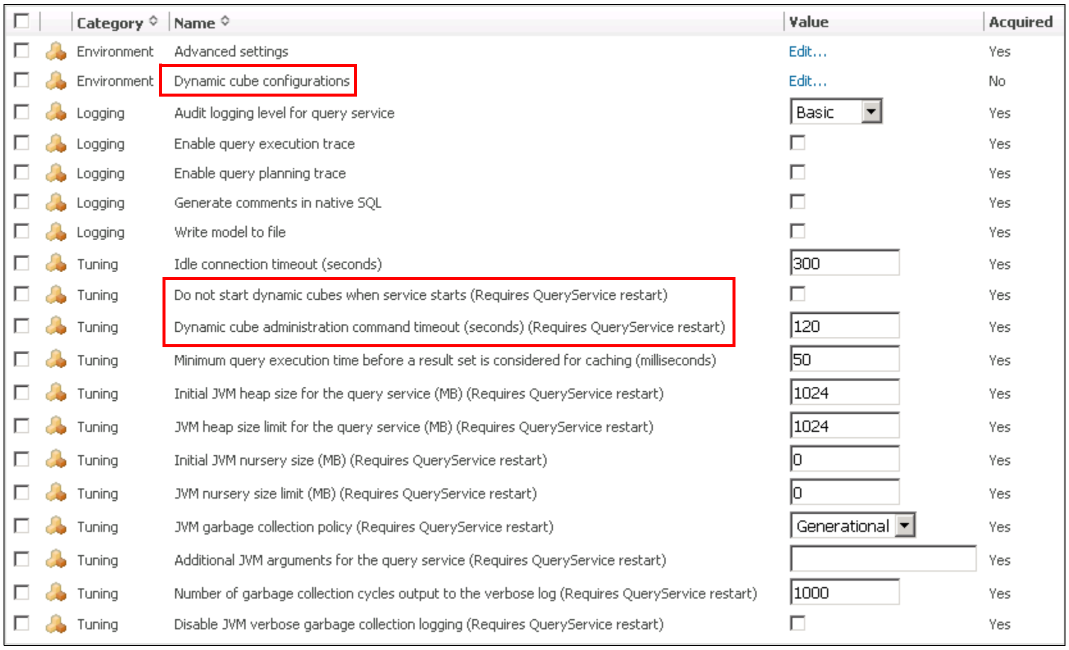


Figure 2-19 Multiple cubes that are deployed across multiple servers

The cube configuration for each cube instance sets the values of the cube properties. These property values are defined for each dispatcher that a cube is configured against. Default values are set for all configuration properties when the cube is added to a query service. Several properties, such as data and aggregate cache sizes, should be assigned non-default values based on an evaluation of the dimension and fact data volumes, the numbers of users, and the expected use of a cube. Other properties values can be assigned during the development and production lifecycle of a cube.

Because dynamic cubes are hosted inside the query service JVM, additional properties must be set on the query service to account for the presence of dynamic cubes, most notably those related to the JVM heap, as shown in Figure 2-20.

Figure 2-20 shows query service settings for dynamic cubes.



Category	Name	Value	Acquired
Environment	Advanced settings	Edit...	Yes
Environment	Dynamic cube configurations	Edit...	No
Logging	Audit logging level for query service	Basic	Yes
Logging	Enable query execution trace		Yes
Logging	Enable query planning trace		Yes
Logging	Generate comments in native SQL		Yes
Logging	Write model to file		Yes
Tuning	Idle connection timeout (seconds)	300	Yes
Tuning	Do not start dynamic cubes when service starts (Requires QueryService restart)		Yes
Tuning	Dynamic cube administration command timeout (seconds) (Requires QueryService restart)	120	Yes
Tuning	Minimum query execution time before a result set is considered for caching (milliseconds)	50	Yes
Tuning	Initial JVM heap size for the query service (MB) (Requires QueryService restart)	1024	Yes
Tuning	JVM heap size limit for the query service (MB) (Requires QueryService restart)	1024	Yes
Tuning	Initial JVM nursery size (MB) (Requires QueryService restart)	0	Yes
Tuning	JVM nursery size limit (MB) (Requires QueryService restart)	0	Yes
Tuning	JVM garbage collection policy (Requires QueryService restart)	Generational	Yes
Tuning	Additional JVM arguments for the query service (Requires QueryService restart)		Yes
Tuning	Number of garbage collection cycles output to the verbose log (Requires QueryService restart)	1000	Yes
Tuning	Disable JVM verbose garbage collection logging (Requires QueryService restart)		Yes

Figure 2-20 Query service settings for dynamic cubes

After a dynamic cube is configured, it is still not accessible to users. Dynamic cubes must be started, either explicitly or implicitly, before they become accessible to users. As with any other data source defined inside the content store, there must also be at least one package that includes the cube to provide the Cognos BI interfaces with access to the cube.

2.1.6 Dynamic cube management

Dynamic cubes are managed on the dispatcher to which they are assigned during configuration. Each instance of a dynamic cube is assigned to a single dispatcher and is managed independently of other instances. Instances of a cube can be started individually or started all together. When started, extra management tasks, such as priming and refreshing or updating the caches and monitoring cube metrics, are required to improve performance and maintain the currency of data inside the running cube instances.

A cube can be started in one of the following ways:

- Starting the query service

  By default, cubes are started automatically when the query service starts. This feature affects all cubes that are hosted on the query service. This feature can be turned off using a query service tuning property. Individual cube instances can be disabled so that they are not started when the query service starts. *Disabled* is a property setting in the cube configuration on the query service.

  It is important to note that after a cube instance is set to disabled, it cannot be started regardless of the method used to start a cube. This property is set on a per-cube instance basis and prevent that specific cube instance from being started.

- Starting the cube from IBM Cognos Administration

  The Dynamic Cubes scorecard on the Status tab of Cognos Administration is dedicated to viewing and managing dynamic cubes. From this scorecard, the administrator can perform a series of actions. These include starting or stopping a cube, refreshing the member cache, incrementally updating data in a cube, and editing the permissions on the security views of a cube.

  By default, the Dynamic Cubes scorecard presents a list of all published cubes along with the status of that cube. This scorecard can be filtered to show only base cubes or virtual cubes. It can also be filtered to show the configured cube instances by server group and query service.

- Starting the cube by using a *dynamic cubes query service administration task.*

  A dynamic cube query service administration task allows the Cognos administrator to create and schedule a task that can start all, or a selection of the cube instances hosted on a nominated dispatcher. Dynamic cube query service administration tasks can also be used for other operations such as stopping or refreshing the cache of selected cube instances. See 7.6, "Scheduling the refresh of the cache" on page 214 for steps to create a dynamic cube query service administration task.

- Starting the cube by using the DCAdmin command-line tool

  The DCAdmin tool (`dcadmin.bat` for Microsoft Windows, `dcadmin.sh` for UNIX) can be used to perform various dynamic cube administrative tasks from the command line, operating on a single cube instance that is configured on a single dispatcher. Available commands include **`startCubes`** to start the cube, **`getCubeState`** to confirm the running state of a cube, and **`getCubeMetrics`** to review the cube metrics. The output can be displayed on a window or saved to an xml file for review later.

- Creating an SDK application that starts a cube

  The software development kit (SDK) can be used to call a ROLAP administrative task to start a cube as part of a maintenance application. The `startCubes` method, part of the `rolapCubeAdministration` method set, can be used to start a dynamic cube on a particular query service.

After a cube is started, it immediately retrieves data from all of the dimension tables in the data warehouse that are represented in the cube model. This information is stored in an in-memory representation of all of the hierarchies and members comprising the cube. After this information is loaded, the cube is available for access from the various BI studios.

A cube can also have in-memory aggregates defined. In-memory aggregates are loaded after a cube is available for use and it is important to note that the cube can be accessed before the in-memory aggregates finish loading. The progress of this load activity is visible in the metrics for a cube in Cognos Administration.

Another management task is to prime the cube caches, which can be achieved by using triggers to run priming reports. The intent of a trigger is to run one or more reports, analyses, or both, to preinstall the data cache of a cube, ensuring that the data is available in the cache for subsequent users. Dynamic cubes can be assigned two types of triggers: One runs on cube start and the other runs on completion of the in-memory aggregate cache. These properties can be found on the cube configuration page for each instance of a cube.

Figure 2-21 shows the properties for assigning triggers to a cube in the cube configuration settings.

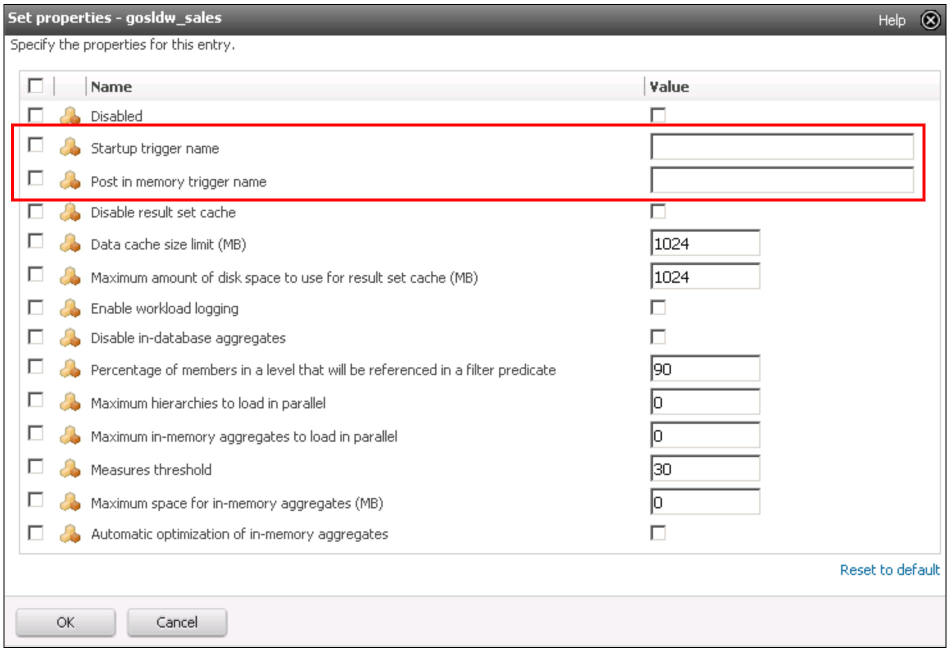


Figure 2-21 Assigning a Startup trigger name or Post in-memory trigger name to a cube

If a cube has the `Startup trigger name` property set, the named trigger runs when the cube is available for use. However, setting this property can cause the associated reports to run before the in-memory aggregates have finished loading.

Setting the `Post in memory trigger` property on a cube can avoid contention between in-memory aggregate loads and priming requests. Delaying priming reports until the in-memory aggregate cache load is complete has the added advantage of ensuring that dependent reports are able to take full advantage of the in-memory aggregate cache. See 2.3.4, “Aggregate cache” on page 51 for more detail.

Several operations can be performed against a cube when it is running:

- Stop
- Restart (stop, start)
- Refresh member cache
- Refresh data cache
- Refresh security
- Clear workload log
- View recent messages

Figure 2-22 shows the menu for administering a cube.

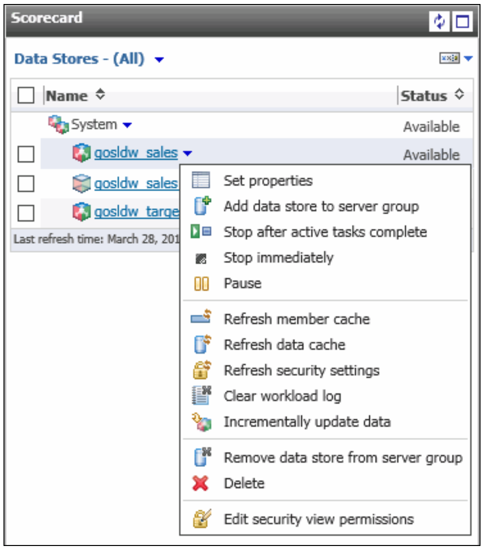


Figure 2-22 Menu for administering a cube

Figure 2-23 shows the new menu entry for filtering the Dynamic Cubes scorecard in Cognos Administration.

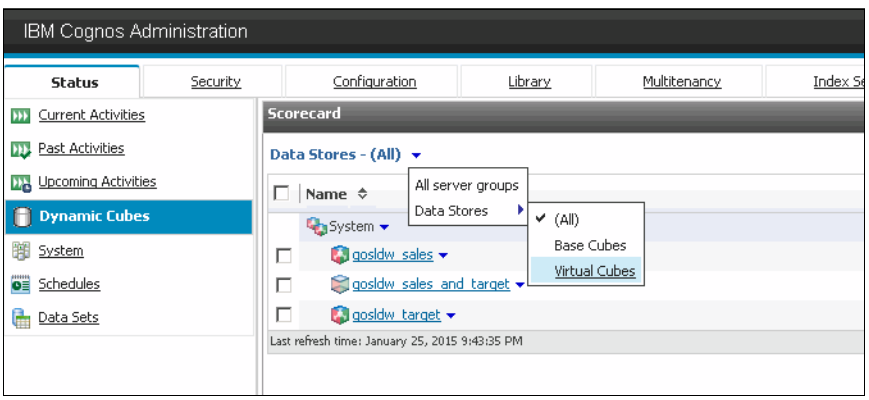


Figure 2-23 Filtering the list of cubes on the Data Stores scorecard

An important management task is to monitor the cube metrics. Metrics can be viewed for an individual cube instance by locating the query services against which the cube is configured (see step 5 under 7.2.1, “Starting a cube” on page 205) and clicking the cube instance that you want to monitor.

The following are useful metrics you can include:

- Cube state
- Cube up time
- Cache hit rates
- Aggregate table hit rates
- Used versus defined cache sizes
- Metadata load times

For a list of cube metrics and more information about viewing the metrics of a cube instance, see 7.2.3, “Monitoring cube state through metrics” on page 208.

2.1.7 IBM Cognos Dynamic Cubes Aggregate Advisor

A key feature of Cognos Dynamic Cubes is its ability to take advantage of both in-database and in-memory pre-computed aggregates. These pre-computed aggregates can improve the performance of queries by high orders of magnitude, providing the level of performance required for interactive reporting and analysis.

If you have pre-existing aggregate (or summary) tables in your data warehouse, you can model them as in-database aggregates in Cognos Cube Designer. When a cube is published and the cube is restarted, it automatically routes SQL queries to the aggregate tables instead of the fact table, when possible. For distributive measures (that is, those for which the aggregation rule is SUM, COUNT, MAX, or MIN), aggregate tables can be employed to compute summary values at higher levels of aggregation than those defined in an in-database aggregate modeled in the cube. For an AVG, if a summary exists that contains both SUM and COUNT, Cognos Dynamic Cubes routes the query to the aggregate table and compute the value SUM/COUNT.

As useful as this capability is, one of the most difficult tasks when defining pre-computed summaries is trying to determine what should be pre-aggregated, especially in a large, multi-user environment that might involve hundreds or thousands of reports and analyses. The Cognos Dynamic Cubes Aggregate Advisor, available as part of the IBM Cognos Dynamic Query Analyzer (DQA), performs this task.

The Aggregate Advisor can be used to suggest database aggregate tables, in-memory aggregates, or both. The Aggregate Advisor makes use of a cube's model along with statistics it gathers from the underlying data warehouse to determine which summary tables to recommend. Aggregate Advisor can also use workload log files that are generated from the execution of reports and analyses to make more accurate suggestions of strategies for optimizing the performance of a dynamic cube application workload. Finally, Aggregate Advisor can incorporate specifications defined by the cube modeler into its recommendations.

Figure 2-24 shows the Aggregate Advisor selection of workload information.

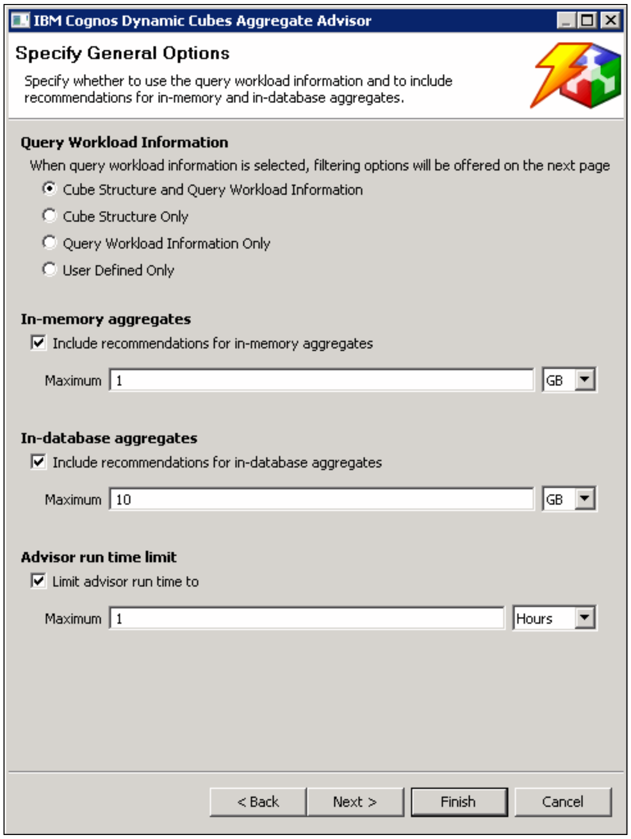


Figure 2-24 Aggregate Advisor selection of workload information

The workload log files that are generated by the query service contain detailed information that can be used as filters to isolate a subset of cube workload to be optimized. In addition, the information in the workload log allows the Aggregate Advisor to determine which values to pre-aggregate for use across a wide range of reports and analyses. For example, aggregate data at the quarterly level and enable the use of this data to compute annual totals without precomputing the annual levels.

An in-database aggregate recommendation contains a detailed description of each suggested aggregation, a generic set of column descriptions, and database-specific SQL that can be used to populate the table. These suggestions can be used by a relational DBA to construct the necessary tables and the corresponding extract, transform, and load (ETL) scripts to build and maintain the tables. The new tables are then added to the cube model as in-database aggregates in Cognos Cube Designer and become available when the cube is republished and started.

In-memory aggregate recommendations for a cube are implemented directly from Aggregate Advisor. This process involves publishing the aggregate specifications to the CS. From Aggregate Advisor, you can select a combination of recommendations from the current and previous Aggregate Advisor runs for publishing. After the cube is restarted, the new aggregate specifications run the necessary SQL statements to retrieve the necessary summarized values and populate the cube’s in-memory aggregate cache.

User-defined in-memory aggregates are specified by the modeler in Cognos Cube Designer based on the modeler's knowledge of anticipated workload, the cube structure, and the structure of the underlying relational database. The cube must be published and then

Aggregate Advisor used to apply the aggregates to the published cube. User-defined in-memory aggregate information is detected by Aggregate Advisor and included in its recommendations. The Aggregate Advisor is required for two reasons: To publish the recommendations to the CS and to estimate the size of any user-defined aggregates.

Virtual cubes cannot have an in-memory aggregate cache because they have no underlying database from which to obtain aggregate values. Virtual cubes cannot have in-database aggregates either because they cannot have aggregate tables associated with them.

Figure 2-25 shows the availability of in-memory aggregates to base and virtual cubes.

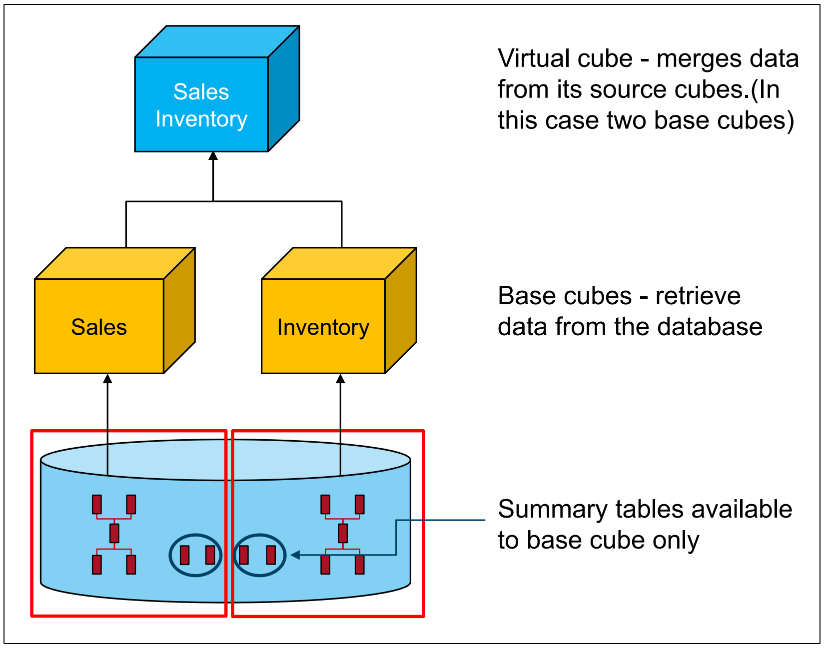


Figure 2-25 Availability of in-memory aggregates - Base versus virtual cubes

Note:

A *virtual cube* cannot access aggregate tables or construct in-memory aggregates because it has no direct access to the data warehouse.

A *base cube* can construct in-memory aggregates from data in the fact table and include in-database aggregates based on summary tables because it has direct access to the data warehouse.

2.2 Dynamic cubes and the query service

Dynamic cubes are hosted in a query service, which in turn, not only hosts the cubes but also interacts with them. As such, an important concept to understand is how dynamic cubes exist inside a query service and how administrative commands and requests for metadata and data are directed to a dynamic cube. This section describes the interaction between a report server and a query service as it pertains to Cognos Dynamic Cubes, the components of a dynamic cube inside a query service JVM, and how commands and requests are directed inside the query service to a dynamic cube. Processing of these requests is described in 2.4, “Dynamic cubes query processing” on page 54.

For more information about DQM query processing and the MDX engine in the query service, see *IBM Cognos Dynamic Query*, SG24-8121.

2.2.1 Dynamic cubes and the report server

A dispatcher can create multiple instances of a report server, but only one query service instance. All of the report servers that run under a dispatcher communicate with the single query service instance through a single port. Thus, any dynamic cubes that are assigned to a dispatcher are all hosted inside a single instance of the query service.

You can distribute a dynamic cube to provide user scalability, by configuring the cube on more than one dispatcher, thus creating multiple running instances of a cube. In this case, the query service of each dispatcher hosts a separate version of the cube. Therefore, the interaction of one cube instance with the underlying relational database is distinct from that of any other instance of the same cube on other dispatchers. Similarly, the caches for each cube instance are distinct from that of any other instance and managed independently. Although distributing a cube allows more users to access a single, logical cube, it is important to note that distributing a cube also generates some additional processor burden.

Keep in mind these considerations for distributing dynamic cubes:

- Each computing node to which a dynamic cube is configured must have the memory and CPU cores necessary to support the cube and its expected user volume, taking load balancing into account.
- If a dynamic cube is deployed across multiple nodes, the load on the underlying relational database can be multiplied notably during the loading of in-memory aggregates if start, restart, and data cache refresh operations are performed at the same time across the multiple instances of the cube.
- The data caches of the different instances of a cube are not synchronized, so if users run a query on one system and then on another system, query response can fluctuate.
- The underlying database might encounter extra load because each instance of a cube is required to load data independently of the other node.
- The dispatcher is not aware of dynamic cubes, so it does not account for whether a cube is available under a particular dispatcher when dispatching a request for execution. Therefore, extra steps are required to ensure that users do not receive messages that indicate that the cube is not available when cube instances are taken online and offline on individual dispatchers.

Additional start and tuning settings have been added to Cognos Dynamic Cubes to assist with these considerations. For more information about dynamic cubes in a distributed environment, see Chapter 18, “Dynamic cubes in a multi-dispatcher environment” on page 543.

2.2.2 Coexistence of 32-bit report server and 64-bit query service

IBM provides a choice of 32-bit and 64-bit versions of the Cognos BI Server installation program for creating the Cognos BI environment. The bit version of installed components is determined by the bit version of the installation program with some exceptions, most noticeably the report server. The 32-bit program installs a 32-bit report server whereas the 64-bit program installs both a 64-bit report server and a 32-bit report server.

The query service is a separate process from the report server, so the query service and the report server can have different bit versions. Usually a 64-bit query service is preferable for its greater addressable memory, especially with Cognos Dynamic Cubes. The 32-bit query service runs as a 32-bit Java process. For this reason, the JVM heap size is limited to 2 or 3 GB. Cognos Dynamic Cubes is designed for large volumes of data, so a typical installation uses the 64-bit installation program on a 64-bit operating system.

Table 2-2 shows the availability of installed components and resulting report server and query service bit versions.

Table 2-2 Bit versions of Application Tier Components

	Installed Components		
Installation files	**32-bit Report server**	**64-bit Report server**	**Query service**
32-bit ATC	✓	✗	32-bit
64-bit ATC	✓	✓	64-bit

IBM Cognos BI uses two different *query modes* to process requests for data; Compatible query mode (CQM) and dynamic query mode (DQM). The term query mode refers to the mode of a report request, which in turn must be matched against the query engine that can process it. CQM requests are handled by the CQM server that is embedded in the report server. DQM requests are handled by the query service. Being an extension of IBM Cognos Dynamic Query Mode, IBM Cognos Dynamic Cubes uses the MDX engine inside the query service.

For both 32-bit and 64-bit Cognos BI installations the dispatcher is configured, by default, to run in 32-bit report server execution mode. Use of a 32-bit report server is the default configuration because a 32-bit report server supports both CQM and DQM, which allows the dispatcher to accept both CQM and DQM report requests. CQM is strictly only supported on dispatchers that are configured to 32-bit report server execution mode. Report server execution mode is an environment property in IBM Cognos Configuration.

Cognos Dynamic Cubes is compatible with both the 32-bit report server and the 64-bit report server.

Table 2-3 shows the compatibility of the report server execution mode setting on a dispatcher with available components in the Cognos BI environment.

Table 2-3 Report server execution mode compatibility by report server and query service

Dispatcher property	**Report server**			**Query service**	
	Report server instances	**CQM request**	**DQM request**	**32-bit Query service**	**64-bit Query service**
Report server execution mode = 32-bit	32-bit	✓	✓	✓	✓
Report server execution mode = 64-bit	64-bit	✗	✓	N/A	✓

Understanding the distinction between report server and report server mode:

The *report server* is a component of Cognos BI installed along with the dispatcher when you select Application Tier Components (ATC) during the installation process. The bit version of the installation files determines which report servers are installed. For example, the v10.2.2 64-bit installation file includes two report servers, one 32-bit and one 64-bit, whereas the v10.2.2 32-bit version of this file includes the 32-bit report server only.

The term *report server* mode refers to the `report server Execution Mode` environment setting on a dispatcher. This setting (32-bit or 64-bit) nominates the bit version of the report servers that this particular dispatcher is capable of using to run a report request. A 32-bit report server is capable of processing both CQM and DQM requests. A 64-bit report server rejects CQM requests.

Note: This dispatcher setting should not be confused with the bit version of the dispatcher itself.

Use 32-bit report server execution mode for a dispatcher that is configured to host dynamic cubes and there is a requirement to support CQM data sources on the same dispatcher. The 32-bit report server can handle both CQM and DQM requests.

A 32-bit report server is sufficient in most cases because the report server does not need a large memory space. The main advantage of running a 64-bit report server is to prevent an out-of-memory condition in the report server process. Use 64-bit report server execution mode if this situation arises, but use of a 64-bit report server can also introduce a requirement to control the routing of requests to the correct server.

Memory metrics, such as high water mark, for the report server are not available in Cognos Administration. Consider using a process monitor tool to observe memory consumption by the report server process (BIBusTKServerMain).

2.2.3 Dynamic cubes in the query service JVM

Running dynamic cube instances exist in memory inside the query service. A dynamic cube, when it is running, involves the following in-memory caches:

- An in-memory *member cache*

  The members of each hierarchy are retained in memory to allow for the traversal of parent-child relationships, and level relationships. This cache is used during DQM query planning to validate member-unique names, and also during query execution to perform member and set operations that do not require the evaluation of measure values, for example, MEMBERS([Level Name]).

- An in-memory *aggregate cache*

  This cache consists of *cubelets* of aggregated values, one for each in-memory aggregate up to the amount of space configured for the cube. This cache is loaded from the in-memory aggregate recommendations published to the CS from the Aggregate Advisor and is only used by cubes that have been configured to assign memory for in-memory aggregates.

- An in-memory *data cache*

  Any queries that retrieve data from the underlying database, or further aggregate data from the aggregate cache, are stored as cubelets in a separate data cache.

- An in-memory *security cache*

  Every unique combination of the security views in a cube is retained in memory to provide efficient application of security filters to queries.

- An in-memory *expression cache*

  Results of set expressions, tuples output by set expressions operating on large sets of members, are stored for reuse by the MDX engine when processing the same query or subsequent queries.

Dynamic cubes also have a file-based cache that is outside of the query service, the result set cache. This cache stores the result set of each MDX query run by the MDX engine.

Figure 2-26 shows the various caches in a dynamic cube.

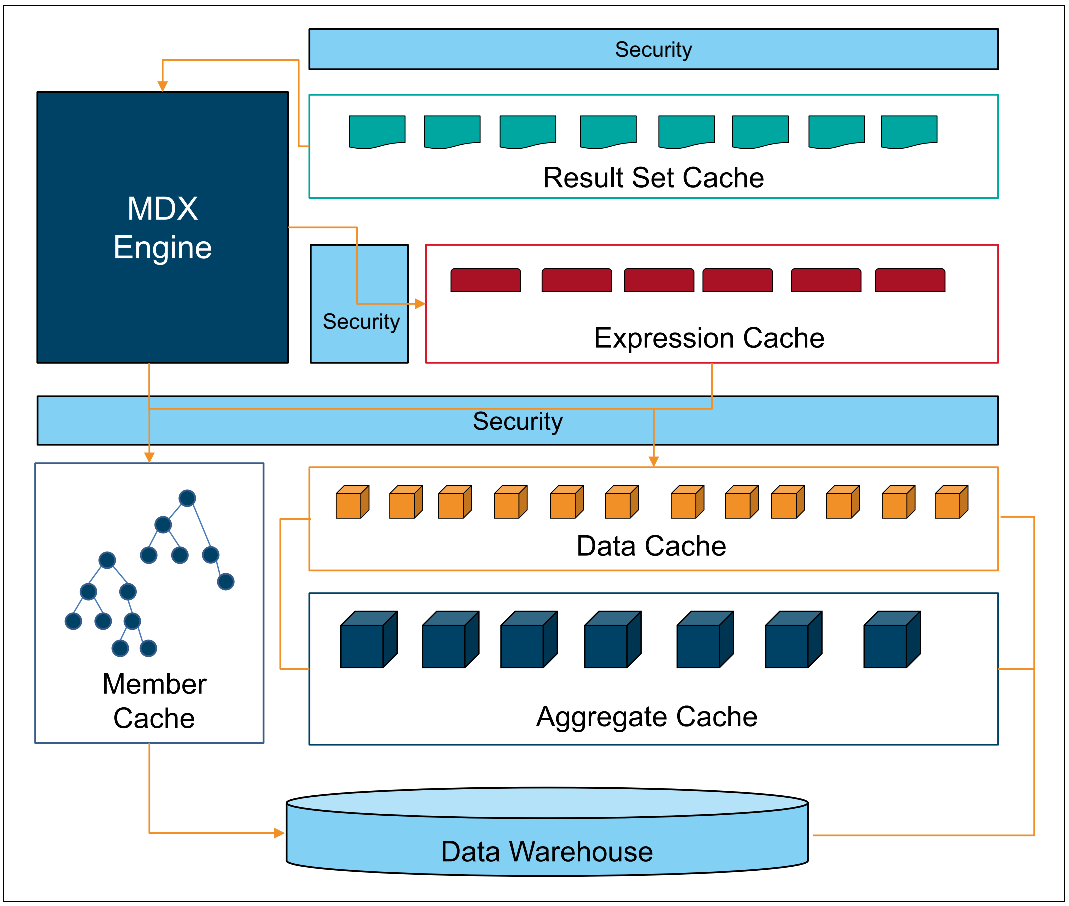


Figure 2-26 The various caches in a dynamic cube

2.2.4 Query service interactions with dynamic cubes

The various caches described in 2.2.3, "Dynamic cubes in the query service JVM" on page 44 are the components that comprise a dynamic cube. These structures are retained in a cube construct inside the Java heap of the query service, but separately from the remainder of the query service.

The query service interacts with a dynamic cube in the following ways:

- Administrative commands
- Metadata requests
- Query execution

Administrative commands can be received either through Java Management Extensions (JMX) or BI bus commands (typically from Cognos SDK applications). These administrative commands are directed by the query service to the Cognos Dynamic Cubes management interface, which in turn directs the requests to the specified cube to perform operations such as stop, start, and so on.

Metadata requests are typically posed by Cognos BI client applications to populate the metadata browser, or by the DQM query planner, which interacts with a cube to obtain the various pieces of metadata required to validate the metadata references of a request.

Query execution results in data requests. The query service converts queries that it receives into MDX queries, which are then run by the MDX engine of the query service. During the running of an MDX query, the MDX engine might need to perform member operations such as obtaining the children of a member. It does so through the metadata interface of a cube. The MDX engine might also require measure values, which it requests through an interface called the *query strategy*.

Both metadata and data requests apply any security defined in the model before returning member metadata and before retrieving measure values.

Data requests, through the query strategy of a cube make use of the aggregate and data caches of a cube.

Figure 2-27 shows the command dispatcher inside the query service.

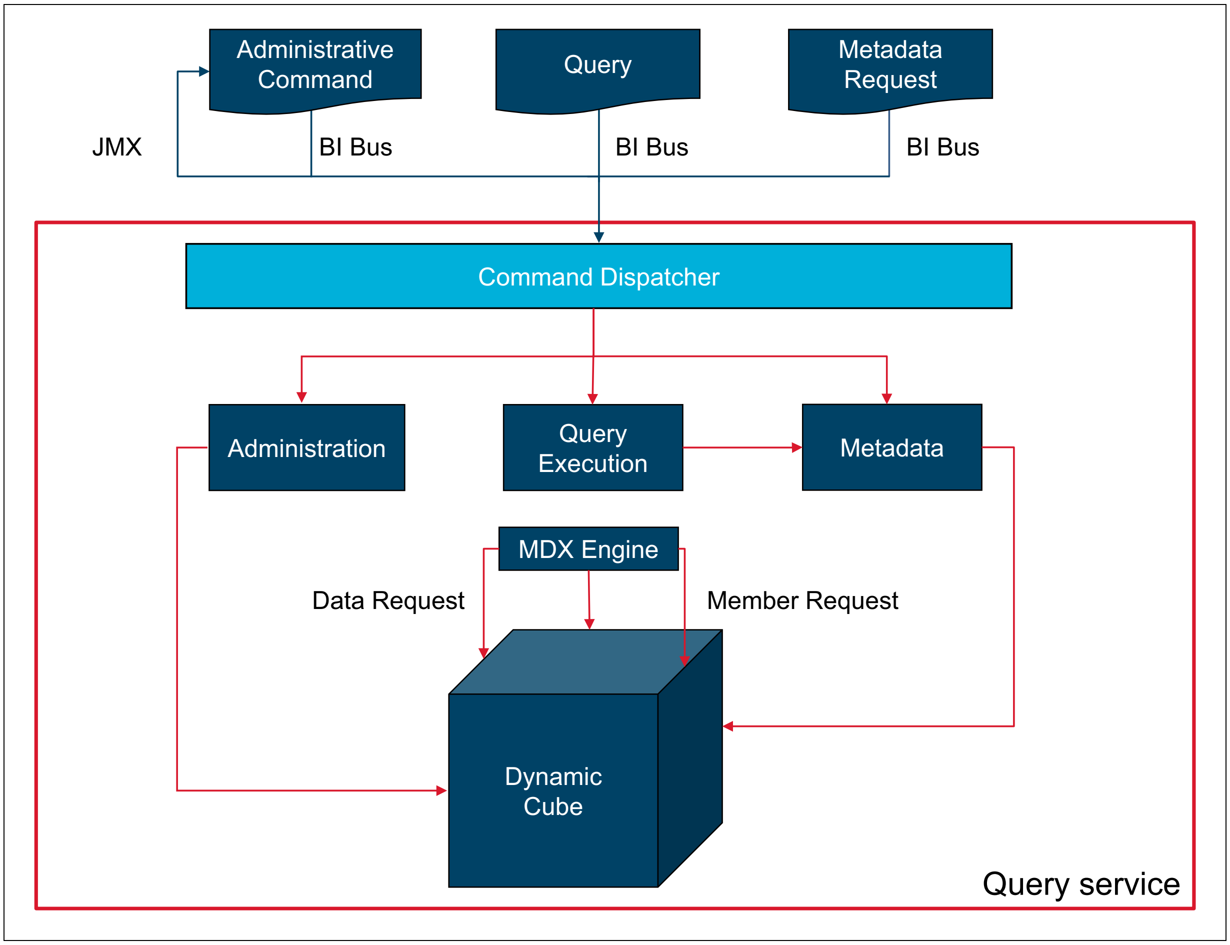


Figure 2-27 Command dispatcher inside the query service

The following are architectural considerations for ensuring successful interactions between the query service and a dynamic cube:

- A single query service can host one or more different dynamic cubes at one time.
- If a query service hosts a virtual cube, all of the cubes upon which the virtual cube is based must also be hosted on the same server.
- As a dispatcher can only be assigned to a single-server group, all cubes that are inside the associated query service are implicitly part of the same server group. Therefore, when configuring advanced routing rules, all packages that reference any of the cubes on the query service must route their requests to this same server group.
- In a scenario in which a dynamic cube is running on multiple dispatchers in a server group, if one dispatcher's query service becomes unavailable due to insufficient memory, then report requests for the cube are routed to another dispatcher's query service within the server group. If all query services in the server group are unavailable due to insufficient memory, then an error is returned to the user.

See Chapter 18, "Dynamic cubes in a multi-dispatcher environment" on page 543 for further architectural considerations in a distributed environment.

2.3 Cognos Dynamic Cubes caching

The basis for the performance of Cognos Dynamic Cubes is its various in-memory caches and its use of database aggregate tables. Additionally, the query service supports 64-bit architectures, and, as such, provides Cognos Dynamic Cubes access to large amounts of memory.

Cognos Dynamic Cubes can reuse data that is held in its various caches rather than requerying the database. By using the memory resident member, aggregate and data caches Cognos Dynamic Cubes not only provide fast query response, but also help to alleviate the processing load on the underlying relational database.

The result set and expression caches, in turn, reduce the amount of processing that occurs inside the MDX engine of the query service, helping to reduce spikes in resource usage.

Cognos Dynamic Cubes also uses available processing cores to do costly operations in parallel, such as loading hierarchy tables, or processing query result sets.

The following sections describe each of the Cognos Dynamic Cubes cache types in more detail.

2.3.1 Result set cache

The result set cache exists at a layer directly above the MDX engine inside the query service. The result set of each MDX query run by the engine is stored within this on-disk result set cache.

Figure 2-28 shows the result set cache.

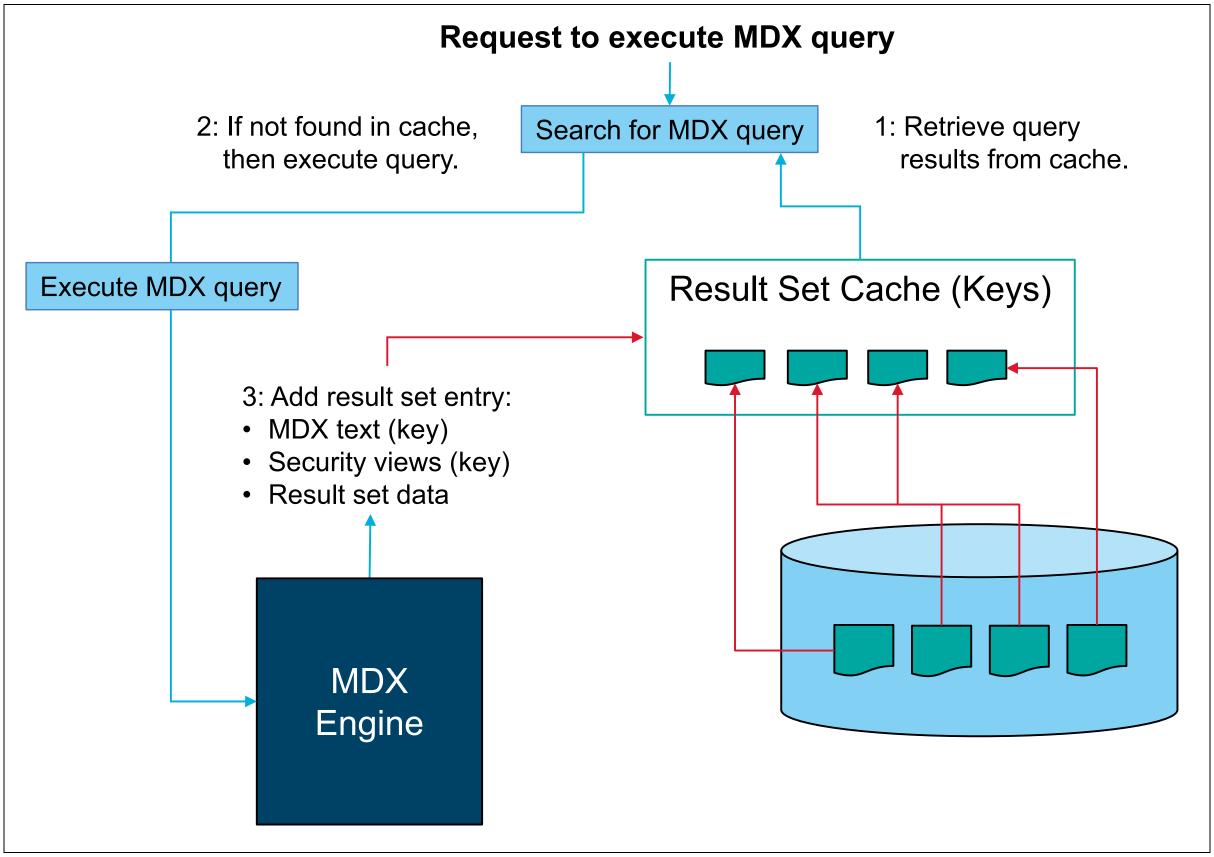


Figure 2-28 Result set cache

On running each query in its plan, the MDX engine first searches the result set cache. If the query has been run previously and the combination of security views for the user who ran the stored result match those of the current user, then this stored result set is reused. Here the query is run only on the first request.

The result set cache is aware of queries that are currently running, so it is not possible for a query to be run twice. If necessary, subsequent requests for the same result set wait for the original query to complete execution. Also, the result set cache is flushed when the data cache is refreshed to ensure that the data and result set caches are synchronized inside the cube instance.

Although stored on disk, the result set cache is capable of providing fast report execution because, in many cases, the output of a report is small compared to the amount of data that must be processed to produce the final result set.

The result set cache not only facilitates the sharing of report execution results between users, but also speeds *drill up* operations because drill up involves a result set that has been generated for an earlier request.

2.3.2 Expression cache

The expression cache is used by the MDX engine to store the results of set expressions that operate on large sets of members and output a much smaller set for subsequent reuse. Reuse can occur either inside the same query or inside subsequent MDX queries processed by the MDX engine. The savings from the use of cached expression results can be significant, given that in some cases a set can contain millions of tuples and only output five or ten tuples.

A set inside the context of MDX syntax contains one or more tuples. A tuple contains a member from one or more dimensions, where one of those dimensions can be the measures dimension. The tuples inside an MDX set each contain one member from a common collection of dimensions specified in the same order for each tuple in the set. These set expressions are used by the MDX engine to generate the query data values.

The expression cache feature focuses on the MDX functions that process a large input set of tuples and return a smaller set of tuples.

Large sets can be characterized as those that process a large set of tuples as input and perform the following tasks:

- Perform a simple operation to return a smaller set of tuples (for example, HEAD)
- Fetch the corresponding values, and return a single value (for example, AGGREGATE)
- Fetch the corresponding values, and return a smaller set of tuples

Specifically, the following MDX functions are supported by the expression cache:

- TOPCOUNT
- BOTTOMCOUNT
- TOPPERCENT
- BOTTOMPERCENT
- TOPSUM
- BOTTOMSUM
- FILTER

The results of these functions are not cached in the following circumstances:

- If any of the inputs to a function contains a calculated member, named set, or inline named set.
- If the report from which the MDX statement was generated has a prompt variable defined and it is referenced within the expression.
- Expressions cached by the engine with an internal, *default* value. The use of a default value is an optimization inside the MDX engine that can conflict with the use of the expression cache.

Each entry in the expression cache has three components that uniquely identify it:

- The text of the MDX expression.
- A security key that identifies a combination of security views
- The context in which the expression was evaluated, expressed as a tuple

As with the result set cache, the intermediate results that are stored in the expression cache are security-aware and are flushed when the data cache is refreshed.

Because the expression cache is stored in memory and is so closely associated with the data cache (see 2.3.5, "Data cache" on page 54), the expression cache is stored within the space allotted to the data cache.

2.3.3 Member cache

The first action that occurs when a cube starts is to populate the member cache. All hierarchies defined in the cube model are populated and loaded into memory. The members of each hierarchy are created by running a SQL statement that returns all of the attributes for all levels within a hierarchy, including multilingual property values. The parent-child and sibling relationships inherent in the data are used to construct the hierarchical structure. The query service runs multiple hierarchy-loading SQL queries in parallel to speed up the member cache load time.

Queries to load hierarchy members are canceled when the memory is overloaded. This feature is intended for dispatchers that are configured with multiple dynamic cubes. For example, consider the scenario where an attempt is made to restart or refresh a cube just as the queries running on the query service against other cubes have temporarily pushed the memory to its limits. In this event, the operation can be stopped to avoid bringing the entire server down.

All metadata requests by the client application, the DQM query planning engine, and all member functions (for example, PARENT, CHILDREN) are serviced from this in-memory cache. Cognos Dynamic Cubes will never pose a query to retrieve member attribute values after the member cache is loaded.

The level key values stored in the member cache of a cube are also used by Cognos Dynamic Cubes as the basis for constructing filters in the SQL queries posed to the underlying relational database when populating the data cache of the cube.

After the member cache is complete, the following actions occur simultaneously:

- The cube state is changed to available and the cube is able to respond to metadata and data requests.
- If a cube has a start trigger defined, the trigger is run. This action can be delayed until after in-memory aggregates finish loading by using the `post in memory trigger` property.
- If a cube has in-memory aggregates defined and space allotted for them (a configuration property), then the queries to retrieve the data for this cache are run.

2.3.4 Aggregate cache

IBM Cognos Dynamic Cubes supports two types of pre-computed aggregate values: Those stored in database tables and those stored in the in-memory aggregate cache of a dynamic cube. The Cognos Dynamic Cube engine improves query performance by routing queries to the in-memory aggregate cache or specified aggregate database tables in preference to the underlying fact table.

In-database aggregates tend to be focused on more granular levels of detail than in-memory aggregates, although with smaller volumes than in-memory aggregates. It is entirely possible that the two types of aggregates can almost overlap, and in some cases it is possible for a modeler to construct a cube that is composed entirely of user-defined in-memory aggregates.

> **Note:** In-memory aggregates can be user-defined or Aggregate Advisor recommended. More detail is provided in this section.

In-database aggregates allow Cognos Dynamic Cubes to use database tables that contain pre-computed aggregate values. These tables, referred to as *summary* or *aggregate* tables, are created by the DBA and referenced in the cube model as *in-database aggregates.* The Aggregate Advisor tool, in the IBM Cognos DQA, can be used to suggest additional aggregate tables to those that exist in the data warehouse.

In-memory aggregates allow Cognos Dynamic Cubes to use a cache of pre-computed aggregate values. As with in-database aggregate tables, in-memory aggregates contain measure values that are aggregated by the members at the level of one or more hierarchies inside the cube. These aggregatescan be used to provide pre-computed values at precisely the same level of aggregation as the specified members. In the case of measures with distributive aggregation rules (SUM, MAX, MIN, and COUNT), in-memory aggregates can be used to compute values at a higher level of aggregation.

Another primary difference between these two types of aggregates is that implementing in-database memory aggregates requires the involvement of both the relational database DBA and the cube modeler. The reason is that in-database aggregates rely on the existence of summary tables in the underlying database. In contrast, in-memory aggregates rely solely on the specifications, known as recommendations, that are used by the aggregate cache load process.

The Aggregate Advisor can be used to suggest a collection of in-memory aggregates. In-memory aggregates can be user-defined, Aggregate Advisor-recommended, or a combination of the two. In either case, the Aggregate Advisor is used to apply the resulting recommendations to the published cube, a task that can be performed by the Cognos administrator. The recommendations are stored in the CS and take effect the next time a cube is started.

Aggregate Advisor-recommended in-memory aggregates are generated by running the Aggregate Advisor to analyze the cube structure, query workload information, or both. If user-defined aggregates have been defined in the cube model, they are also included in the Aggregate Advisor recommendations.

User-defined in-memory aggregates are defined by the cube modeler in Cognos Cube Designer and detected when Aggregate Advisor is run. User-defined in-memory aggregates are based on the modeler's knowledge of the underlying relational table structures and cube use.

Build timing for aggregate types: In-database aggregates are usually built during a predefined ETL processing window. In-memory aggregates are built during cube-start, which suggests that cubes should be started during, or immediately after, the ETL process.

Tip: The number of queries posed concurrently to populate the in-memory aggregate cache can be controlled by a cube configuration property to ensure that the underlying relational database is not saturated with concurrent requests computing summary values.

Figure 2-29 shows the loading of in-memory aggregates.

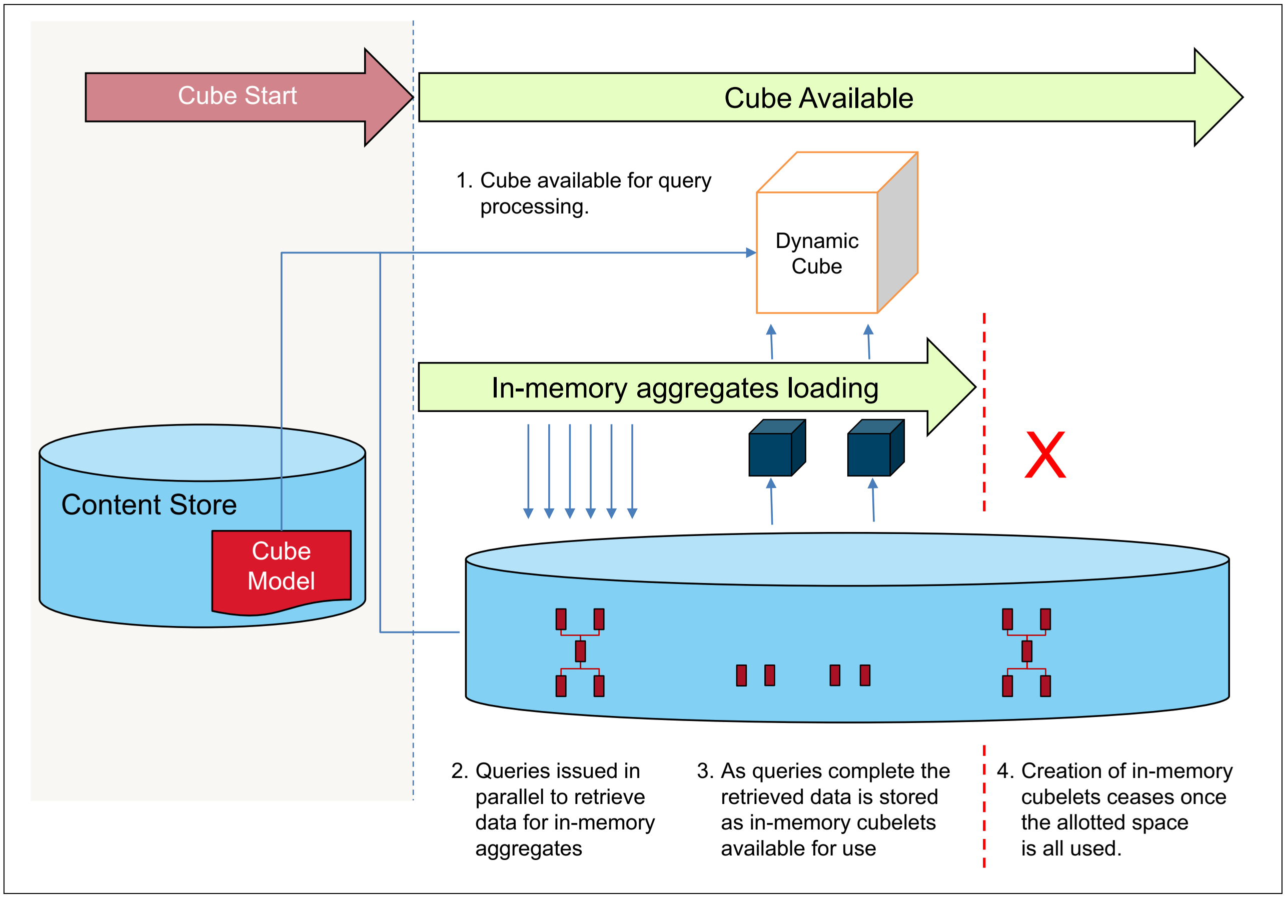


Figure 2-29 Loading in-memory aggregates

Queries that are issued by Cognos Dynamic Cubes to retrieve values for the aggregate cache are planned in the same manner as those used to populate the data cache. As a consequence, queries for aggregate cache values are able to take advantage of any available in-database aggregate tables. This feature can be used by ensuring in-database aggregates exist to support the various in-memory aggregates that are defined for a cube.

Note: Aggregate Advisor formulates in-memory aggregate recommendations and can also suggest aggregate tables to augment the loading of in-memory aggregates.

During the loading of in-memory aggregates, whenever possible, Cognos Dynamic Cubes load an in-memory aggregate from available, previously loaded in-memory aggregates. This feature is only supported for measures with associated aggregation rules, such as SUM, MAX, MIN, and COUNT. It is enabled by default. Enabling this feature helps to reduce the number of queries posed to the underlying relation database, thus reducing load on the database and in most cases, reducing the time required to load a set of in-memory aggregates. To disable this feature, add the advanced property `UseStackedAggregates` to the query service and assign it the value of `false`.

The size of the in-memory aggregate cache is specified in the configuration properties of a dynamic cube. This setting is a maximum limit. An aggregate cache size of zero disables the aggregate cache.

When starting a cube, the Dynamic Cube engine issues queries for data to populate the in-memory aggregate cache in parallel. However, the cache is loaded on a first-come basis. If an aggregate result does not fit into the cache, it is discarded. If the cache is full, no more queries are issued to retrieve aggregate values.

A dynamic cube can be configured for automatic optimization of in-memory aggregates. This configuration setting triggers a *lite* version of Aggregate Advisor to run intermittently against the cube and perform background updates of the in-memory aggregate cache while the cube is running. The automatic optimization process commences 1 hour after cube start and then runs every hour after the previous cycle completes.

2.3.5 Data cache

The data cache contains the result of queries posed by the MDX engine to a dynamic cube for data values. Each query is posed as a set of tuples and each tuple in the set specifies a single measure for which a value is required. The data obtained by each of these queries is stored in the data cache as a *cubelet*, a multi-dimensional container of data. The purpose of this cache is to store previously retrieved values for reuse in subsequent requests. Thus, the data cache is a collection of cubelets. These cubelets are continuously reordered to reduce the time that is required to search for data in the cache. The cache is also flushed of some of its data when it is nearly full, removing the cubelets that were least used to make room for more recent data.

A request for data from the MDX engine can be satisfied by values that exists in the data cache or in-memory aggregate cache. However, it is possible that the data required to satisfy the request might not be present in either cache, or that only a part can be retrieved from one of the caches. If this occurs, the dynamic cube engine obtains the data either by aggregating values available in the aggregate cache, or querying the underlying relational database.

Data inside the data cache is shared among all users, which can be done because security is applied in a layer above the data cache. This approach ensures that users are provided access only to data to which they have been granted access or to which they have not been denied access.

Both dynamic (or base) cubes and virtual cubes can be assigned a data cache. If a cube is not directly accessible by users, that is, the cube has no packages associated with it, a data cache is not necessary. However, if a cube is being used to retain historic data as part of a virtual cube, then a data cache on this base cube is useful because it ensures that the historic data remains in memory even when the latest data in another base cube is updated.

2.4 Dynamic cubes query processing

This section describes the types of query processing that can be used by the query service: Metadata and online analytical processing (OLAP).

2.4.1 Metadata query processing in the query service

Most metadata requests, either those posed by a client application (to populate a metadata browser) or those issued by the DQM query planner (to validate metadata references inside a query) are managed by the metadata framework (MFW) component of the query service.

When searches are made for members within a hierarchy or level within a metadata browser, these searches are performed by the MDX engine of the query service, which in turn obtains the required member information directly from the underlying dynamic cube.

A dynamic cube maintains all of its metadata in memory, including information about the following objects:

- Cube
- Dimensions
- Hierarchies
- Levels
- Members
- Properties
- Measures (including calculated measures)

2.4.2 OLAP query processing in the query service

OLAP query processing occurs as a result of reports and analyses being processed by the report server. The report server issues one or more corresponding queries to the query service for processing. These requests from the report server are posed in XML. When it receives a command for execution, the query service first reads the command type to determine how to dispatch the command inside the server. A query is first dispatched to the query planner, and if that is successful, it is then dispatched to the query execution engine for execution of the final query plan.

The query planner, described in 2.4.1, “Metadata query processing in the query service” on page 54, interacts with the DQM MFW component during query planning to validate (bind) metadata references inside a query, including items such as dimension and member references. The MFW4J component, in turn, interacts with the Cognos Dynamic Cubes OLAP data provider (ODP) metadata interface to obtain metadata. This component in turn retrieves the necessary metadata from the cube itself.

The outcome of successful query planning against any OLAP data source in DQM is an MDX query. In the case of Cognos Dynamic Cubes, the query is run inside the query service by using its own MDX execution engine. Following this query execution process flow, the MDX engine typically interacts with both the metadata and the query strategy interfaces of Cognos Dynamic Cubes to obtain metadata and data from a dynamic cube.

The final output of the execution of the MDX query is a cross tabular result set. The result set is then processed by the result set component of the query service, which converts the query results into a format that the report server can accept.

Figure 2-30 shows a conceptual overview of OLAP query processing in the query service.

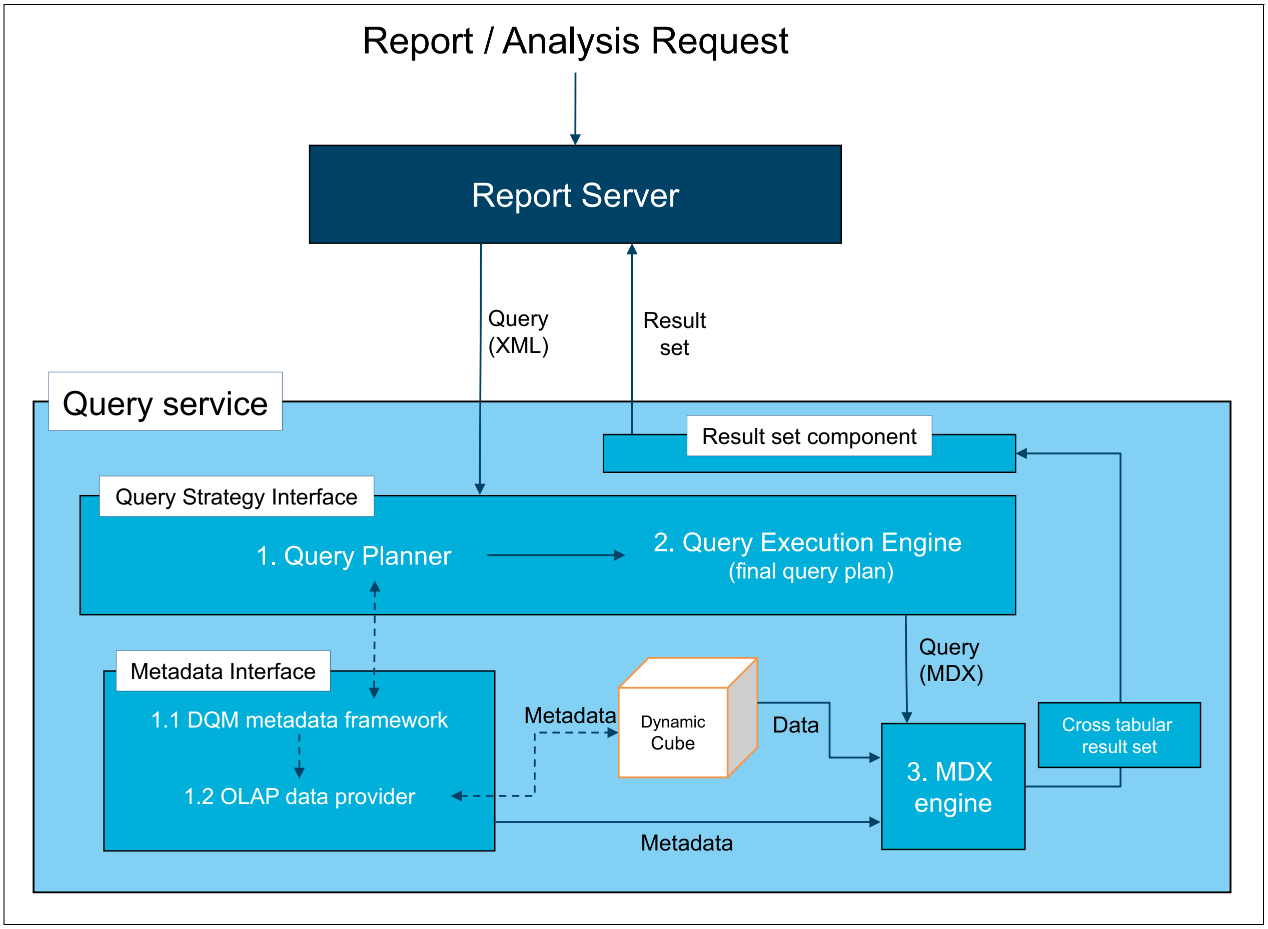


Figure 2-30 Conceptual overview of OLAP query processing in the query service

The process of querying a dynamic cube is described in more detail in the following steps:

1. A match for the posed query is searched for in the query service plan cache.
   a. If the query is found, the run tree that is stored in the cache is used and no query planning occurs.
   b. Otherwise, query planning is performed on the posted query, the result of which is an MDX run tree.
2. Cognos Dynamic Cubes searches the cube result set cache for a match to this MDX query.
   a. If a match is found and the security views of the result and the current user also match, the result set is returned to the query service for further processing.
   b. Otherwise, the MDX engine runs the run tree, running nodes as it *walks* the tree.
3. Members and member metadata are retrieved from the metadata cache of the cube to satisfy particular nodes of the run tree.

4. For particular nodes, such as FILTER and TOPCOUNT, the cube expression cache is checked for previously computed results.
   a. If none are found, the expression is evaluated and if applicable, the result set is added to the expression cache.
   b. When a run tree node requires data from a dynamic cube, a request is made to the cube.
5. Cognos Dynamic Cubes first checks the cube data cache for cubelets that contain the data required to satisfy the data request.
   a. If all of the data required is found, a result set is returned to the MDX engine containing all of the requested data.
   b. Otherwise, Cognos Dynamic Cubes examines the in-memory aggregate cache of the cube. The cubelets inside the in-memory aggregate cache are used only if they are able to provide values for all of the tuples within the remaining data request. That is, the tuples must contain all members in each hierarchy that are from the same level.

      A cubelet from the in-memory aggregate cache can be used if it matches the levels of the tuples for the requested data values, or if the data in the cubelet can be rolled up to the level requested. If aggregation is required, the resulting values are stored in the cube data cache for reuse by subsequent queries.

      There are actually two searches performed: The first search for an exact level match and, if that fails, a second search for cubelets that can be rolled up. If there are multiple cubelets that can be used, Cognos Dynamic Cubes uses a heuristic algorithm to determine the most efficient cubelet to use.
6. If there are still tuples for which data has not been retrieved, Cognos Dynamic Cubes then queries the underlying relational database for the remaining values. The tuples resulting from this query are organized into groups of identical levels for each hierarchy.
7. Cognos Dynamic Cubes examines the cube metadata for in-database aggregates to determine whether there is an aggregate table in the database that can be used to obtain the required values for each query group. Queries are then directed to aggregate tables or the original fact table, as required.
8. Data retrieved from the database is stored in cube data cache and then added to the result set that is returned to the MDX engine.

The flow from step 3 through step 8 is repeated recursively, following on the structure of the run tree, until the final result is computed.

Caching considerations during OLAP query processing.

Each cache inside a cube attempts to provide data values for the unresolved portion of a query after the previous cache finishes attempting to provide data:

- The caches at the beginning of this processing flow are coarse grained, smaller, and provide less reuse and sharing. For example, an entry in the result set cache contains only the output of an MDX query and can be shared only by users with the same combination of security views as the user who first ran the query.
- The caches at the end of the flow are more fine grained, larger, and support more reuse and sharing. For example, the in-memory aggregate cache might contain gigabytes of detailed data, but this data can be shared among all users regardless of their assigned security privileges. The reason being that the data retrieved from this cache passes through more security layers before it is matched to a result set.

Because the in-memory aggregate cache is preinstalled into memory, its contents are typically based on prior workload. Because this data can be further aggregated, the in-memory aggregate cache can be thought of as the primary cache of a dynamic cube. This is not the case for virtual cubes, as they do not have their own in-memory aggregate caches, and instead they rely on the in-memory aggregate caches of their base cubes.

If a cube is not assigned an in-memory aggregate cache, then the data cache becomes the primary cache. Otherwise, the data cache plays a secondary role of caching the values that do not directly access the in-memory aggregate cache during OLAP query processing.

During general query execution, queries that are posed to virtual cubes are decomposed into separate queries for each of the underlying cubes. The results of these decomposed queries are then merged using the merge rule that is specified in the definition of the virtual cube. The benefit of virtual cubes becomes most apparent when observing query decomposition of requests for data values against this kind of cube. For example, if a measure from a virtual cube is present in only one of the two underlying (base) cubes, then only that cube is queried for its corresponding measure value.

Figure 2-31 shows query decomposition with virtual cubes.

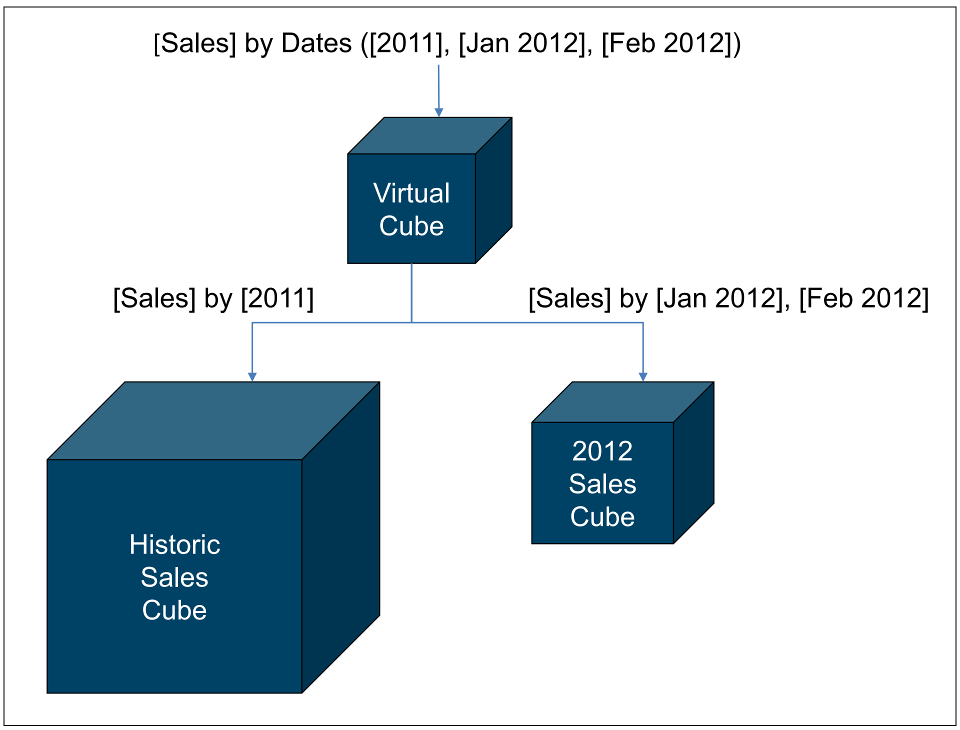


Figure 2-31 Query decomposition with virtual cubes

2.5 Using the database to evaluate set expressions

For particular MDX functions, given a sufficiently large input set, Cognos Dynamic Cubes pushes the evaluation of the expression down to the underlying database for execution. This feature is called *query pushdown* and allows the database to be used to evaluate large set expressions.

Without this feature, those MDX functions, such as TOPCOUNT, would retrieve large sets of data from the database and perform the function in the MDX engine. The transfer of the large data sets, which can be in the range of millions of rows of data, can degrade query performance.

The intent of query pushdown is to use the proximity of the relational database management system (RDBMS) to the data, allowing the operation to be performed without requiring the transfer of the data to the MDX engine. The effect of this feature is similar to that of a cache, in that it provides faster response time and has lower and more consistent resource requirements.

Query pushdown is supported for the FILTER, TOPCOUNT, and TOPSUM functions, but is not supported in the following cases:

- Sets of members from different levels of the same hierarchy
- Parent-child hierarchies
- Semi-aggregate measures
- Measures with an UNKNOWN aggregation rule
- Complex numeric expression in the `count` parameter of the TOPCOUNT function
- Calculated members and measures in the numeric expression
- Sets containing both negative values and nulls

Pushdown is also not supported for queries that are posed to virtual cubes. Keep in mind that virtual cubes do not have direct access to the underlying database.

2.6 Management of Cognos Dynamic Cubes

DQM accepts most of its commands from the report servers by using an XML protocol through a single port open for all such requests, including report execution and metadata retrieval. DQM, as a Cognos service (the query service), also accepts BI bus requests, notably those that are used to manage dynamic cubes. In addition, DQM also supports JMX, which is used to implement portions of the Cognos Administration interface for managing requests.

When any of the cube management requests are received by the query service, they are directed to the Cognos Dynamic Cubes engine, which then dispatches the request (command) to the specified cube. The following are examples of cube management requests:

- Start cube
- Stop cube
- Restart cube
- Refresh metadata cache
- Refresh data cache
- Refresh security
- Clear workload log
- View messages

Cognos Dynamic Cubes ensures that concurrent administrative commands are handled in a logical fashion and that concurrent user report executions behave correctly when any of these commands occur. For example, a request to refresh the metadata cache of a cube fails immediately if that cube is being stopped.

Looking at this example, when the member cache of a cube is refreshed, queries continue to run by using the existing member cache while a new member cache is constructed in the background. When the new cache is constructed, all new queries reference the new member cache. Any queries that are running when the new cache is made available continue to use the old cache. The old cache is only discarded after all queries that reference it have completed running.

Although this approach does require extra memory (enough memory to store two versions of the member cache in memory at one time), user queries can continue to operate normally and with consistent performance while the member cache is being refreshed.

The process of refreshing the data cache is straightforward. Because the data cache is constructed on demand, a request to refresh the data cache results in the construction of a new cache. All new queries use (and populate) the new data cache, while queries that were running at the time of the refresh continue to use the old cache. The old data cache is then discarded when all queries that reference it complete execution.

Refreshing the data cache affects the other caches of the cube. Because the values in the data cache are aggregated relative to the members in the member cache, a refresh of the member cache implicitly causes a refresh of the data cache.

Similarly, when the data cache is refreshed, the expression cache and the result set cache are also implicitly flushed. The reason is that the expression and result set caches are contained within the data cache and must be synchronized with the data cache.

Finally, the in-memory aggregate cache is flushed and the queries that are used to populate the aggregate cache are run again. While the queries that load the in-memory aggregate cache are running, any user queries that are posed during that time are unable to take advantage of these aggregates. The in-memory aggregates are made available to subsequent queries after the data for a particular in-memory aggregate has been loaded.

2.6.1 Memory usage

The Cognos administrator must determine the total amount of space required for all cubes on a dispatcher and ensure that the query service is tuned accordingly, with sufficient resources to start and query the cubes. Each query service is a JVM process with its own Java heap. Each instance of a cube configured against a dispatcher is allocated space in the overall Java heap of the dispatcher's associated query service.

The query service not only hosts dynamic cubes, but also monitors memory usage and queues or cancels queries against the cubes. Memory allocations within the query service Java heap are managed by configuring the in-memory cache and data cache limits for each cube. Overall heap size is managed using the JVM heap settings on the query service.

Figure 2-32 shows cubes configured against multiple dispatchers and the running cube instances hosted within the associated query service JVM of each dispatcher.

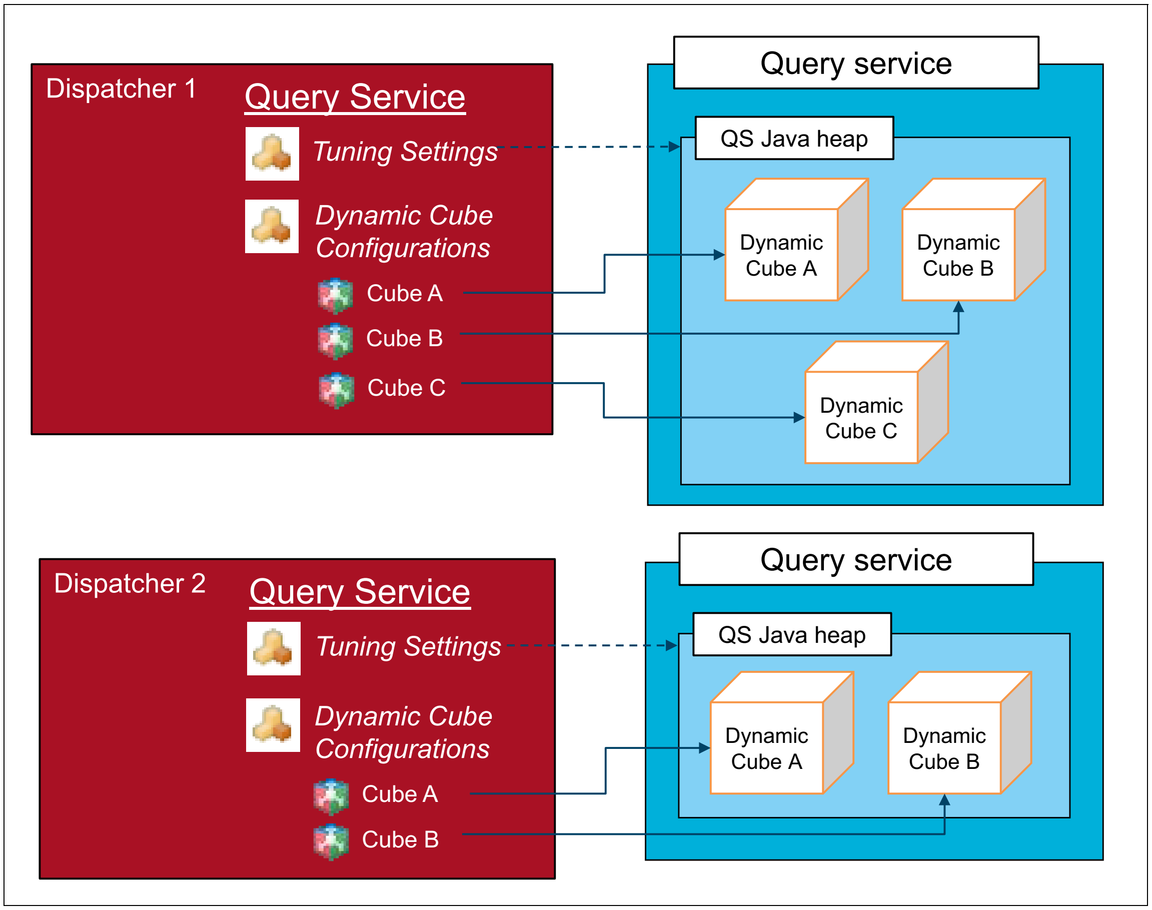


Figure 2-32 Cube instances hosted in the query service JVMs of different dispatchers

There is no governor setting for limiting the amount of memory allotted to the member cache of a dynamic cube. However, it is possible to estimate the size of this cache accurately. In some cases, the member cache size can also be reduced by allowing multiple cubes to share the member cache of one or more dimensions. This behavior can be established for all cubes that use the same dimensions and are hosted on the same dispatcher.

It is also not possible to set a governor limit for the temporary query space associated with the execution of queries against a dynamic cube. The temporary query space is used to create an intermediate member or cross-join sets and result sets during query processing. The amount of space that is required varies according to the types of reports or analyses involved and the nature of the queries themselves. If necessary, query execution can also use any unused in-memory aggregate cache space allocation. However, the query service monitors memory usage, and if the memory usage exceeds 90% of the available JVM heap, the query service begins canceling queries to reduce amount of memory used. Therefore, a resource-intensive query can be canceled during the regular course of business, yet can run successfully in off peak hours when more memory is available to individual queries.

Figure 2-33 shows memory allocations in the Java heap of a query service.

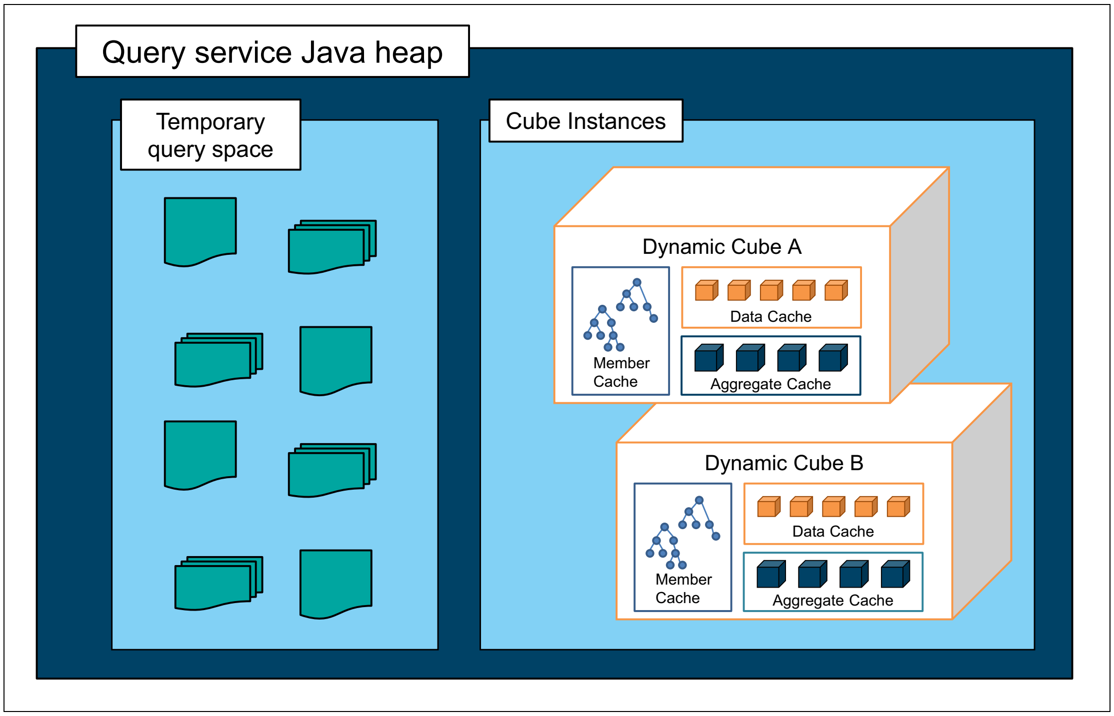


Figure 2-33 Memory allocations in the Java heap of a query service

Both the member and in-memory aggregate caches are loaded fully when a cube starts and remain in memory until the cube is stopped. The data cache is loaded on demand as queries against the cube are processed. This cache has the capacity to expand and contract over time. The size of the data cache is monitored continually by the Cognos Dynamic Cubes engine and will be partially flushed, giving priority to the most used data. Data cache flush occurs at 90% capacity or in accordance with the query service advanced environment settings back to 60%.

The Java heap metrics for each query service are presented in the System tab of Cognos Administration. The query service dynamic cube management and Java heap tuning settings should be monitored regularly and adjusted where necessary to ensure that sufficient resources are available to support the cubes.

Figure 2-34 shows the Java heap metrics for a query service.

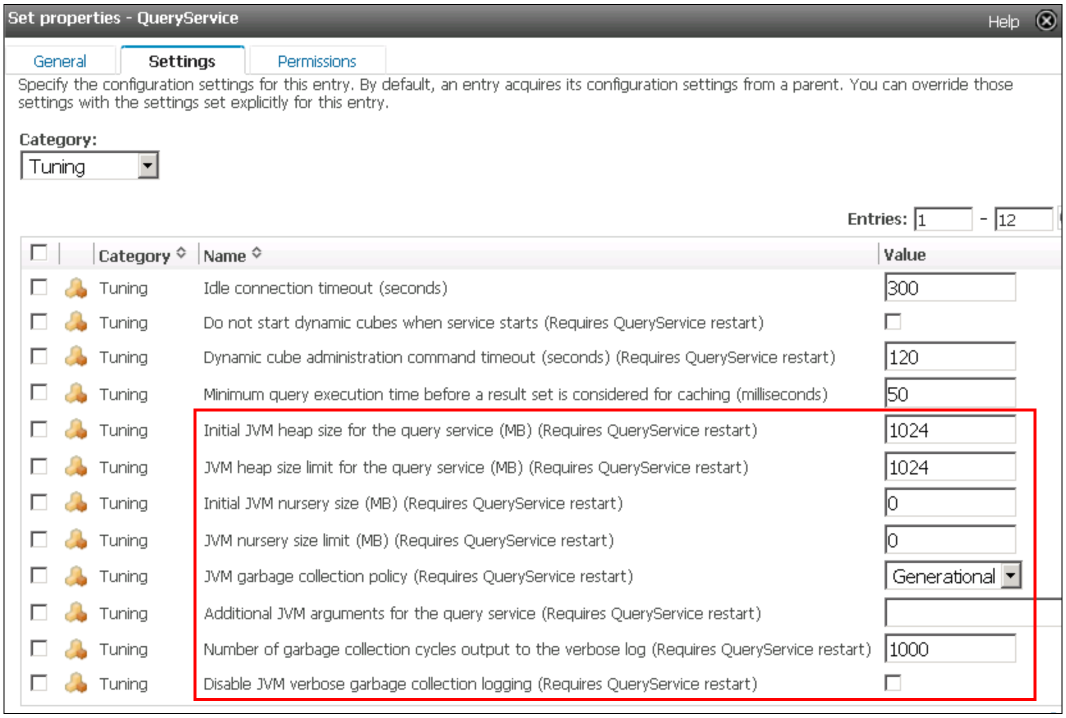


Figure 2-34 JVM heap metrics for a Query Service

Figure 2-35 shows where to find the in-memory and in-database cache limit settings for a dynamic cube instance.

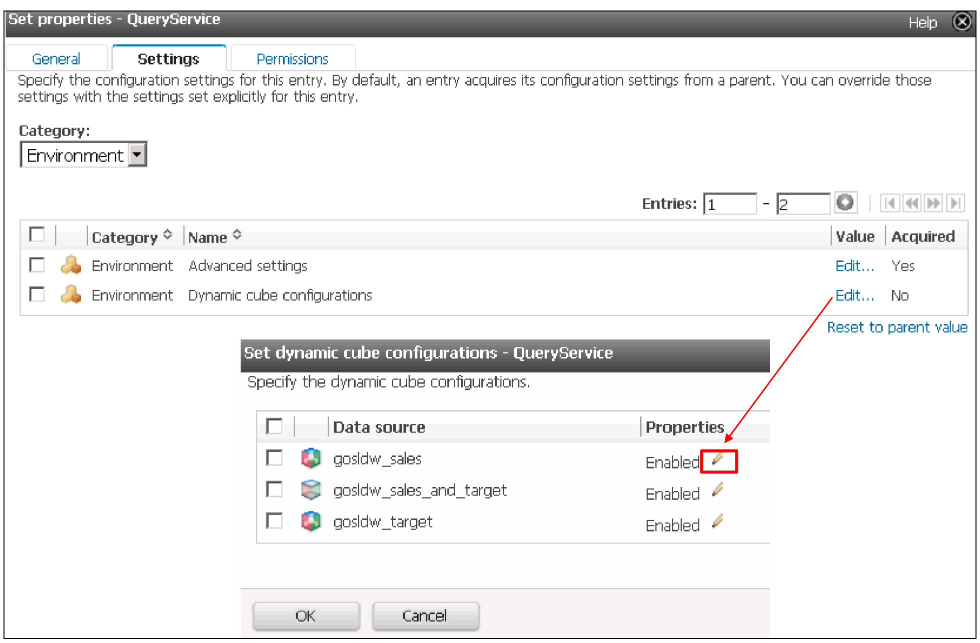


Figure 2-35 Setting in-memory and in-database cache limits for a dynamic cube

2.6.2 Data currency

Dynamic cubes can be updated on a near-real time basis. However, they are likely to be stopped and started regularly around scheduled ETL data loads. Their heavy reliance on strict structural rules in the underlying data warehouse mean that ETL processes are an important consideration in managing dynamic cubes.

Aggregate tables contribute significantly to the efficiencies of the Cognos Dynamic Cubes engine, but can also add to the amount of ETL processing required. Building these tables in layers to create stacked aggregates can reduce the constraints imposed by a fixed ETL time window. Stacked aggregates can reduce ETL query time by ensuring that each table has significantly fewer records that the one on which it is based. This concept is described in more detail in “ETL window impacts” on page 423.

2.6.3 Communication

Aligning the needs of dynamic cube administrators, modelers, and business users with an organization’s corporate, IT, and information management strategies is an essential consideration for managing a solution that includes Cognos Dynamic Cubes. The best gains from Cognos Dynamic Cubes are obtained when there is effective communication between all parties in this mix.

The following people need to work together, understand the vernacular, and reach a common ground:

- Cube modelers: They are affected by and affect the other parties.
- Relational DBAs: They must understand dimensions and OLAP structure, and also star schema structures and data warehouse design.
- Report authors: They must be able to use member functions and understand OLAP constructs.
- Cognos administrators: They must become familiar with managing the actual dynamic cube data source.
- Combined DBA and ETL involvement: Implement in-database aggregate recommendations and other tuning practices.
- Combined DBA and security administrators involvement: Implement table-based security filters.

Making the investment in devising and maintaining these channels of communication are a key contributor to the establishment of a reliable and manageable Cognos Dynamic Cubes solution.

3

Installation and configuration of IBM Cognos Cube Designer and IBM Cognos Dynamic Query Analyzer

This chapter helps you prepare your environment and install IBM Cognos Cube Designer and IBM Cognos Dynamic Query Analyzer so you can use IBM Cognos Dynamic Cubes in your Cognos BI environment.

For the most recent information about Cognos Dynamic Cubes installation and configuration, see the *Dynamic Cubes Installation and Configuration Guide 10.2.2* in the Cognos Business Intelligence 10.2.2 Product Documentation web page at:

http://www-01.ibm.com/support/docview.wss?uid=swg27042003

For the most recent information about Dynamic Query Analyzer installation and configuration, see the *Dynamic Query Analyzer Installation and Configuration Guide 10.2.2* in the Cognos Business Intelligence 10.2.2 Product Documentation web page at:

http://www-01.ibm.com/support/docview.wss?uid=swg27042003

This chapter contains the following sections:

- Introduction
- Cognos Cube Designer operating requirements
- Cognos Dynamic Cubes hardware requirements for the BI Server
- Installing IBM Cognos Cube Designer
- Installing Cognos Dynamic Query Analyzer

3.1 Introduction

Fully understanding the various components of IBM Cognos 10.2.2 and its architecture is important. All the components are described in the documentation that is included with the software. Also, review Chapter 2, "IBM Cognos Dynamic Cubes architecture" on page 13, which explains in detail the architecture of Cognos Dynamic Cubes.

The remaining chapters in this book refer to Cognos Dynamic Cubes sample data. This sample data is available in the IBM Cognos 10.2.2 installation files and should be installed by the Cognos administrator.

For information about purpose, content, location, installation, and setting up the IBM Cognos Business Intelligence samples, see the IBM Knowledge Center at:

http://www-01.ibm.com/support/knowledgecenter/SSEP7J_10.2.2/com.ibm.swg.ba.cognos.inst_cr_winux.10.2.2.doc/c_instsamplesoverview.html

3.2 Cognos Cube Designer operating requirements

Before you use Cognos Dynamic Cubes, prerequisite components must be installed and running to ensure a successful implementation.

This section outlines the key components that are necessary before you install Cognos Cube Designer.

3.2.1 IBM Cognos 10.2.2 Business Intelligence Server

Cognos Dynamic Cubes is a component of IBM Cognos 10.2.2 Business Intelligence, so it requires having an existing IBM Cognos 10.2.2 Business Intelligence Server in the network. At a minimum, the Business Intelligence Server should have a gateway, an application tier, and a content store readily available.

The workstation where Cognos Cube Designer will be installed must have network access to the Business Intelligence Server. Authenticating in IBM Cognos Connection through a web browser is a good way to test the connectivity.

IBM Cognos 10.2.2 Business Intelligence Server is available in a 32-bit and 64-bit version. To be able to use Cognos Dynamic Cubes, the 64-bit version of the server must be installed.

Also, ensure that JDBC connectivity is configured to your data warehouse from the BI server. Consult the Business Intelligence Installation and Configuration Guide at:

http://www-01.ibm.com/support/docview.wss?uid=swg27042003

3.2.2 IBM Cognos 10.2.2 Report Server

IBM Cognos 10.2.2 Business Intelligence provides a 32-bit and 64-bit version of the report server. By default, the report server is set to use the 32-bit version only.

Because Cognos Dynamic Cubes is based on the Dynamic Query Mode, if you do not need to support a 32-bit technology such as a PowerCube, enable the 64-bit version of the report server as shown in Figure 3-1.

Complete the following steps to enable the 64-bit report server:

1. Go to the IBM Cognos Configuration.
2. In the Explorer window, click **Environment**.
3. For the **Report Server execution mode**, select the **64-bit** option.
4. Save the Cognos Configuration and restart the IBM Cognos Service.

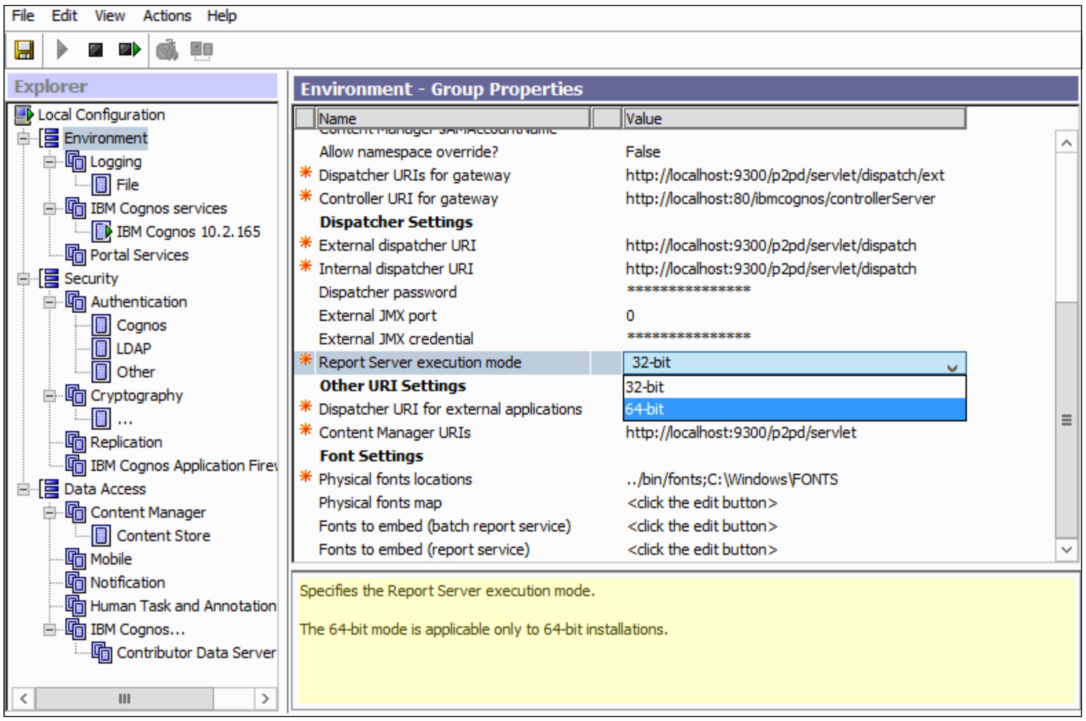


Figure 3-1 IBM Cognos Configuration Report Server execution mode (64-bit)

3.2.3 Java Runtime Environment Libraries

Cognos Dynamic Cubes requires IBM Java 6 SR 10 FP1 or newer version of the Java Runtime Environment (JRE) libraries. IBM Cognos 10.2.2 Business Intelligence Server on Microsoft Windows automatically installs an IBM version of the JRE.

3.2.4 Supported relational databases

For the most recent information about data warehouse platforms (relational databases) that are supported for use with Cognos Dynamic Cubes, see *Cognos Business Intelligence Supported Software Environments* at:

http://www-01.ibm.com/support/docview.wss?uid=swg27042164

Select the tab that corresponds to your Cognos Business Intelligence version, for example, **10.2.2**, and then click **Software (including application servers, data sources, and web browsers)**, select the Supported Software tab, and click **Data Sources**.

3.2.5 Cognos Cube Designer supported operating systems

For current version information of supported operating systems for Cognos Cube Designer, see the IBM Support website that describes supported environments. Select your Cognos Business Intelligence version and select Requirements by Platform. Select Windows (Cube Designer is only supported on Windows) and then hover over the version you are using.

http://www.ibm.com/support/docview.wss?uid=swg27014432

3.3 Cognos Dynamic Cubes hardware requirements for the BI Server

For more information about hardware requirements, see *Understanding Hardware Requirements for Dynamic Cubes* in the Business Analytics proven practices website at:

http://www.ibm.com/developerworks/analytics/practices.html

You can also use the hardware sizing calculator from Cognos Cube Designer. For more information, see 16.2, “Cube Designer and Framework Manager paradigms compared” on page 520.

3.4 Installing IBM Cognos Cube Designer

The most common method of installation for IBM Cognos Cube Designer is called *attended installation*. This method involves the use of the Cognos Cube Designer installation wizard that guides you through all of the necessary steps to correctly install the application on the workstation.

This book covers only the attended installation method. For instructions to use the unattended installation method, see IBM Cognos Dynamic Cubes Version 10.2.2 Installation and Configuration Guide at:

http://www-01.ibm.com/support/docview.wss?uid=swg27042003

Complete the following steps to install Cognos Cube Designer:

1. Start the installation by going to the location of the installation files in the `root` folder where the `iso tar` file is extracted. Right-click **Autorun.inf** and select **Install** (Figure 3-2).

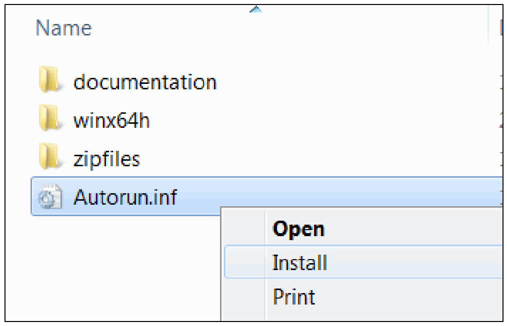


Figure 3-2   Run Autorun.inf

2. Depending on the security level that is set at the workstation, you might be prompted to confirm the correct file (Figure 3-3). Click **Yes**.

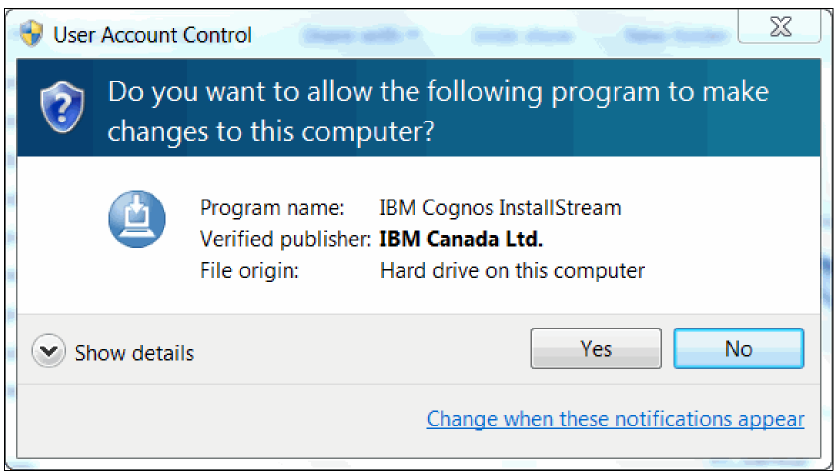


Figure 3-3   Security warning

An IBM welcome window opens (Figure 3-4), followed by the first option of the installation wizard (Figure 3-5).

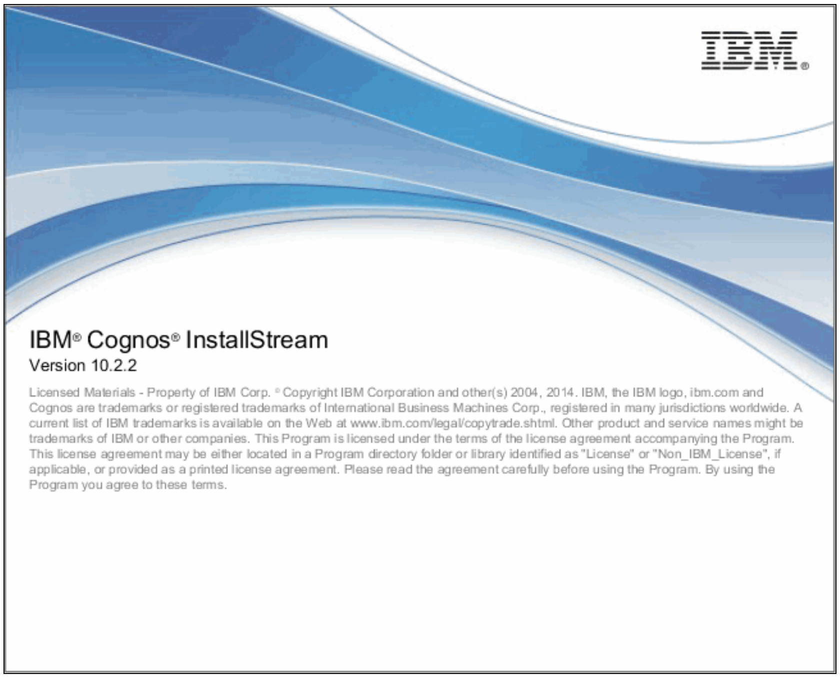


Figure 3-4 IBM Cognos InstallStream welcome window

3. Select a language, and then click **Next** (Figure 3-5).

   Notice on this window that you can click the **Installation Guide and Release Notes** link to view the manual.

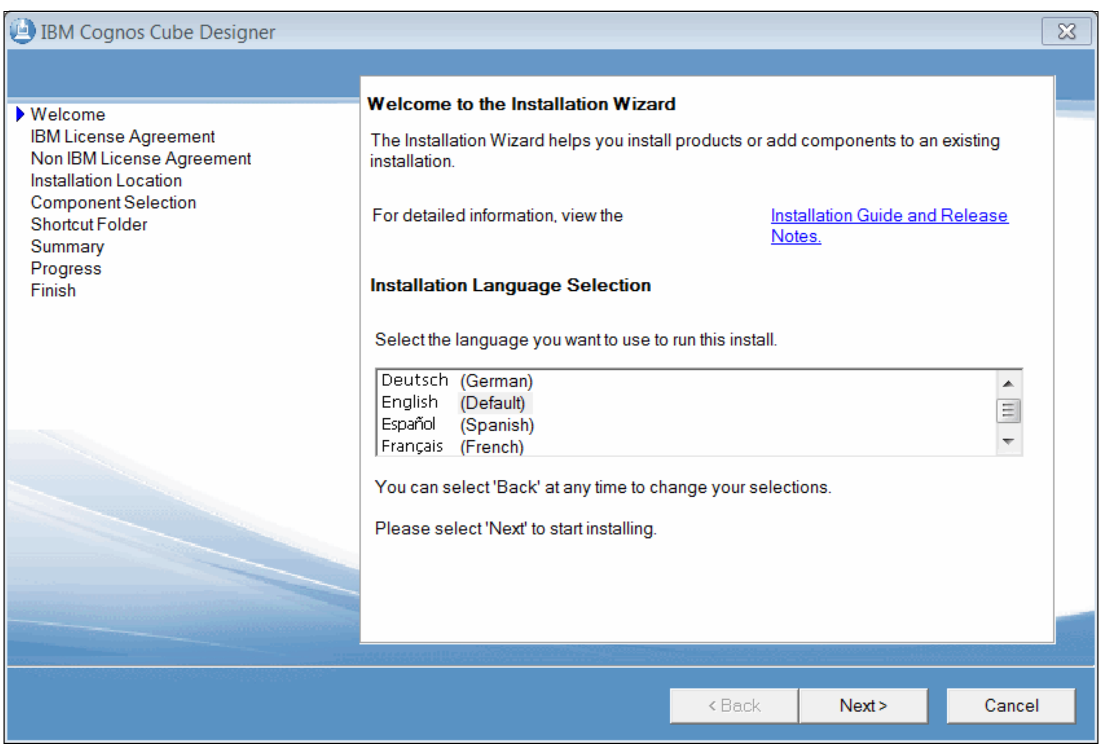


Figure 3-5 Language selection

4. The IBM License Agreement window opens (Figure 3-6). Read the terms of the license agreement. If you agree with these terms, select **I Agree**, and then click **Next**.

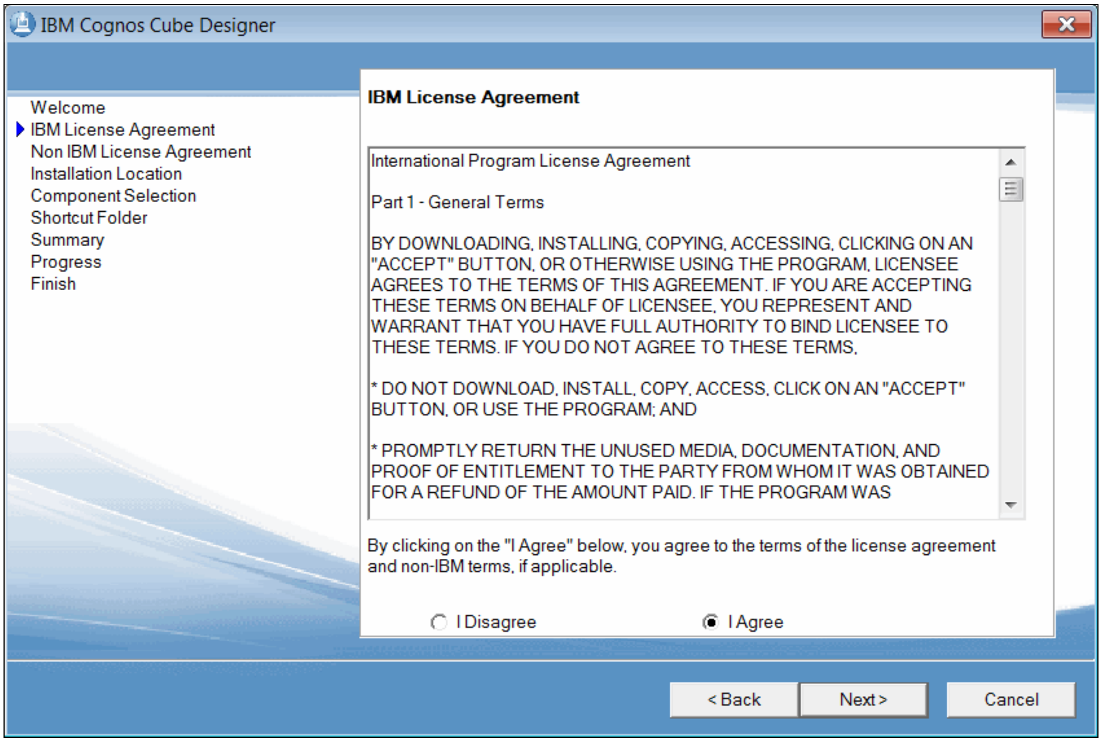


Figure 3-6 IBM License Agreement

The first window of the installation wizard opens (Figure 3-7) that lists the installation location.

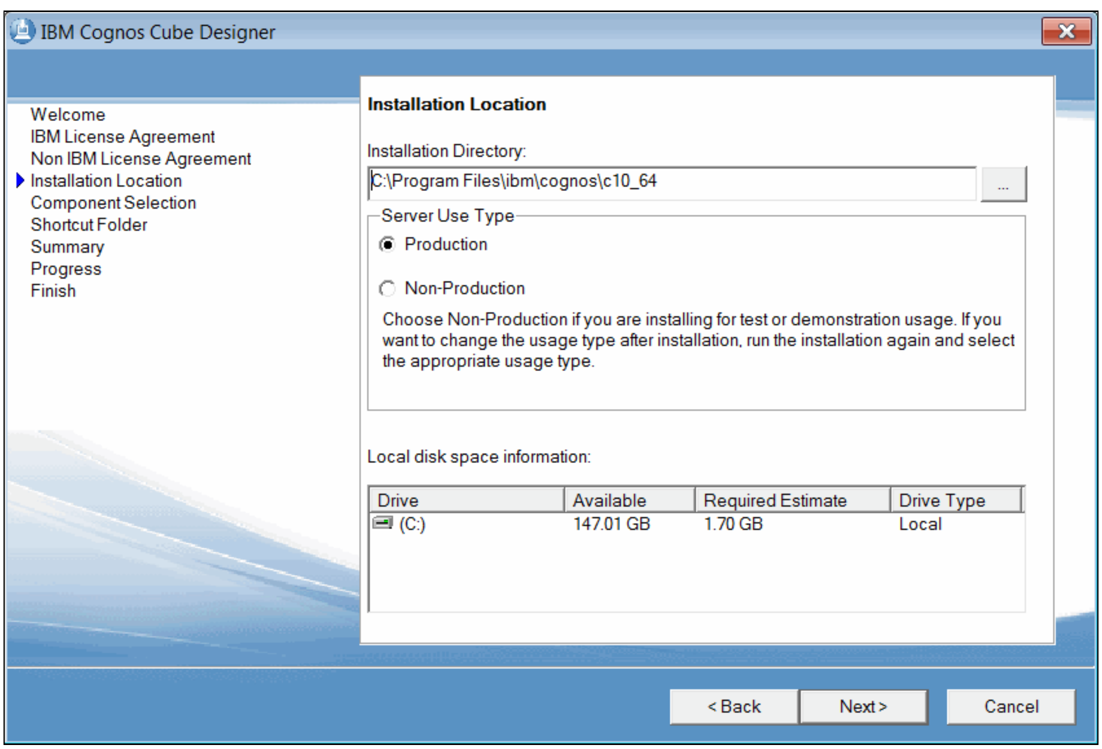


Figure 3-7 Installation location

5. In this window, you can change the installation location. By default, the installation directory is `C:\Program Files (x86)\ibm\cognos\c10`. Some business requirements might determine the use of a different directory. For example, some organizations allow only the operating system to be on the `C` drive, whereas other applications, such as Cognos Dynamic Cubes, must be installed on a secondary drive. Use this window to change the default location if necessary.
6. The Component Selection window opens (Figure 3-8). Cognos Dynamic Cubes is a single component named IBM Cognos Cube Designer. This option is selected by default. Click **Next**.

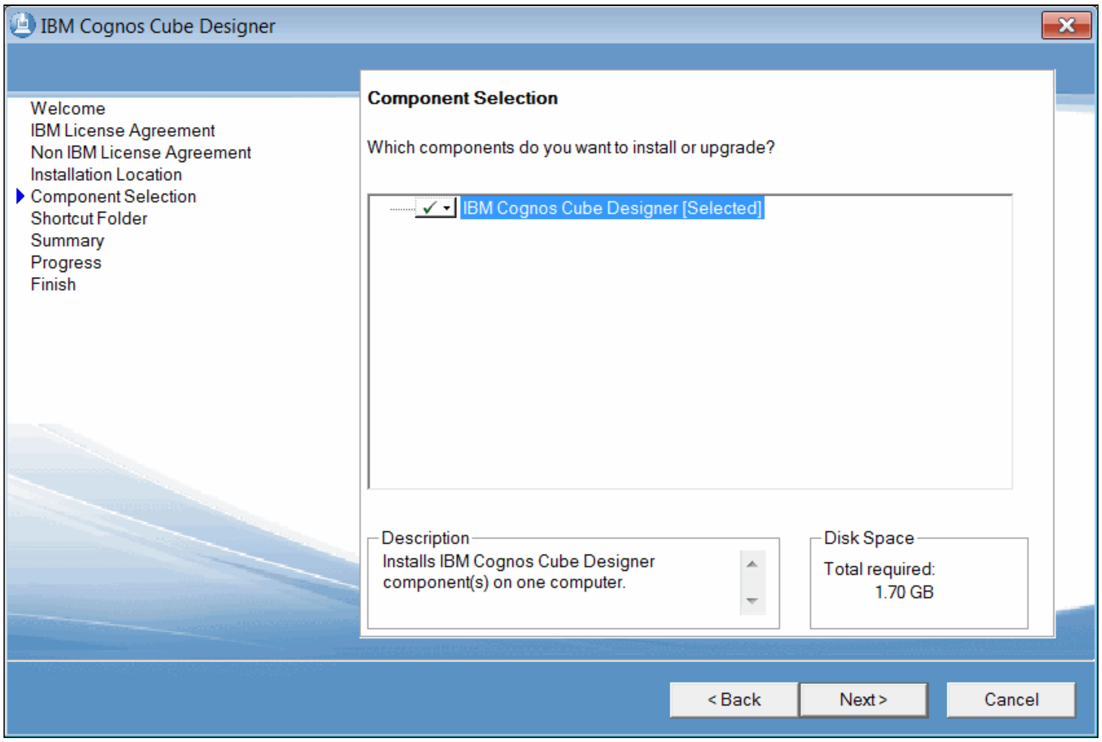


Figure 3-8 Component selection

7. The Shortcut Folder window opens (Figure 3-9). In this window, you can select the Start Menu folder where you want to install the Cognos Cube Designer application. By default, the Cognos Cube Designer shortcut is placed in the IBM Cognos 10 folder.

   In this window, you can also make available the Cognos Cube Designer shortcut to all users for that workstation, or you can limit the shortcut to only the user account that you are using to install the application. Security is the main reason why some organizations might want to limit the shortcut to only the installation user, especially when multiple users have access to the same workstation.

   When you finish with your selections, click **Next**.

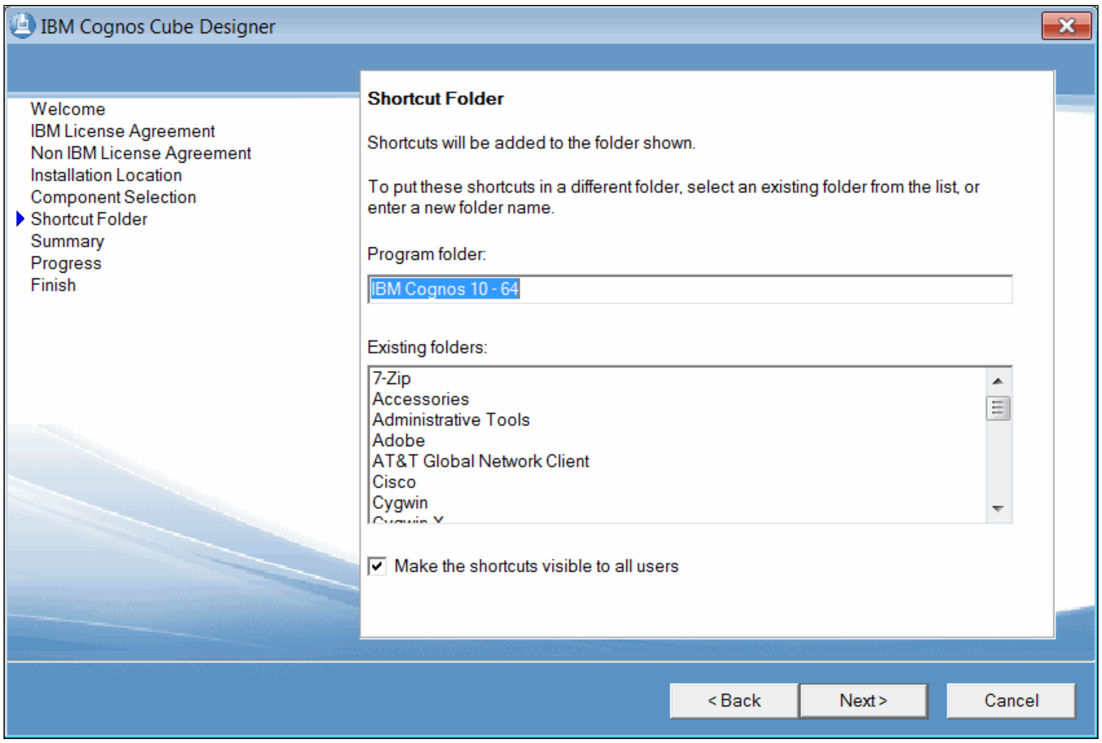


Figure 3-9 Shortcut folder

8. The Installation Summary window opens (Figure 3-10). It lists all the options that you selected during the installation for you to review. If you want to change a setting, click **Back** to go through all the options of the installation wizard. Make any necessary changes and return to the installation summary. When you are satisfied with your selections, click **Next**.

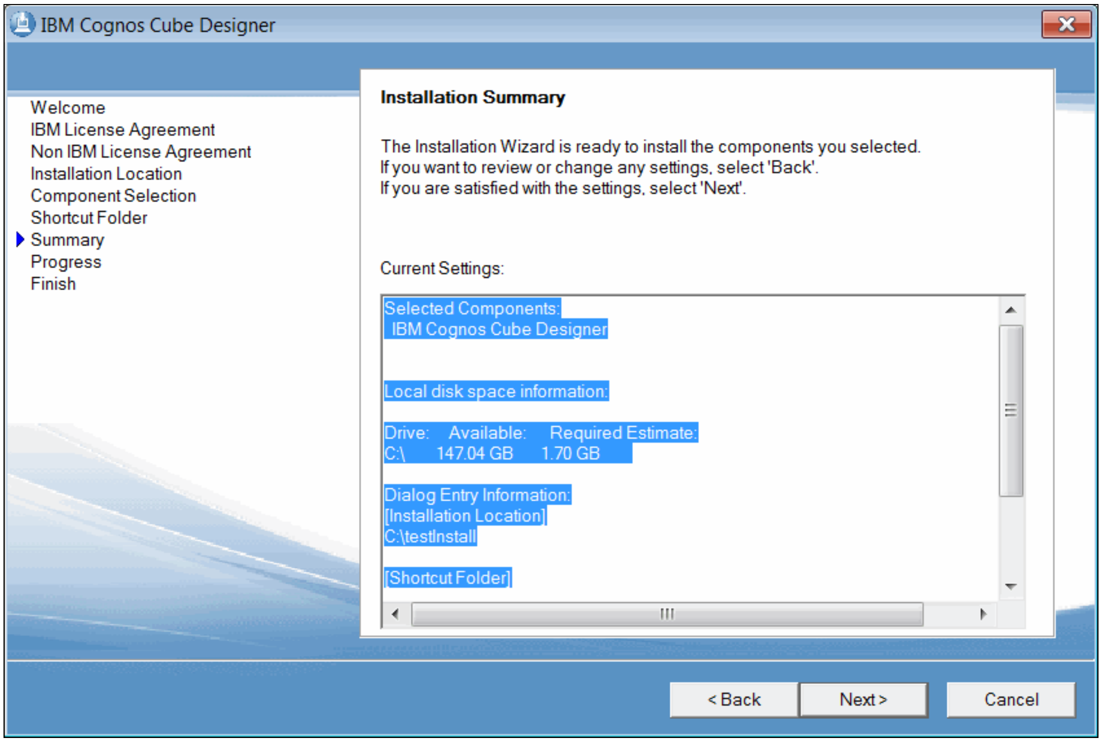


Figure 3-10 Installation summary

The installation wizard begins the installation on the workstation. A progress window opens (Figure 3-11).

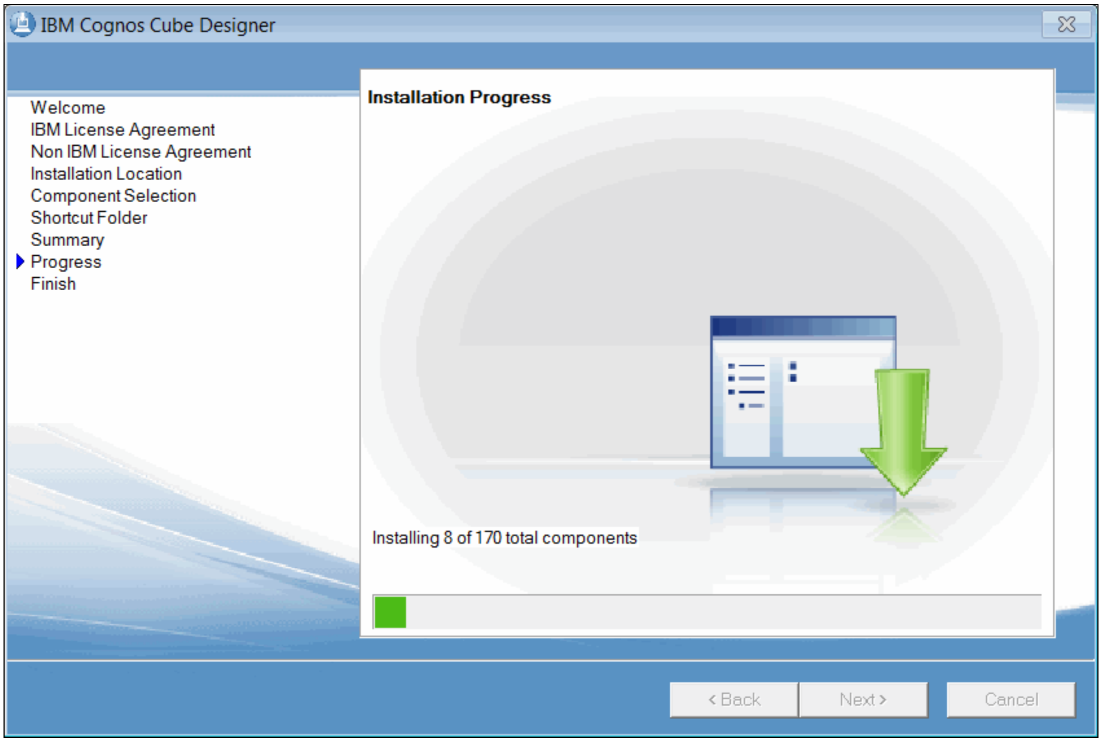


Figure 3-11 Installation progress

9. When the installation is complete, a Finish window opens (Figure 3-12). You can review the transfer log and summary-error log. Click **Finish** to complete the installation.

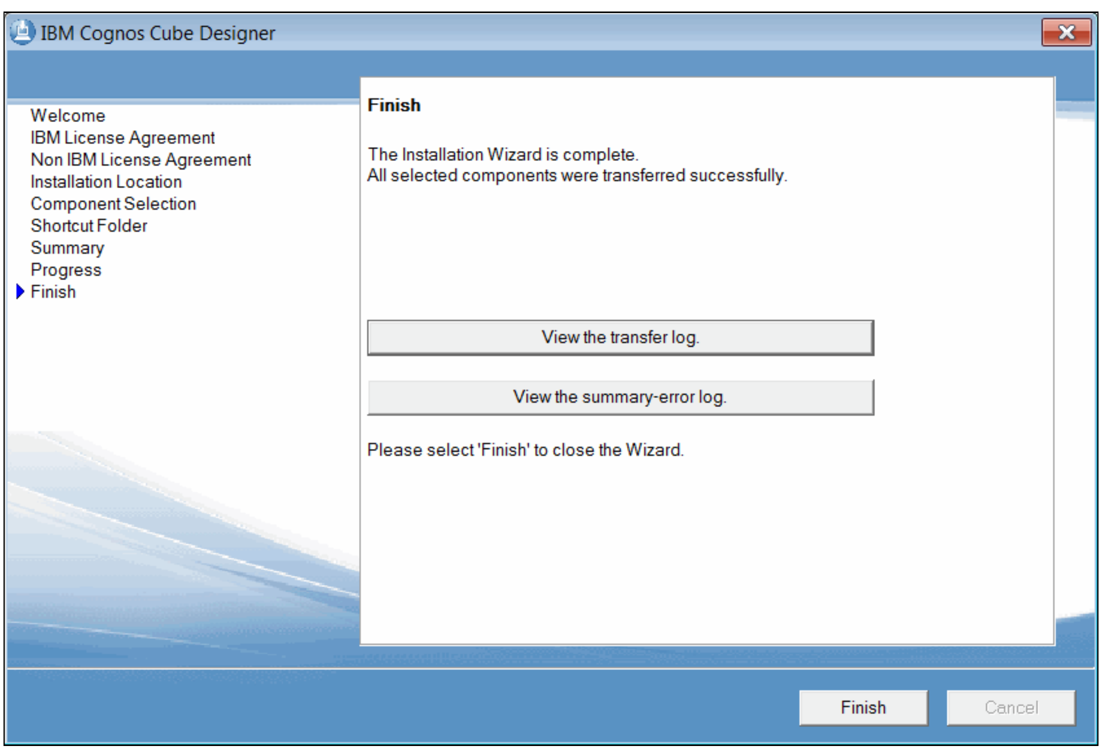


Figure 3-12 Finish window

3.5 Installing Cognos Dynamic Query Analyzer

Dynamic Query Analyzer (DQA) is a troubleshooting application that analyzes the query logs generated by dynamic cube query requests. It provides a straightforward graphical interface to help you understand and analyze queries for performance tuning.

Dynamic Query Analyzer can be installed on Microsoft Windows or Linux. It is available in 32-bit and 64-bit versions on Windows and 64-bit only on Linux. This book only describes the Microsoft Windows 64-bit version. The Microsoft Windows 32-bit version uses the same installation method, except that the default location is different.

If you want to install the Linux version of the Dynamic Query Analyzer, see the installation guide that is included with IBM Cognos BI 10.2.2.

3.5.1 Cognos Dynamic Query Analyzer installation requirements

DQA can be installed locally on, or remotely from, your IBM Cognos BI Server. The installation folder can be independent or shared with other IBM Cognos software components. There are no inherent dependencies. Consult the *Dynamic Query Analyzer Installation and Configuration Guide 10.2.2* for available configuration options:

http://www-01.ibm.com/support/docview.wss?uid=swg27042003

3.5.2 Installation procedure

Complete the following steps to install Dynamic Query Analyzer:

1. Go to the location of the installation files in the `root` folder where the `iso tar` file is extracted. Right-click **Autorun.inf** and select **Install** (Figure 3-13).

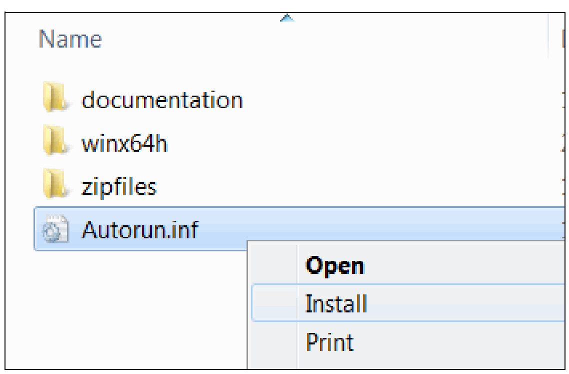


Figure 3-13 Autorun.inf

2. Depending on the security level set at the workstation, you might be prompted to confirm the correct installation file (Figure 3-14). If so, click **Yes**.

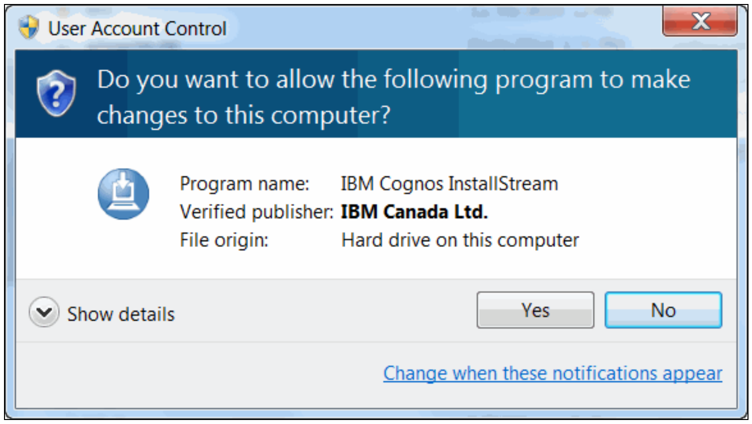


Figure 3-14 Security warning

An IBM welcome window opens (Figure 3-15), followed by the first option of the installation wizard (Figure 3-16 on page 78).

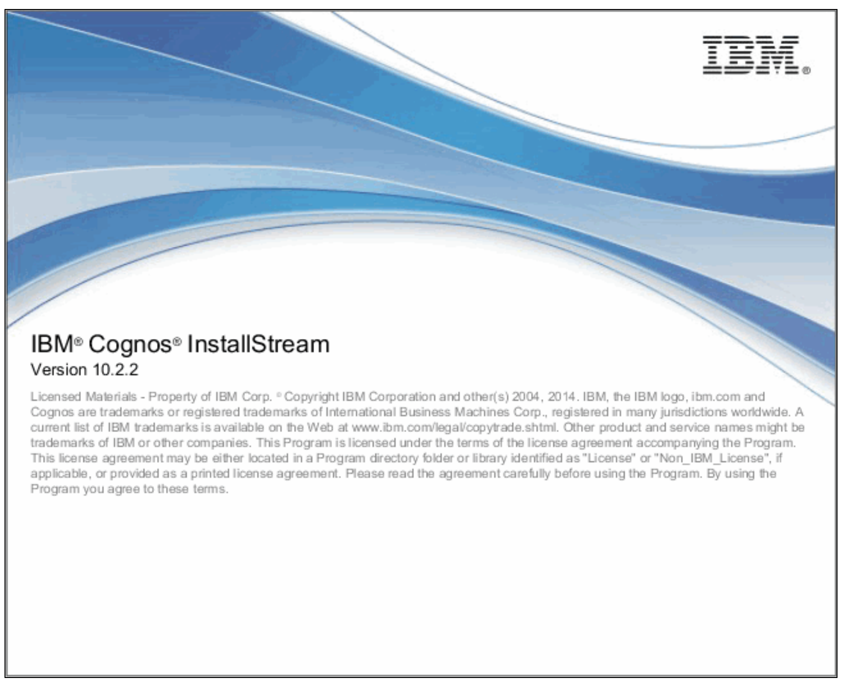


Figure 3-15 IBM Cognos InstallStream welcome window

3. Select a language to use during the installation and click **Next** (Figure 3-16).

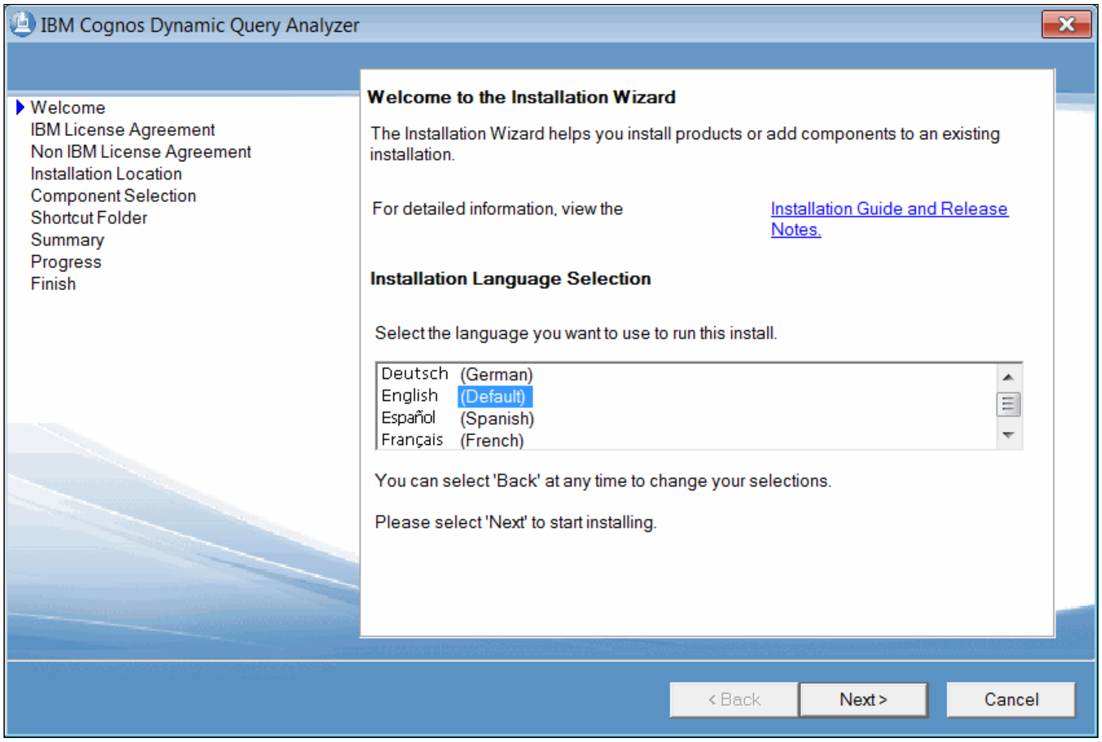


Figure 3-16 Language selection

Notice on this window that you can click the **Installation Guide and Release Notes** link to view the manual.

4. The IBM License Agreement window opens (Figure 3-17). Read the terms of the license agreement. When you fully understand and agree with these terms, select **I Agree** and click **Next**.

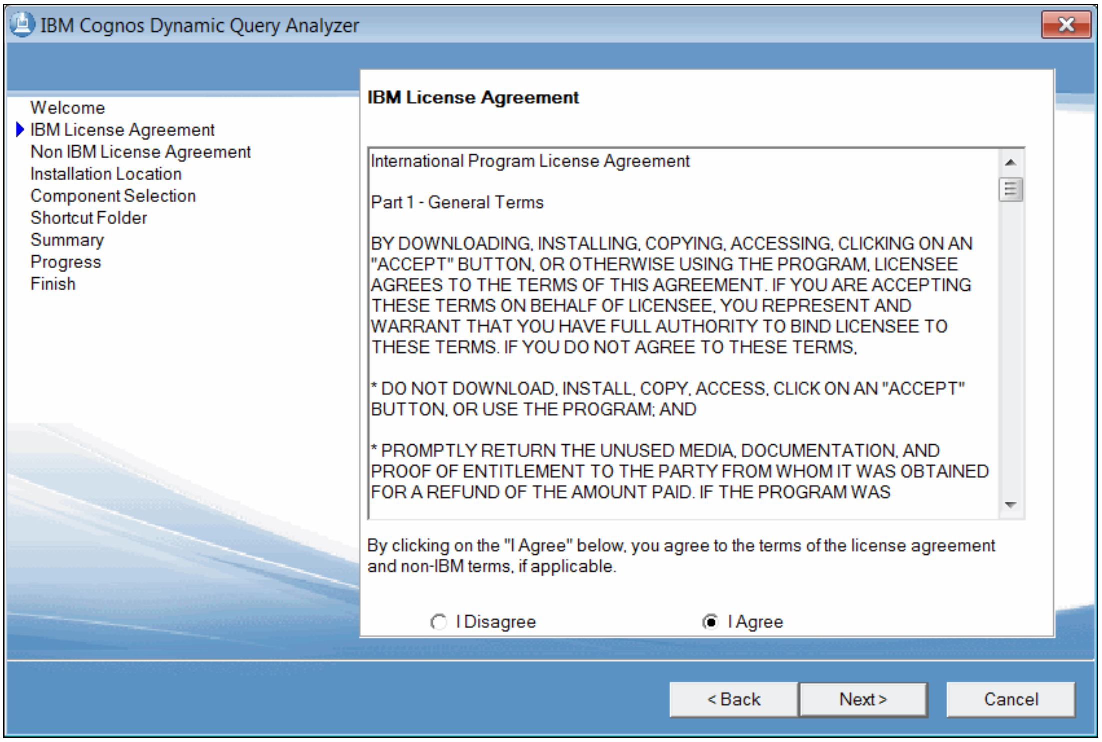


Figure 3-17 License agreement

The first window of the installation wizard opens that shows the installation location (Figure 3-18).

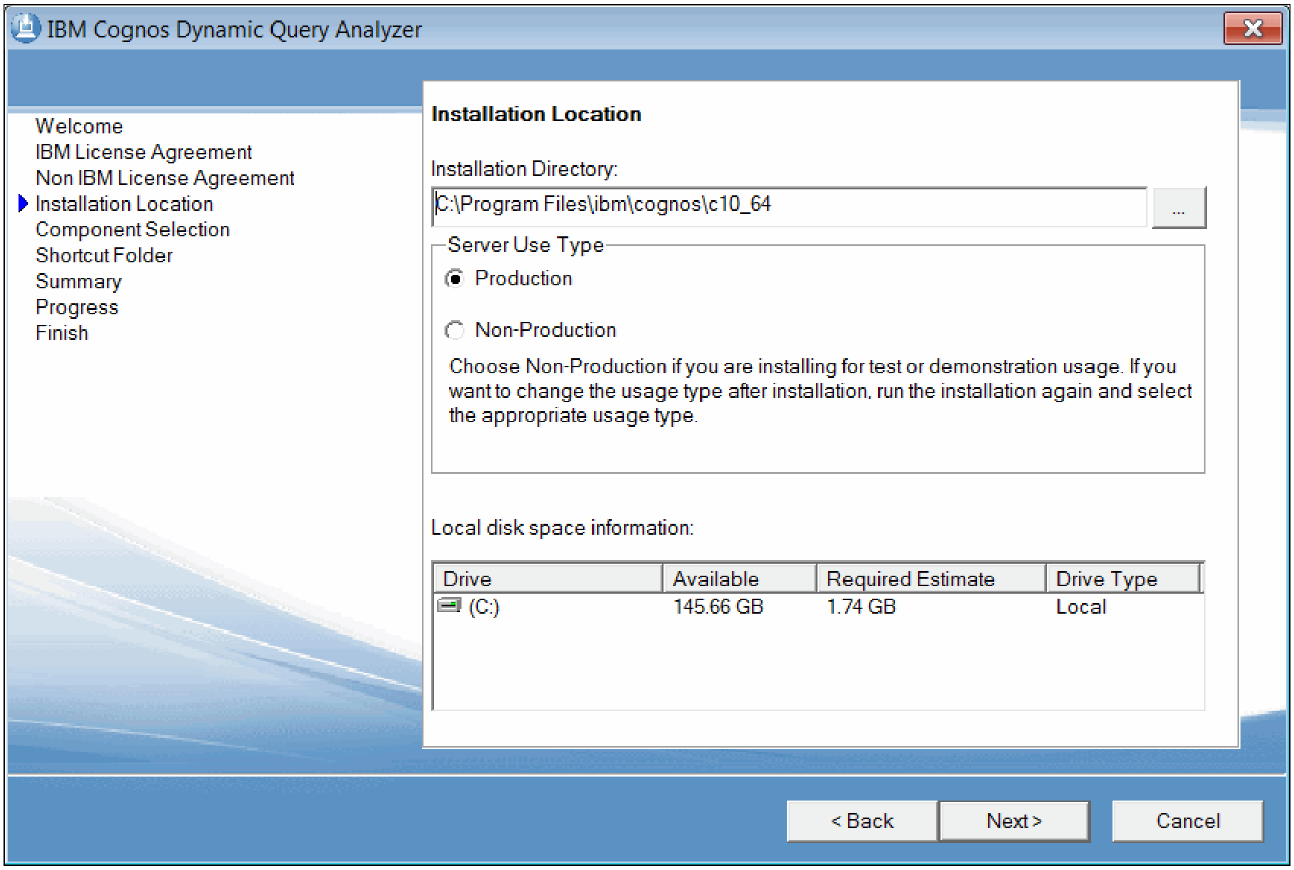


Figure 3-18 Installation location

In this window, you can change the installation. By default, when you install the 64-bit version, the installation directory is `C:\Program Files\ibm\cognos\c10_64`. A different default location might be necessary for some business requirements. For example, some organizations allow only the operating system to be on the `C` drive, whereas other applications, such as Dynamic Query Analyzer, must be installed on a secondary drive. Use this window to change the default location.

After you select the correct location for the installation, click **Next**.

5. The Component Selection window opens (Figure 3-19). IBM Cognos Dynamic Query Analyzer is a single component. This option is selected by default. Click **Next**.

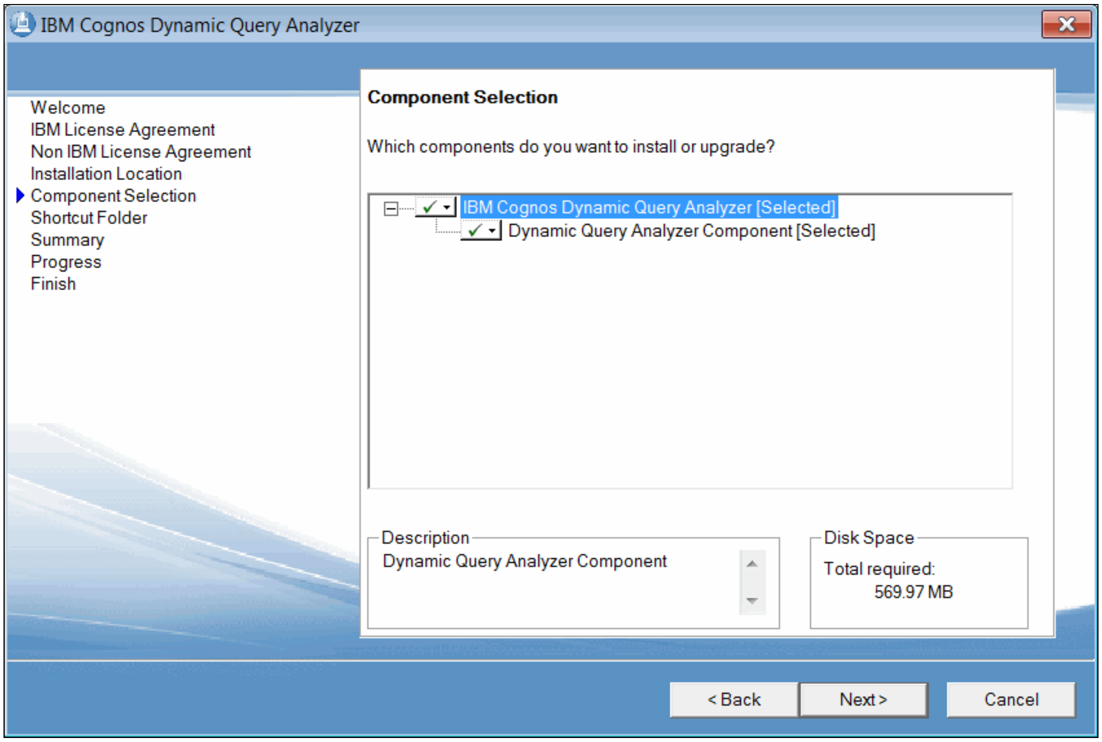


Figure 3-19 Component selection

6. The Shortcut Folder window opens (Figure 3-20). In this window, you can select the Start Menu folder where you want to install the Dynamic Query Analyzer application. By default, the Dynamic Query Analyzer shortcut is placed on the IBM Cognos 10 folder.

   In this window, you can make available the Dynamic Query Analyzer shortcut to all users for that workstation, or you can limit the shortcut to only the user account that you are using to install the application. Security is the main reason why some organizations might want to limit the shortcut to only the installation user, especially when multiple users have access to the same workstation.

   When you finish making your selections, click **Next**.

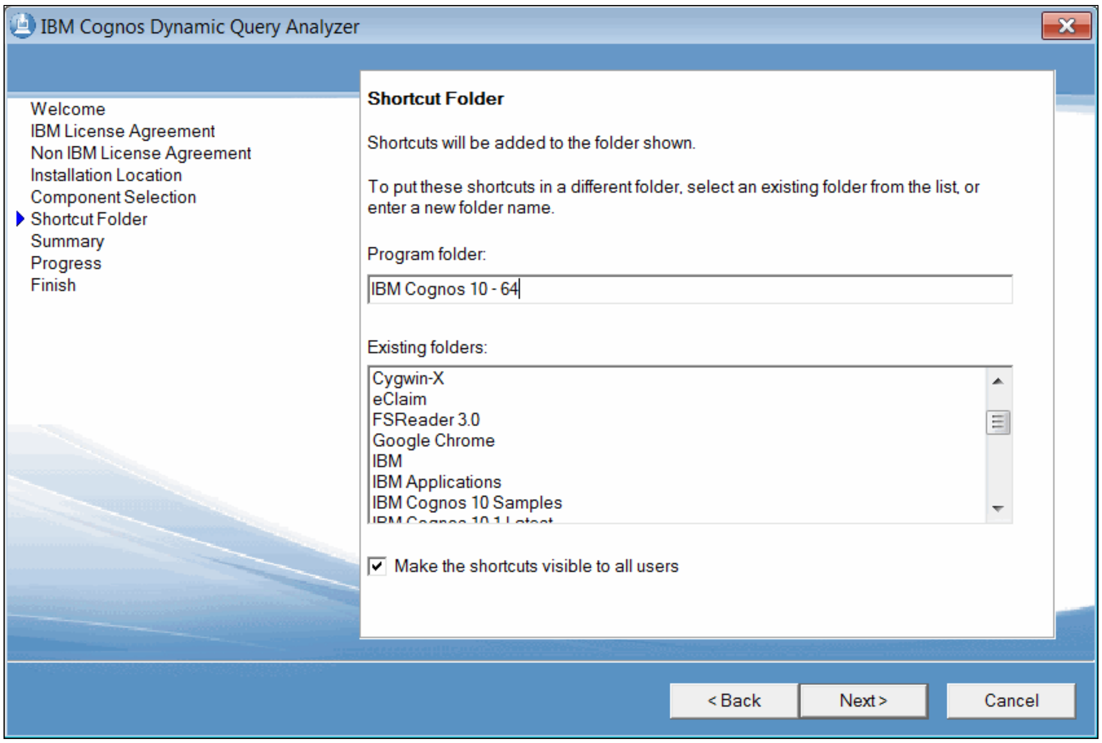


Figure 3-20 Shortcut folder

7. The Installation Summary window opens (Figure 3-21). This window lists all the options that you selected during the installation. Carefully review all the settings. If you want to change a setting, click **Back** to go through all the options of the Installation Wizard. Make any necessary changes and return to the Installation Summary window. When you are satisfied with your selections, click **Next**.

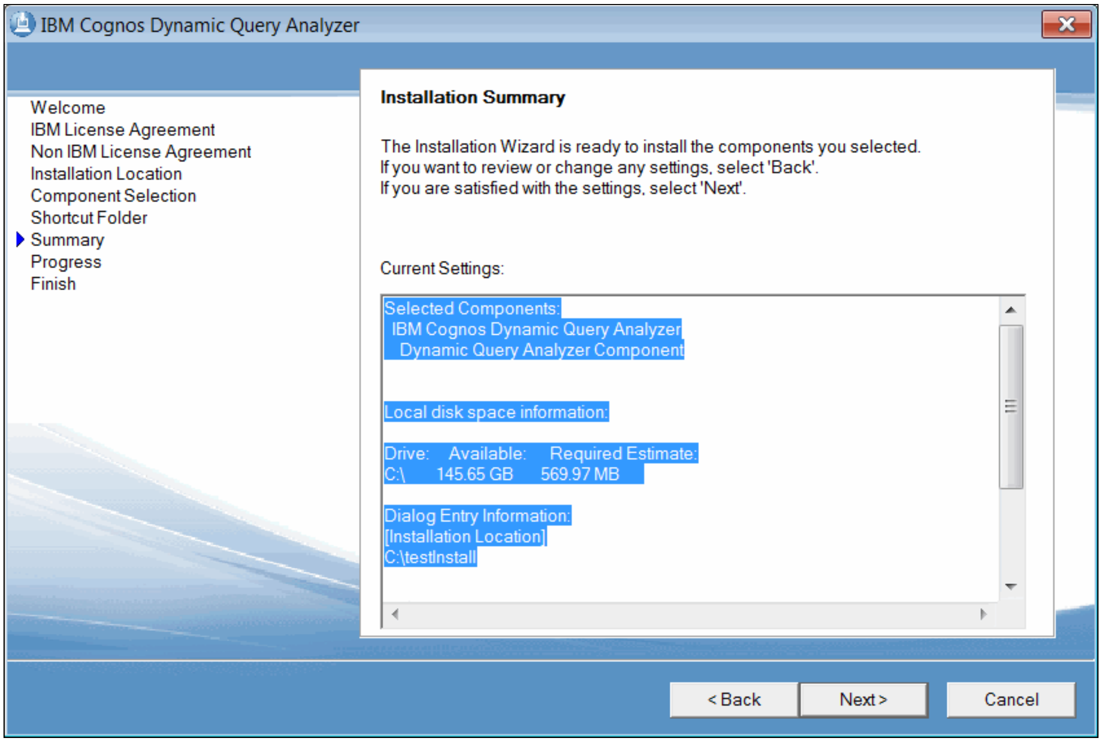


Figure 3-21 Installation summary

The installation wizard starts installing the Dynamic Query Analyzer on the workstation. A progress window opens (Figure 3-22).

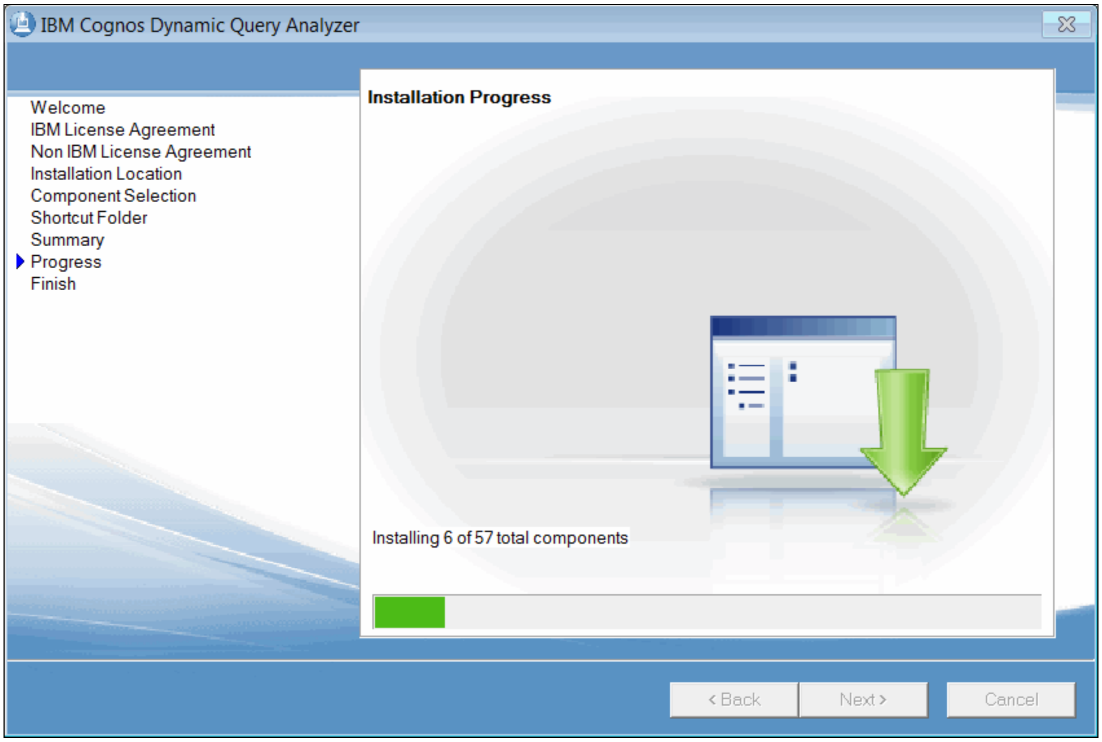


Figure 3-22 Installation progress

8. When the installation is complete, a Finish window opens (Figure 3-23). You can view the transfer log and summary-error log. Click **Finish** to complete the installation.

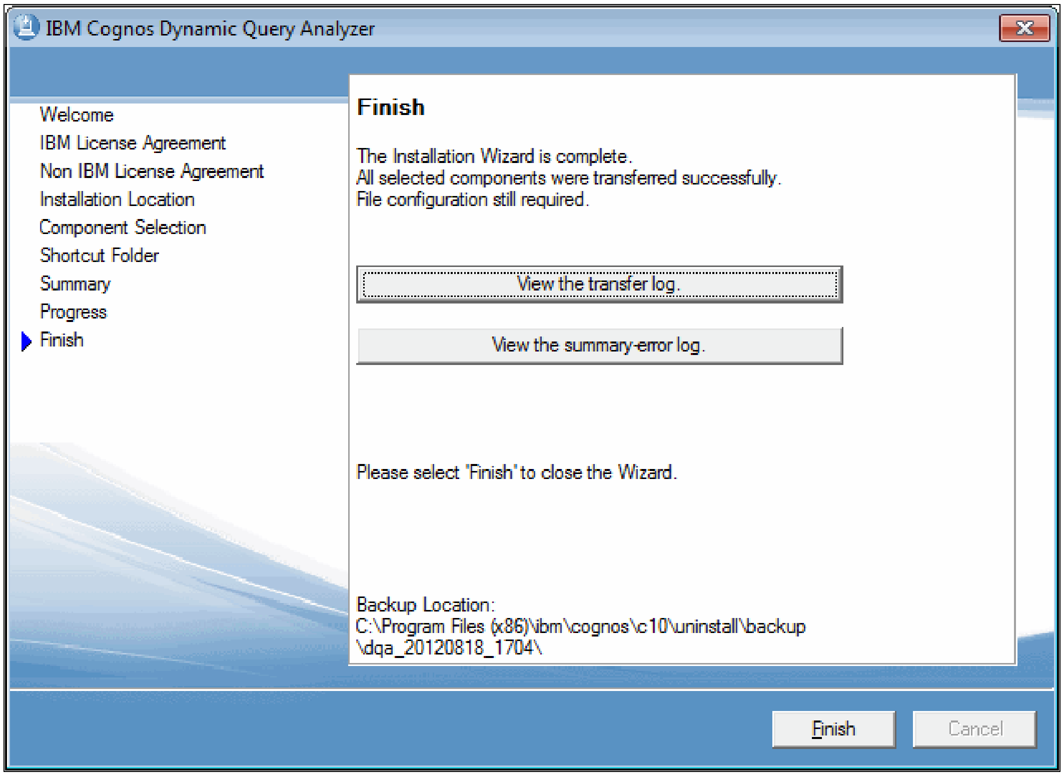


Figure 3-23 Finish window

3.5.3 Configuring Dynamic Query Analyzer

To be able to use Dynamic Query Analyzer, you must properly configure it to connect to the IBM Cognos 10.2.2 Business Intelligence Server. Use the following procedure to configure Dynamic Query Analyzer.:

1. Start Dynamic Query Analyzer. Click **Start** → **Programs** → **IBM Cognos 10**, and then click **IBM Cognos Dynamic Query Analyzer**.
2. After Dynamic Query Analyzer is loaded, select **Window** → **Preferences**, as shown in Figure 3-24.

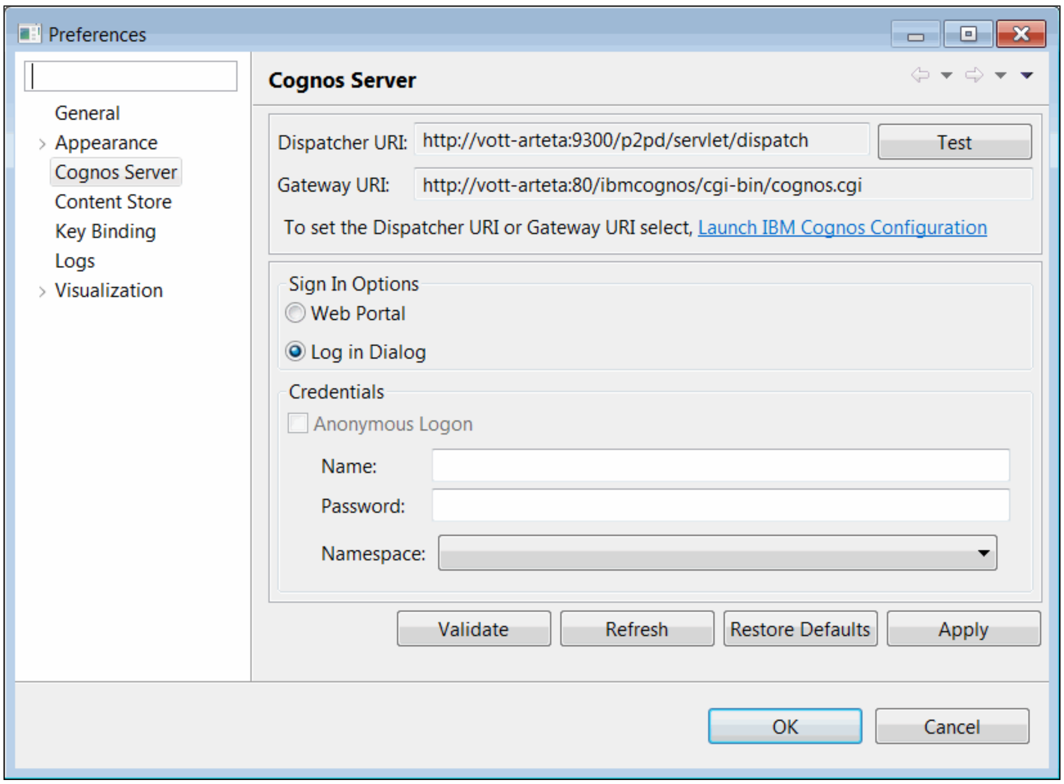


Figure 3-24 Preferences

3. Select **Cognos Server** from the navigation pane (Figure 3-25).

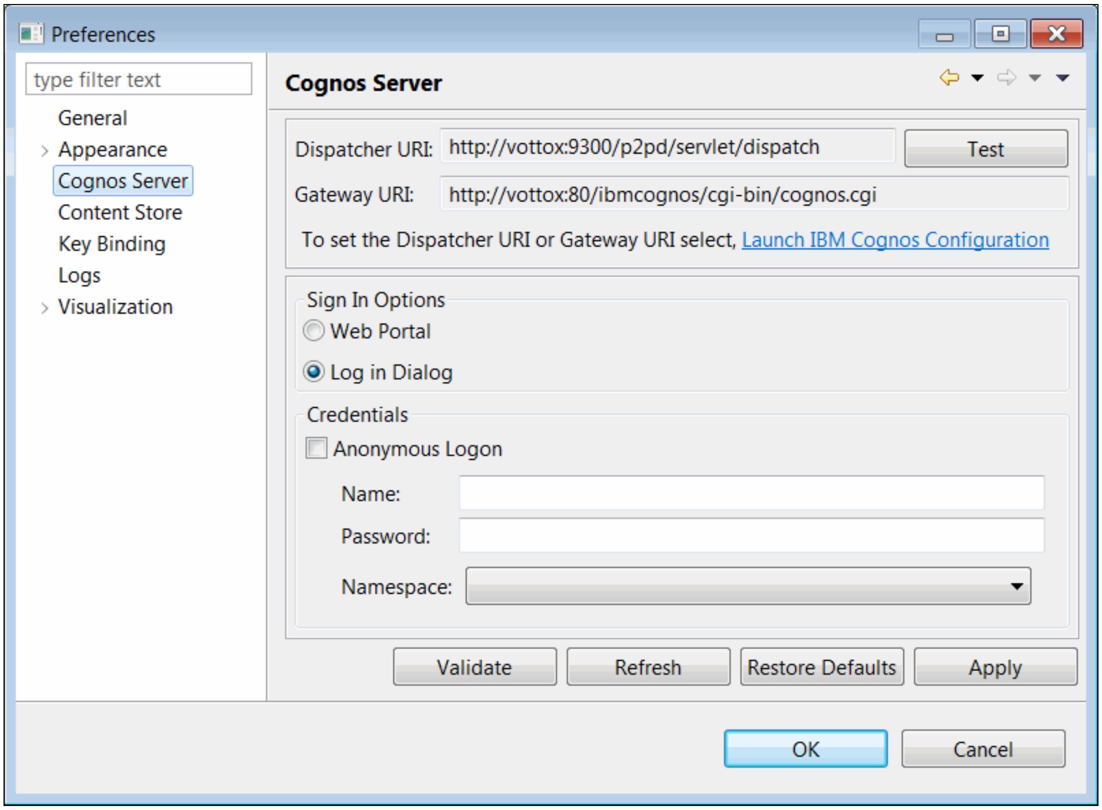


Figure 3-25 Cognos server preferences

Update the following information:

– **Dispatcher URI** and **Gateway URI**

By default, the **Dispatcher URI** and **Gateway URI** are set to `Localhost`. If the IBM Cognos 10.2.2 Server is on a different computer, then update both settings. If you do not know the location of the IBM Cognos 10.2.2 Server, contact your Cognos administrator.

– **Name**, **Password**, and **Namespace** of the user

If security is implemented on the IBM Cognos 10.2.2 Business Intelligence Server, then type your user name, password, and select the name space. If you do not know your user name, password, or name space, contact your Cognos administrator.

4. If you want to use Dynamic Query Analyzer to analyze query log files, select **Logs** in the navigation pane (Figure 3-26).

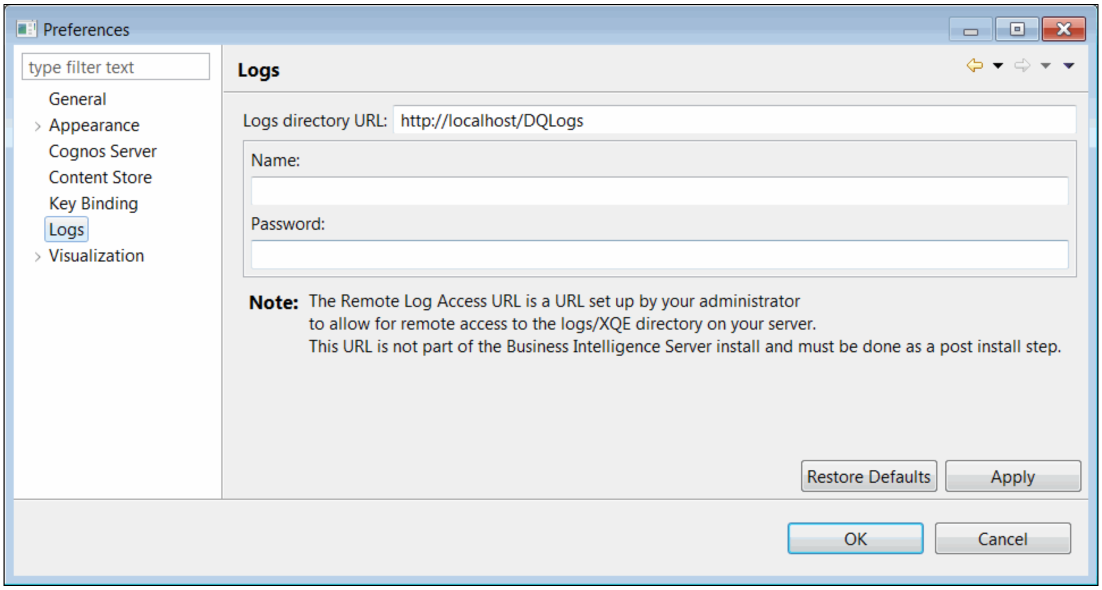


Figure 3-26 Dynamic Query Analyzer logs preferences

Update the following information:

– **Virtual directory path**

In the Logs directory URL field, enter the virtual directory where IBM Cognos 10.2.2 Business Intelligence Server is saving the query log files. Note that this URL is not created by default, and so it needs to be set up by your Cognos administrator. It should be a file:, https:, or http: URL that points to the `logs/XQE` directory of your installation.

– **Name** and **Password** of the user

If security was implemented on the virtual directory, enter your user name and password. If you do not know your user name or password, contact your Cognos Administrator.

4

Modeling with IBM Cognos Cube Designer

This chapter introduces IBM Cognos Cube Designer, the modeling and design tool for dynamic cubes. This chapter describes the major components of Cognos Cube Designer and how to navigate the user interface. Chapter 5, “Basic modeling” on page 105 introduces you to the basics of building a dynamic cube.

This chapter contains the following sections:

- Introduction to Cognos Cube Designer
- Starting Cognos Cube Designer
- Creating a project
- The Cognos Cube Designer workspace
- Using the Get Metadata menu
- Exploring metadata and data

4.1 Introduction to Cognos Cube Designer

Cognos Cube Designer is the modeling environment for designing and deploying dynamic cubes. Cognos Cube Designer contains a number of features to streamline the modeling process and help users ensure that they design and deploy a well-designed and efficient cube. Modelers can explore the structure of their data and examine the relationships between the tables in their data warehouse to determine the objects to be used in their cube model.

This chapter focuses on exploring the components of the Cognos Cube Designer user interface. For more information about using Cognos Cube Designer to model and deploy dynamic cubes, see Chapter 5, “Basic modeling” on page 105 and Chapter 6, “Advanced topics in modeling” on page 159.

4.2 Starting Cognos Cube Designer

You can start Cognos Cube Designer from the Windows **Start** menu or from IBM Cognos Framework Manager by selecting the **Run IBM Cognos Cube Designer** option from the **Tools** menu. Starting with IBM Cognos 10.2.2, it is no longer necessary for Cognos Framework Manager to be installed to use Cognos Cube Designer. Figure 4-1 shows the Getting Started window that is displayed when Cognos Cube Designer is first started. You have the option of creating a new project or opening an existing project from the file system.

Figure 4-1 Getting Started page of Cognos Cube Designer

4.3 Creating a project

To create a project, start Cognos Cube Designer and either select **Create New** from **Metadata** on the Getting Started window, or **Create New Blank Project**. Creating a project from metadata opens the Select a database schema window, which displays the IBM Cognos Business Intelligence (BI) data sources that you have access to. All configured data sources from your Cognos BI server environment to which you have access are displayed. From here, you can select the schema that you want to use and progress to the metadata import step.

If you choose to create a project without importing metadata, the Cognos Cube Designer workspace appears and you can either model your cube or import metadata or a published Cognos Framework Manager model.

4.4 The Cognos Cube Designer workspace

The Cognos Cube Designer workspace is organized into a menu bar and four main working panes. Figure 4-2 shows the workspace of Cognos Cube Designer with the four working panes highlighted: Metadata, Project Explorer, Editor, and Details. You can expand or collapse each area to free space in the window for the area you are currently using.

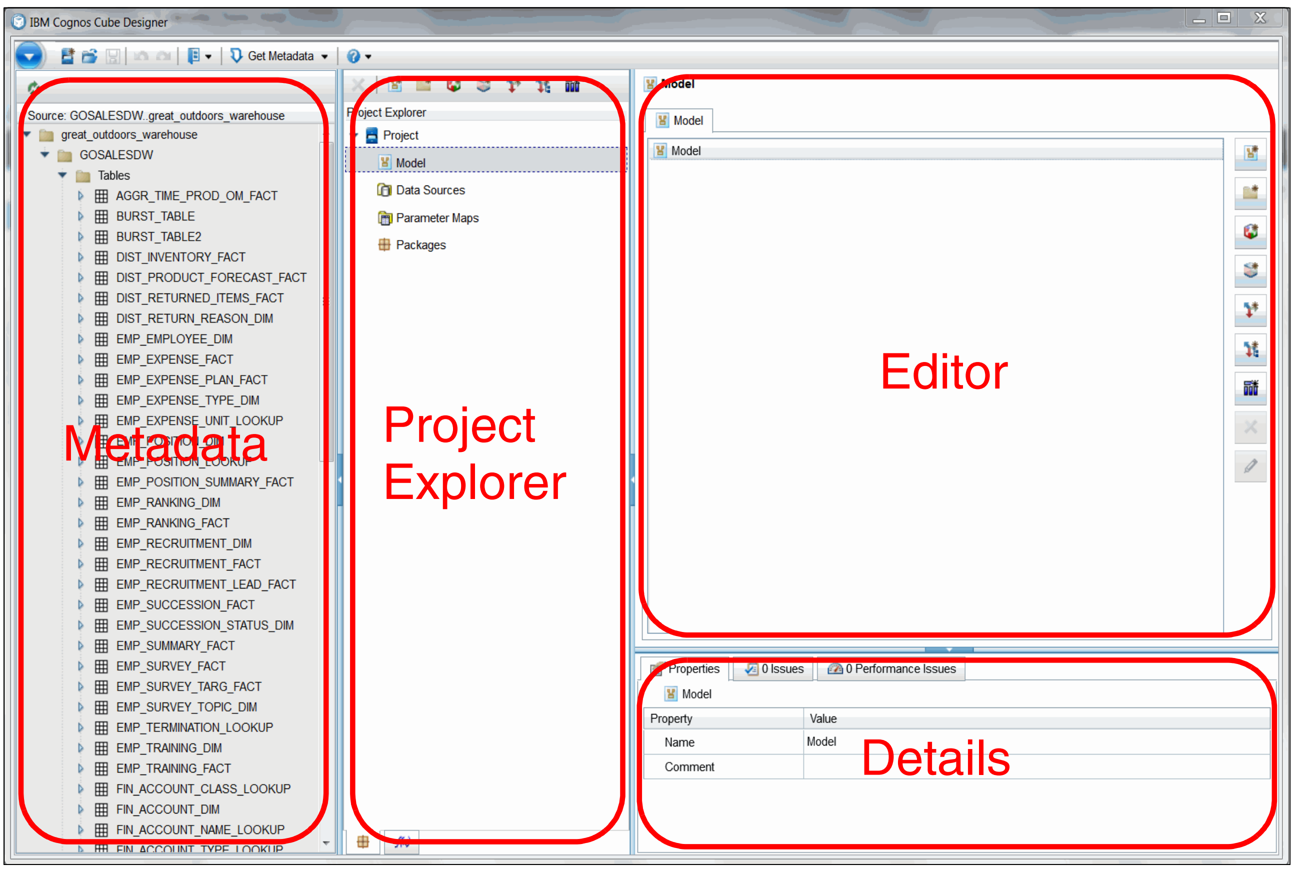


Figure 4-2 Workspace of Cognos Cube Designer with the four main working panes highlighted

4.4.1 The Cognos Cube Designer menu bar

The Cognos Cube Designer menu bar contains buttons and menu items for working with your model, configuring Cognos Cube Designer, and interacting with Cognos BI.

The first icon on the left is the **File** menu. The next three icons are **New**, **Open**, and **Save**. The **Undo** and **Redo** icons follow for undoing and redoing changes to your model. The **View** icon allows you to configure visual aspects of Cognos Cube Designer. The **Get Metadata** menu allows you to access and work with data sources or Cognos Framework Manager packages. There is more information about this menu in 4.5, “Using the Get Metadata menu” on page 100. Finally, the **Help** menu provides access to help with Cognos Cube Designer, allows you return to the Getting Started window, and shows the product About window.

The File menu

The **File** menu contains several commands for working with your model or interacting with Cognos BI. Figure 4-3 shows the Cognos Cube Designer **File** menu.

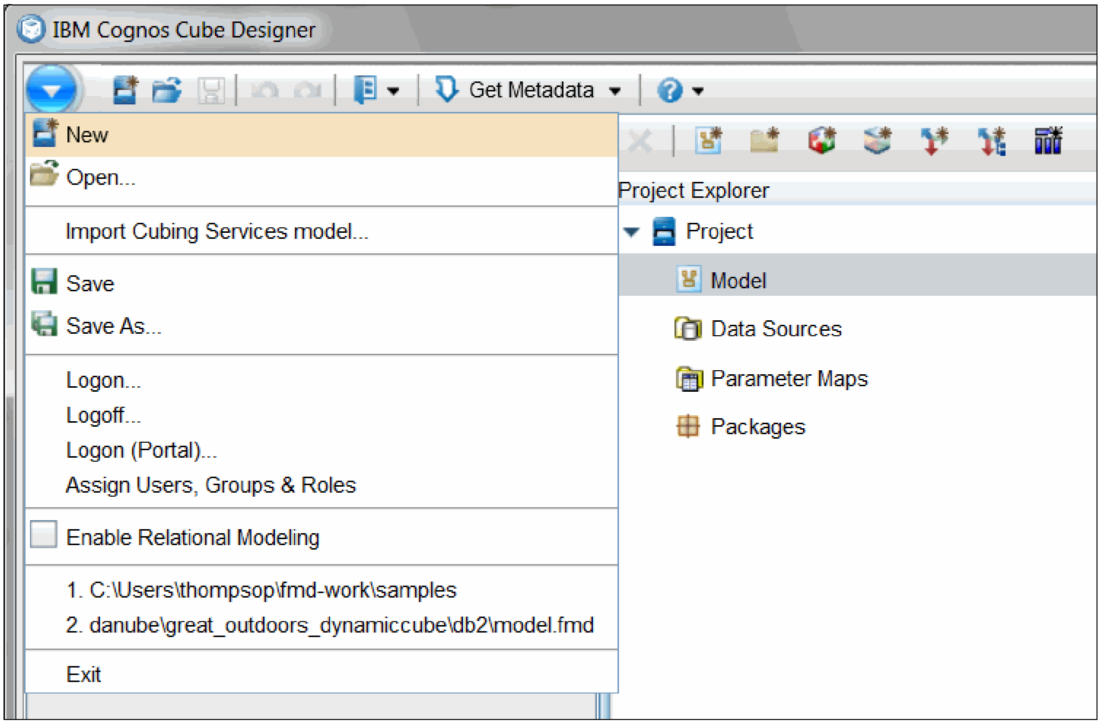


Figure 4-3 The Cognos Cube Designer File menu

The **File** menu provides these options:

- **New**, **Open**, **Save**, and **Save As** are used for creating a blank model, opening an existing model, and saving your model.
- **Import Cubing Services model** is used for importing a model from an IBM Cubing Services application.
- **Logon**, **Logoff**, and **Logon (Portal)** are used for authenticating with a Cognos BI server.
- **Assign Users, Groups & Roles** and **Enable Relational Modeling** enable you to create simple relational modeling in Cognos Cube Designer. They are also used to specify the user, group, and role access for published relational models from Cognos Cube Designer. Creating relational models is not discussed in this book, but you can learn more by reading *Relational and DMR modeling in Cognos Cube Designer* at:

  http://www.ibm.com/support/knowledgecenter/SSEP7J_10.2.2/com.ibm.swg.ba.cognos.ug_cog_rlp.10.2.2.doc/c_cog_rlp_model_dmr_rel_data.html
- The most recently used file list contains a list of your most recently used models, allowing easy access to models you use frequently.

The View menu

The **View** menu allows you to configure visual aspects of Cognos Cube Designer. Figure 4-4 shows the **View** menu in its default state.

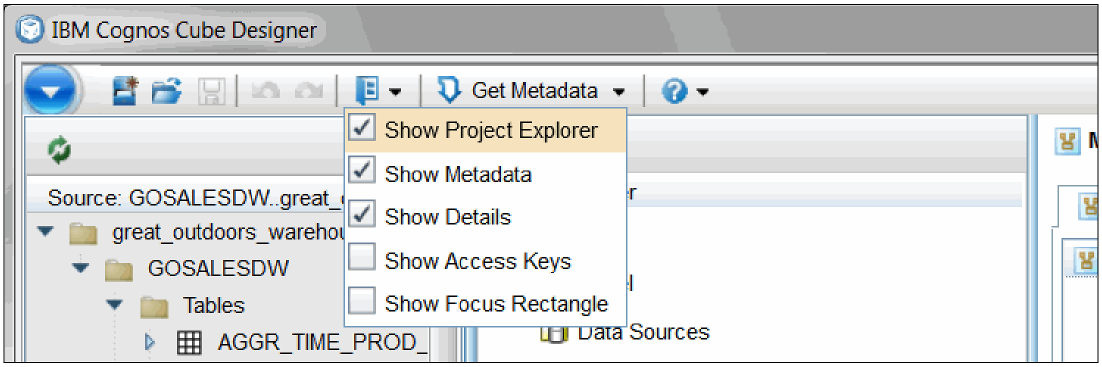


Figure 4-4 The View menu in its default state

The **View** menu provides these options:

- **Show Project Explorer** allows you to easily show or hide the Project Explorer.
- **Show Metadata** allows you to easily show or hide the metadata.
- **Show Details** allows you to easily show or hide the Details window.
- **Show Access Keys** and **Show Focus Rectangle** are used to show and hide accessibility aids in Cognos Cube Designer.

The Get Metadata menu

The **Get Metadata** menu is used for accessing metadata so that you can work with it in Cognos Cube Designer. Section 4.5, “Using the Get Metadata menu” on page 100 explores this subject in more detail.

The Help menu

Figure 4-5 shows the **Help** menu of Cognos Cube Designer.

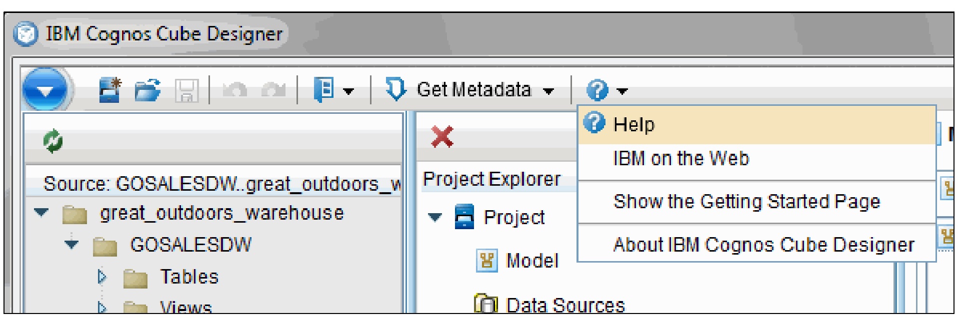


Figure 4-5 The Help menu of Cognos Cube Designer

Use the **Help** menu to access online help for Cognos Cube Designer, access the IBM website, return to the Getting Started window of Cognos Cube Designer, and show the About box.

4.4.2 Metadata

The Metadata window is where you can view the metadata that you will use to build your dynamic cubes. Metadata is an accordion control: Each pane shows a single data source or Framework Manager package. See 4.5, "Using the Get Metadata menu" on page 100 for details about selecting metadata sources. If you have multiple metadata sources, each metadata source has its own pane in the accordion control. You can switch between data sources by clicking the header bar for each pane. By clicking a closed pane, the current pane that is open collapses so that the details for only a single metadata source are shown at a time.

> **Tip:** Cognos Cube Designer uses interface elements in the shape of a triangle called twisties that users click to collapse or expand sections or categories. A twistie to the left of the text of the object indicates that an object has child objects. A white twistie that is pointing to the right indicates that the object has child objects, but that they are not currently shown. A twistie that is black and points downwards indicates that the object has been expanded so that its immediate children are shown. Click the twistie to show or hide the children of an object.

Within a data source pane, tables, views, and synonyms are organized into separate folders in the tree named Tables, Views, and Synonyms, as shown in Figure 4-6. If there are many tables, views, or synonyms, they are further organized into subfolders with names indicating the first and last database objects in the subfolder. Cognos BI recognizes most data sources, so a specialized database object likely shows up in the Tables folder if they are valid for dynamic cubes. For example, Materialized Query Tables (MQT) objects (a feature of IBM DB2®) appear in the Tables folder.

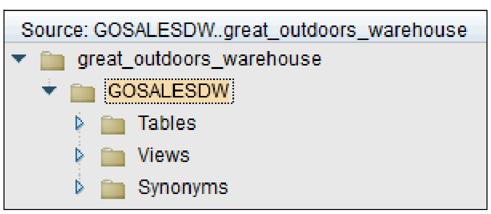


Figure 4-6 The Tables, Views, and Synonyms folders within a data source

Figure 4-7 shows some of the metadata for the `great_outdoors_warehouse` sample database that is included with Cognos BI. Some objects such as the `great_outdoors_warehouse`, GOSALESDW, and the Tables folder are expanded, showing their children. Other objects such as AGGR_TIME_PROD_OM_FACT, BURTS_TABLE, and BURST_TABLE2 are collapsed, but show that they have children.

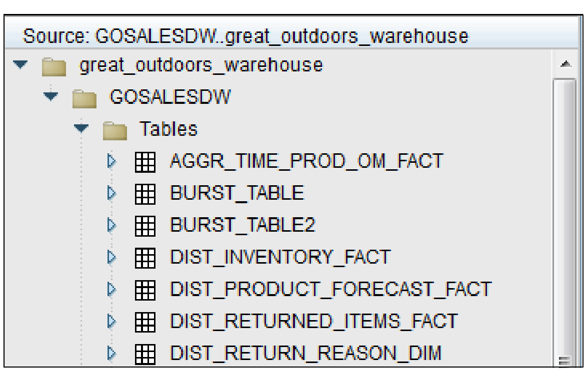


Figure 4-7 The Metadata window with the great_outdoors_warehouse metadata

If you select any Framework Manager model to import modeling objects from, the model also appears as a pane in the Metadata window, organized in the same way the Framework Manager model is organized.

4.4.3 The Project Explorer

The *Project Explorer* is the central place to manipulate a model. It is composed of a toolbar and a tree of objects (objects exist as children of other objects). The toolbar contains interface elements that are context-sensitive, that is, you only see icons for actions that are appropriate for the object you selected in the tree. All objects in the tree have a menu that can be accessed by right-clicking the object. The menu of an object contains the commands that are associated with all of the toolbar elements and any other action you are able to perform on that object.

The organization of objects in the tree is similar to the organization of objects you see in the metadata tree in the IBM Cognos reporting studios. When you create a project as described in 4.3, "Creating a project" on page 90, Cognos Cube Designer creates a default project with a number of objects already in it:

- Model is the root namespace for the project. All of the dimensions and cubes that you create exist under this namespace. Optionally, you can also create other name spaces and folders to organize your model.
- Data Sources contains the list of data sources that are used by your model. You never directly create a data source object. Rather, they are created for you as you map model objects to columns in a metadata source. However, you might need to edit the properties of a data source depending on your IBM Cognos environment.
- Parameter Maps are a way to create sets of key-value pairs that you can then reference in other model objects.
- Packages are the means of presenting and deploying cubes to your IBM Cognos environment, so that report authors and users can use them.

Chapter 5, "Basic modeling" on page 105 includes more information about working with these objects.

The Project Explorer has another aspect to it. Figure 4-8 shows the two tabs in the Project Explorer.

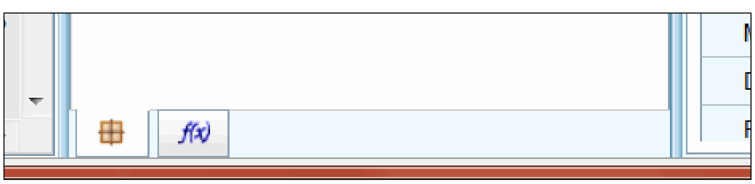


Figure 4-8 Project Explorer tab and the Functions tab

The first tab is for the Project Explorer itself. Selecting the second tab hides the Project Explorer and provides access to a reference library of functions that can be used when you are creating or editing calculations, along with syntax help (the same help that appears in IBM Cognos Report Studio). To return to the Project Explorer, click the first tab again.

4.4.4 The Editor window

The Editor window is where the main object edit windows are located. Double-clicking an object in the Project Explorer shows the editor for that object, if there is one. If there is no editor, you see a message stating that there is no editor. In general, editors are used for specifying complex properties of an object and managing child objects of that object (some of which can also be managed in the Project Explorer, and some cannot). Figure 4-9 shows the editor for a hierarchy object.

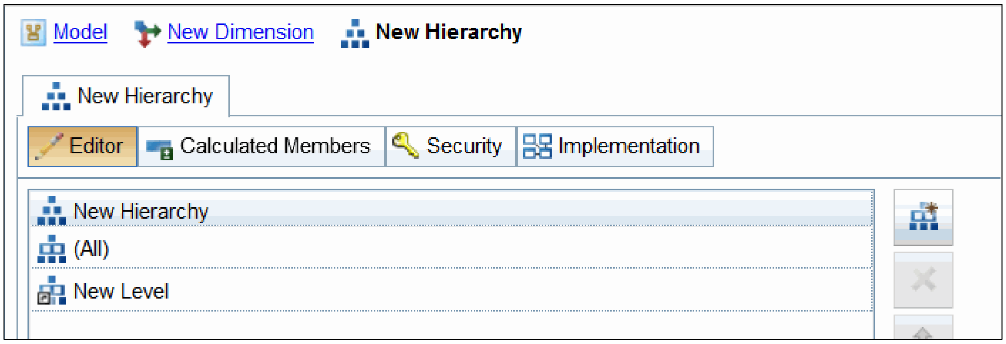


Figure 4-9   Editor for a hierarchy object

Figure 4-10 shows the breadcrumb links at the top of the Editor window.

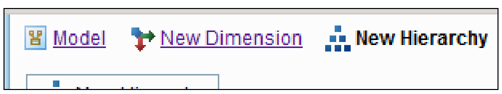


Figure 4-10   Breadcrumb links for the New Hierarchy editor

The breadcrumb shows you where the object you are editing is in relation to its parent and ancestor objects up to the root namespace of the model. When you are editing objects in the editor, you might end up switching to the editor for a child object. The breadcrumb allows you to quickly return to the editor for a parent object. Clicking any of the ancestor labels in the breadcrumb links starts the editor for that object.

Figure 4-11 shows the editor label, with the name and icon of the object the editor is for.

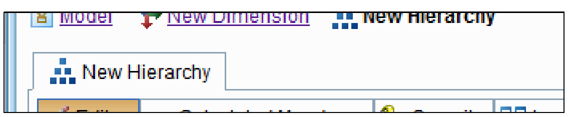


Figure 4-11   Label for the New Hierarchy editor

Usually, when you start the editor for an object, it replaces the current editor. However, there are situations where you want to have more than one editor open at a time. You can prevent Cognos Cube Designer from automatically closing an editor by pinning it. Right-click the tab for the editor and click **Pin**. Figure 4-12 shows two Editor windows. The New Hierarchy editor is pinned, and the New Level editor shows the Pin menu.

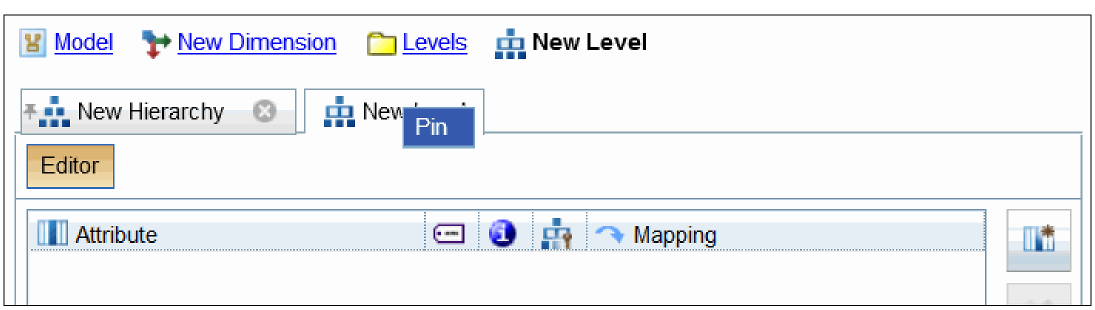


Figure 4-12   New Hierarchy editor pinned and the Pin menu

A pinned editor has an icon on the left of the label that looks like a pin. Clicking the **Pin** icon unpins the editor (if you double-click another object to start another editor, the unpinned editor is replaced). To close the editor, click **X** on the right. Pinning an editor can be useful if you want to compare the properties of two objects at once, or if you want to reference the data in a table, as described in 4.6.1, "Viewing data from a table" on page 104.

4.4.5 The Details window

The Details window shows details for the most recently selected object in either the Project Explorer or the Editor. Figure 4-13 shows the Details window with details for the New Dimension object.

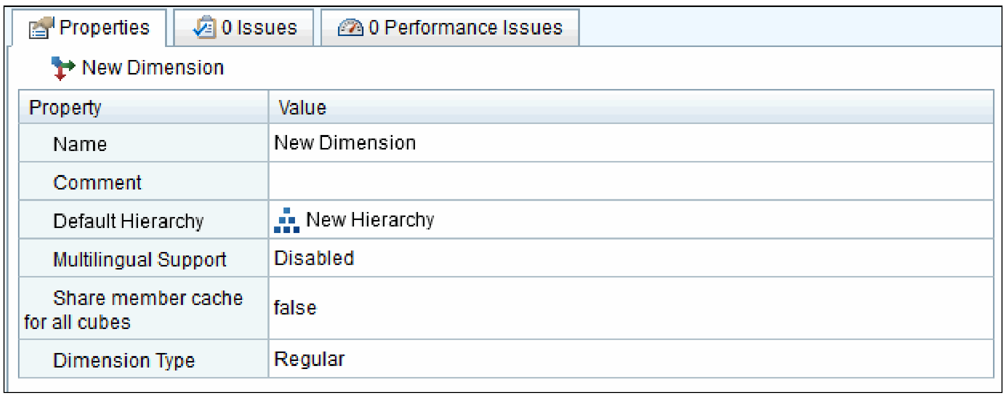


Figure 4-13 Details window for the New Dimension object

The Details window contains three tabs with specific types of information for the selected object: Properties, Issues, and Performance Issues tabs.

- The Properties tab contains simple properties for the item that is selected in either the Project Explorer or Metadata. Many of these properties are editable. As a general guideline, simple properties for an object are in the Properties tab of the object while more complex properties and child objects are in the editor of the object.
- The Issues tab contains a list of any unresolved issues with the object currently selected in the Project Explorer. These issues must be resolved before the object you are working on is valid. You cannot deploy a cube to IBM Cognos Connection if there are outstanding errors to resolve.
- The Performance Issues tab contains a list of suggestions that might improve the performance of your dynamic cube. You can safely ignore these suggestions if they are unsuitable for your application.

4.4.6 The expression editor

Many objects that you will work with when building your application will include an expression such as attributes, measures, data filters, or security filters. To successfully create and edit expressions, you must have a good knowledge of the different dimensional and relational objects that you can model in Cognos Cube Designer. These model objects are covered in greater detail in Chapter 5, "Basic modeling" on page 105. If you are new to modeling with cubes, you might find it easier to jump ahead to 5.1, "Modeling Concepts" on page 106 and then come back to this section.

When you start the expression editor by double-clicking the Expression property of a model object, you see a window similar to Figure 4-14.

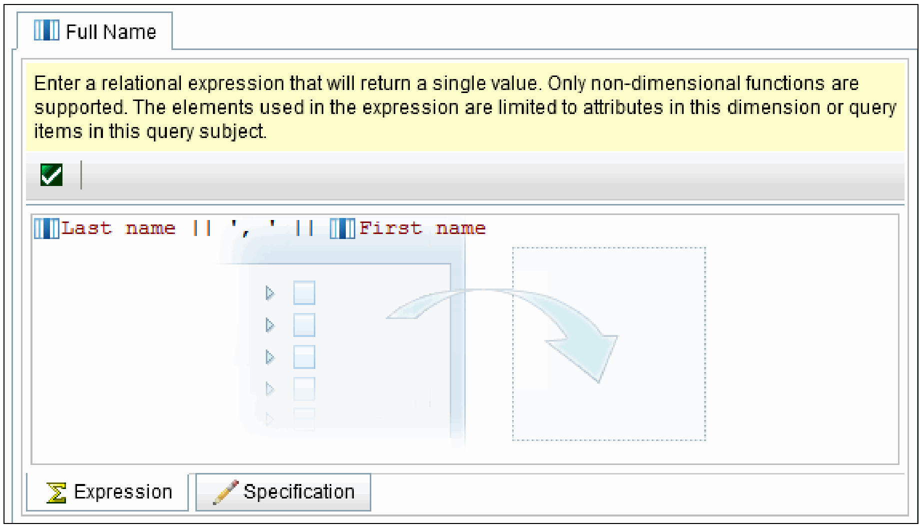


Figure 4-14 Expression editor

The text in the yellow box at the top of the expression editor describes what kind of expression should be used. The requirements of the expression and what it should resolve depends on the context where the expression is used.

There are two basic kinds of expressions you can create with Cognos Cube Designer:

- Relational expressions that use relational database constructs and are ultimately pushed to the underlying database. Model objects that use relational expressions include attributes, measures, data filters, and lookup-table-based security filters. Attributes and measures must resolve to a string or numeric value. Filters must resolve to a Boolean and test column values against specified constants or other values. A relational expression can include references to other attributes or measures.
- Dimensional expressions that are resolved by the dynamic cubes. Model objects that use dimensional expressions include calculated members, calculated measures, named sets, and security filters. Calculated members and calculated measures must resolve to a value. A named set expression must resolve to a set of members. You can refer to dimensions, hierarchies, levels, attributes, and other calculated members and measures in dimensional expressions.

The green check box is for validating your expression. Note that it is only simple syntax validation. To fully test your expressions, you need to deploy your cube and test it in the studios.

The expression editor has two tabs, each with a slightly different purpose. The Expression tab allows you to type in expressions and even paste text from text editors, and you can drag-and-drop other model objects from the Project Explorer into the expression. However, you cannot copy-and-paste model object references in the Expression tab. Figure 4-15 shows the Specification tab of the expression editor.

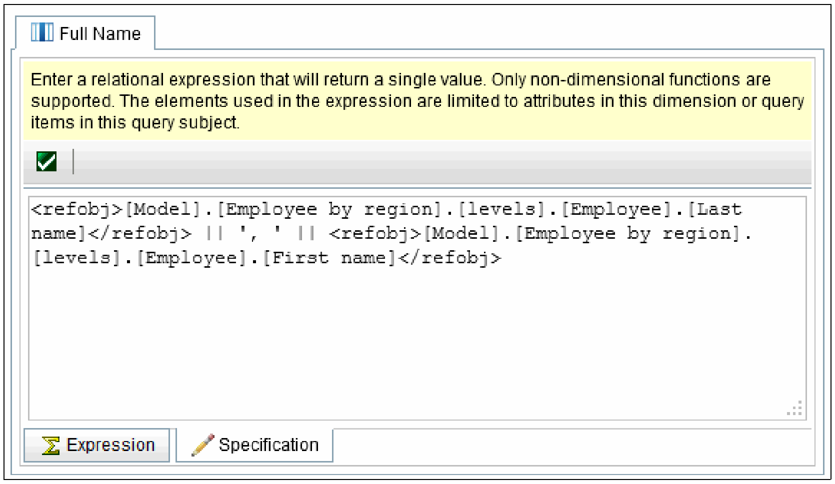


Figure 4-15 The Specification tab

In the Specification window, you cannot drag-and-drop model objects into the tab. However, you can manually edit model object references. It is important to be conscious of the structure of model object references when you are editing. The identifier itself is composed of one or more object names in the *Design Locale* surrounded by square-brackets. See 5.2.1, "Managing locales" on page 110 for details about setting the design locale. The names are generally a chain of object names all the way up to a namespace or cube. The names of cubes and name spaces must be unique within the model. The object names are separated by the period character. Finally, the entire reference must be surrounded by `<refobj>` and `</refobj>` tags (these are XML tags: The internal format of a model is XML). For example, a valid reference to the `Last name` attribute in an expression looks like this:

```
<refobj>[Model].[Employee by region].[levels].[Employee].[Last name]</refobj>
```

In this example, the `Last name` attribute is part of the `Employee` levels. Levels are contained in the special `levels` folder within a dimension. The dimension in this case is `Employee by region`, which is contained in the `Model` namespace.

Adding references to attributes and levels in an expression depends on what kind of expression you are editing. If you are editing a relational expression, such as an attribute with an expression, you must add attributes from the level under the `Levels` folder of the dimension. Levels cannot be added to relational expressions because they are not relational objects. If you are editing a dimensional expression, such as a calculated member, then you must add attributes and levels by using their shortcuts under a hierarchy.

4.5 Using the Get Metadata menu

There are two sources of metadata that you can use to model dynamic cubes. You can build them directly using tables and columns from a data source in the IBM Cognos environment, or you can select objects from a published Framework Manager package and import them. In either case, you start the process by choosing the appropriate item from the Get Metadata menu, as shown in Figure 4-16.

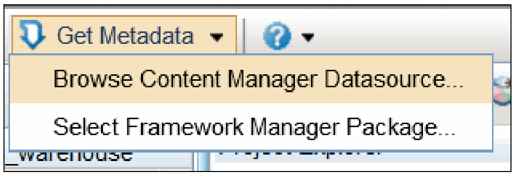


Figure 4-16 Get Metadata menu

It is important to note that, by bringing metadata into Cognos Cube Designer, you do not import anything into your model or changed your model in any way. You are simply making that metadata available for your use while modeling.

In a secured Cognos environment, if you are already logged in to IBM Cognos, you are prompted to log in the first time you attempt to use Get Metadata.

4.5.1 Using Get Metadata with a data source

When you use Get Metadata with a data source, the list of data sources you see are those configured for dynamic query mode (DQM) in the IBM Cognos Business Intelligence (BI) server environment. The list of data sources you see is further filtered to show only those DQM data sources that your administrator has granted you access to. If you do not see a data source that you believe you should have access to, contact your IBM Cognos system administrator.

Generally, you should import the metadata from your reporting data warehouse before you begin building the dimensions and cubes that will form your application. Although this is the suggested workflow, you can build the outline for your cubes, dimensions, and measure dimensions, and then fetch the metadata and perform a mapping exercise to associate the level attributes with columns in your tables and views.

After you select **Browse Content Manager Datasource** from the **Get Metadata** menu, Cognos Cube Designer shows the *Select a database schema* window. It contains a tree of the databases, catalogs, and schemas you have access to, including system objects, as shown in Figure 4-17.

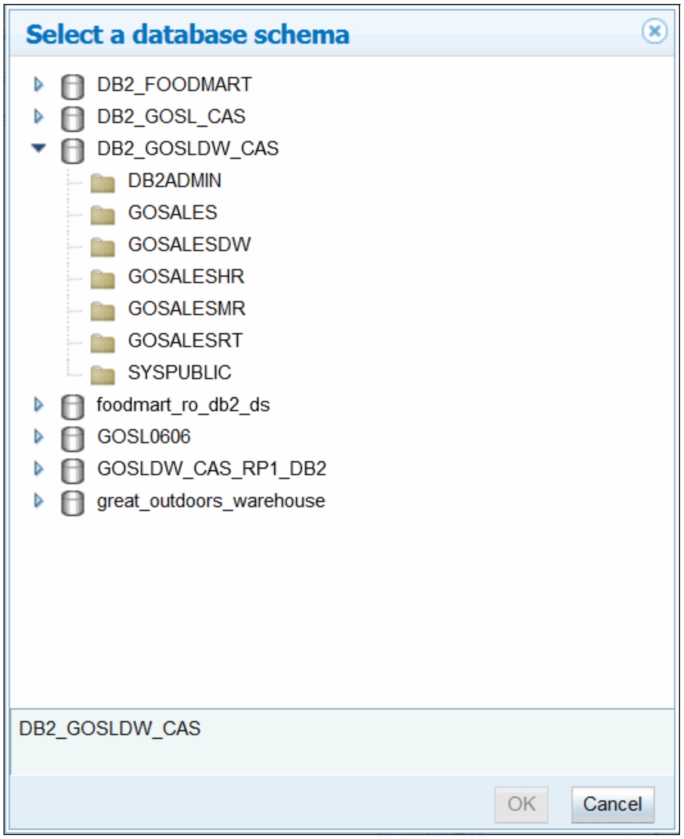


Figure 4-17 Select a database schema window with several data sources

You can only fetch the metadata for a single schema at a time. You can import more than one schema at a time if you need to by repeatedly starting Select a database schema and selecting each schema that you need to fetch in turn. The database schema that you selected appears in a new pane in the Metadata window.

4.5.2 Using Get Metadata against Framework Manager

Cognos Cube Designer allows you to reuse your Framework Manager packages by fetching the model and then allowing you to import portions of the model to use as dimensions in a dynamic cube. You can use a model built for either DQM or compatible query mode (CQM). However, if the data sources defined in the model use CQM, you must update the data sources in Cognos Cube Designer to refer to DQM data sources.

The process of using Get Metadata for a Framework Manager package is similar to using Get Metadata for a data source. After you have fetched a model, you can import individual parts of the model as dimensions for use in a dynamic cube. You can use any Framework Manager package, whether it was built using DQM or CQM. Click **Select Framework Manager Package** from the **Get Metadata** menu to open the Select a Package window to select a Framework Manager model to import.

Figure 4-18 shows the Select a Package window with a package and a folder.

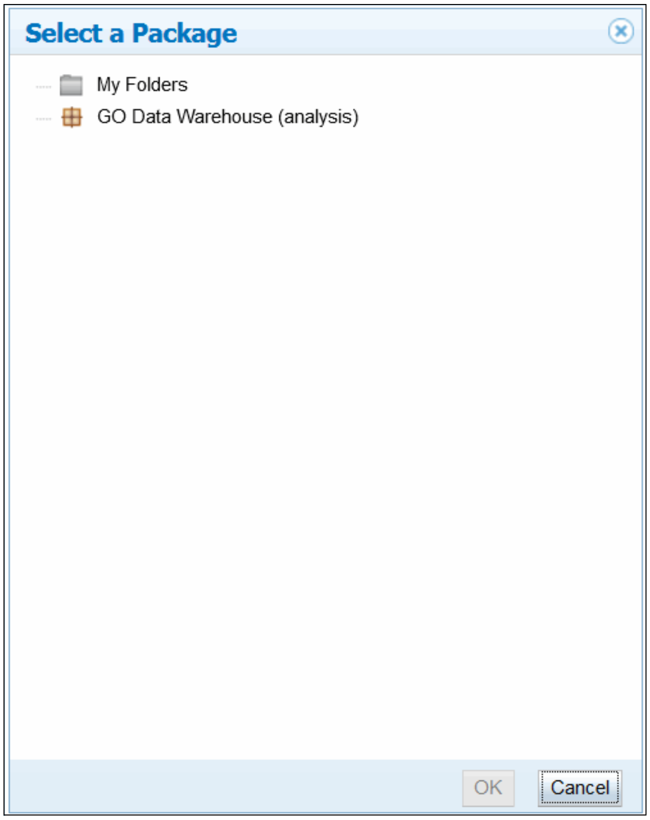


Figure 4-18 Select a Package window

Be aware that the Select a Package window lists all packages that you have access to, whether they are a fully modeled package created from Framework Manager or a light-weight model for a third-party online analytical processing (OLAP) data source. Therefore, you should be familiar with the package you want to see. A package for a light-weight model is displayed as an empty pane in the Metadata.

The process for importing dimensions from a Framework Manager model is covered in further detail in Chapter 16, "Using Framework Manager models" on page 519.

4.6 Exploring metadata and data

During the process of building your application, you might want to explore the data source you are using, to either see what tables are related with primary-foreign key relationships, or how a dimension table joins to the fact table. Alternatively, you might want to see the contents of your data source in a more flexible view than the standard tree view in the Data Source Explorer.

Cognos Cube Designer features an explorer diagram that you can use to see the tables and views in your data source, and show how the tables relate to one another. To open the diagram, right-click a table in the Data Source Explorer control and select **Explore Metadata**. The diagram opens and adds the selected table to the default view.

After the explorer diagram is open, you can drag more tables onto the diagram from the Data Source Explorer control. Any primary-foreign key relationships that exist are automatically visible. You can also choose to show any related tables to those already visible on the diagram. The **Show Joined Tables** buttons on the toolbar automatically add these tables to the pane. Buttons show all related tables or tables that are joined in a `1..n` or `n..1` relationship to the selected table.

Use the slider control on the toolbar to control the level of detail visible on the canvas. You can choose to see a low level of detail, such as the table names, or higher levels of detail, such as the individual columns and keys for the object in the pane. Figure 4-19 shows a Relational Explorer Diagram for the SLS_SALES_FACT table and all of the tables joined to it.

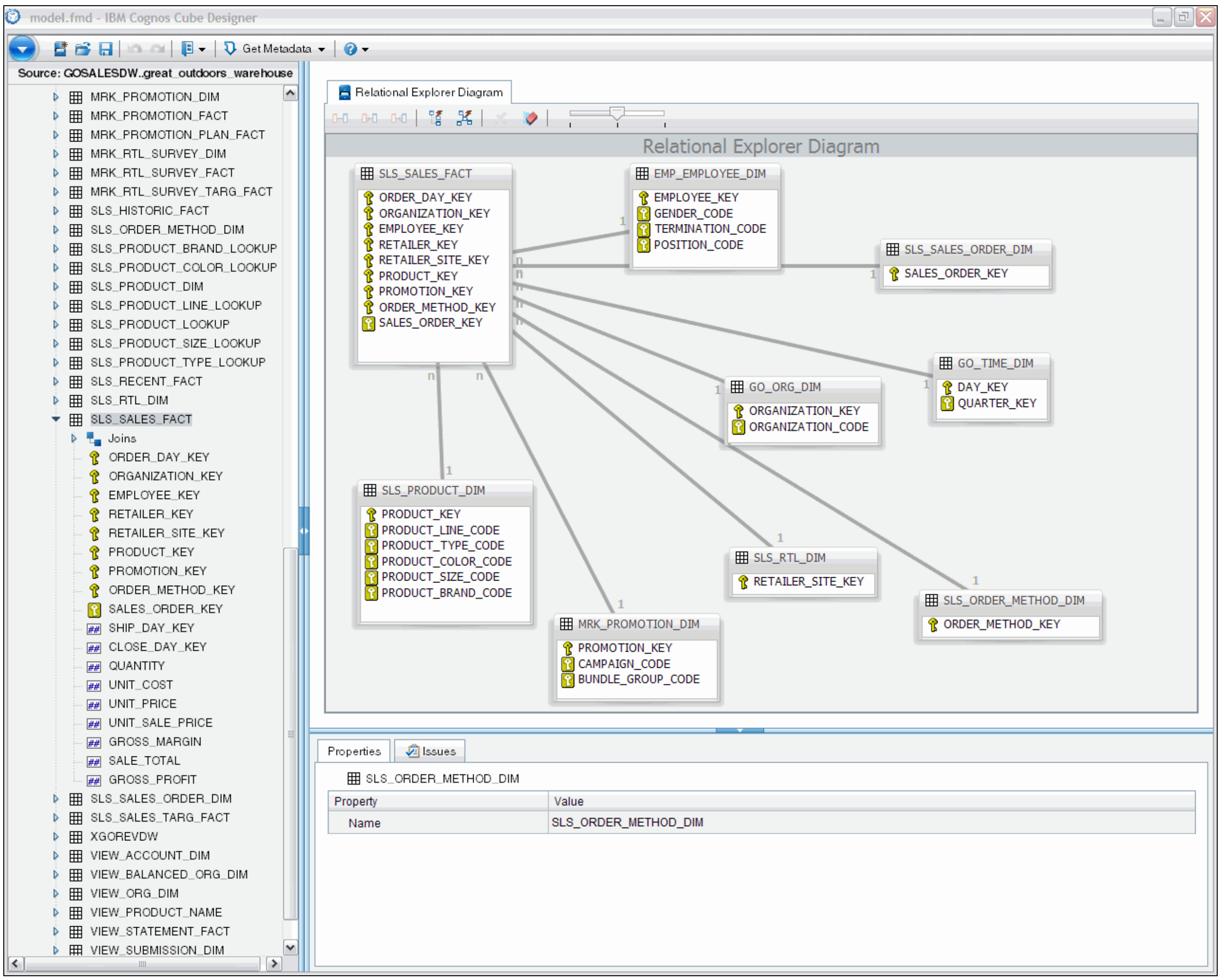


Figure 4-19 Relational Explorer Diagram showing the SLS_SALES_FACT table and all of the tables joined to it

> **Diagram is read-only:** The diagram is read-only. The purpose of this diagram is to enable users to examine the objects in the data source and how they relate to one another. You cannot edit any of the objects in the diagram or create objects. These operations must be done in either the Project Explorer or an Editor.

4.6.1 Viewing data from a table

You can view the data in a table. Right-click the table of interest and select **View Data**. Figure 4-20 shows the menu for the SLS_SALES_FACT table from the `great_outdoors_warehouse` sample database with **View Data** highlighted.

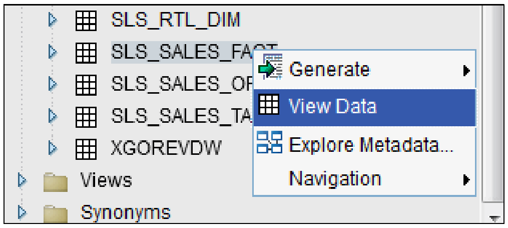
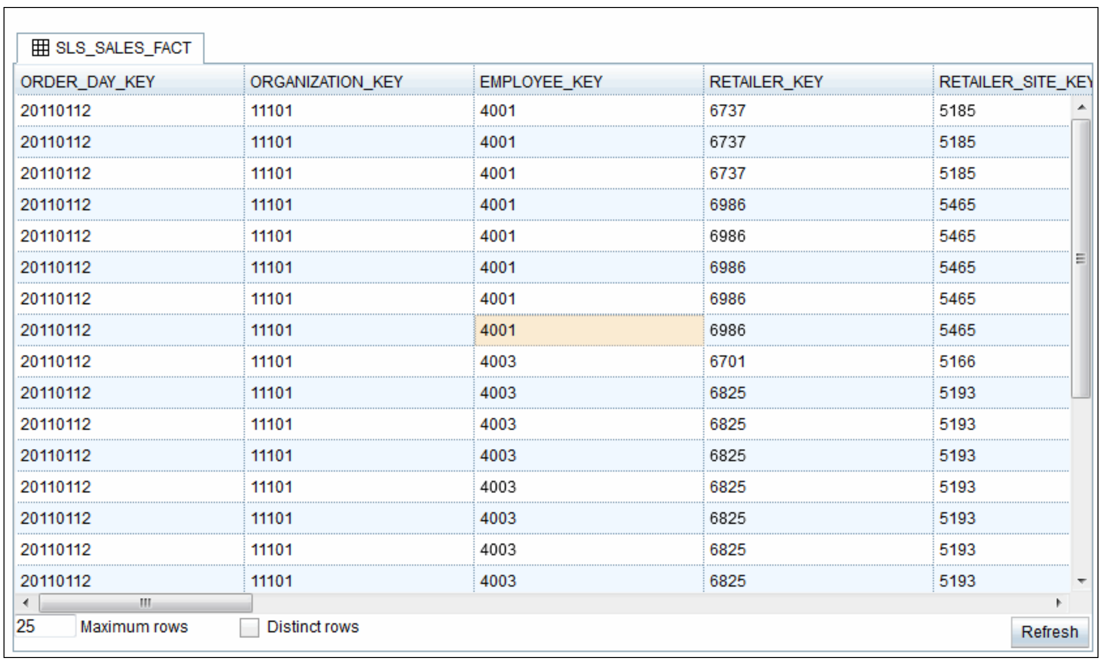


Figure 4-20 Menu for SLS_SALES_FACT with View Data highlighted

For many modeling operations, such as modeling level keys and identifying hierarchies in dimensions, knowing the data that you are working with is imperative. The data is shown in a grid in the Editor window. By default, the first 25 rows of data for all columns in the table are retrieved. Figure 4-21 shows the data for the SLS_SALES_FACT table.



SLS_SALES_FACT

ORDER_DAY_KEY	ORGANIZATION_KEY	EMPLOYEE_KEY	RETAILER_KEY	RETAILER_SITE_KEY
20110112	11101	4001	6737	5185
20110112	11101	4001	6737	5185
20110112	11101	4001	6737	5185
20110112	11101	4001	6986	5465
20110112	11101	4001	6986	5465
20110112	11101	4001	6986	5465
20110112	11101	4001	6986	5465
20110112	11101	4001	6986	5465
20110112	11101	4003	6701	5166
20110112	11101	4003	6825	5193
20110112	11101	4003	6825	5193
20110112	11101	4003	6825	5193
20110112	11101	4003	6825	5193
20110112	11101	4003	6825	5193
20110112	11101	4003	6825	5193
20110112	11101	4003	6825	5193

25 Maximum rows   Distinct rows   Refresh

Figure 4-21 Data in the SLS_SALES_FACT table.

You can pin the data grid the same way that you pin an editor. You can change the number of rows to retrieve by editing the **Maximum Rows** value and clicking **Refresh**. Note that the performance of this action depends on the performance of your database.

5

Basic modeling

This chapter describes how to use the IBM Cognos Cube Designer to design and deploy dynamic cubes, how to model cube dimensions and measures, and how to validate and analyze the models that you create.

Before you can successfully model and deploy IBM Cognos Dynamic Cubes, it is important to understand some basic concepts and terminology. Many of these concepts are standard within the business intelligence industry and are not exclusive to dynamic cubes.

The examples in this chapter use the `gosldw_sales` data source that is included in the IBM Cognos BI 10.2.2 samples. A model is used to illustrate points and to allow you to get experience with Cognos Cube Designer. This model is similar to the sample Cognos Cube Designer model that is included with the IBM Cognos BI samples. If you want to follow along with the activities in this chapter, you can use the Cognos Cube Designer sample model.

For information about purpose, content, location, installation, and setting up the IBM Cognos Business Intelligence samples, see the IBM Knowledge Center at:

http://www-01.ibm.com/support/knowledgecenter/SSEP7J_10.2.2/com.ibm.swg.ba.cognos.inst_cr_winux.10.2.2.doc/c_instsamplesoverview.html

For information about setting up the samples for IBM Cognos Dynamic Cubes, see IBM Knowledge Center at:

http://www-01.ibm.com/support/knowledgecenter/SSEP7J_10.2.2/com.ibm.swg.ba.cognos.ig_rolap.10.2.2.doc/C_ig_rolap_sample_setup.html

This chapter contains the following sections:

- Modeling Concepts
- Starting a dynamic cubes project
- Modeling dimensions
- Modeling measures
- Bringing dimensions and measures together in a cube
- Parameter maps
- Validation
- Deploying dynamic cubes for reporting and analysis
- Managing data sources

5.1 Modeling Concepts

Before you begin creating a dynamic cube, you must understand certain fundamental concepts. If you worked with other multidimensional technologies such as IBM Cognos PowerCubes, you can safely skip this section and start on 5.2, "Starting a dynamic cubes project" on page 110. Otherwise, you familiarize yourself with the terminology and technology described in this section.

5.1.1 Cubes

A cube is a data store designed to present data to users in an easy to understand manner. It contains a structured view of the entities within an organization, along with metrics, also known as measures, facts, and key performance indicators (KPIs). The structure can contain objects such as a Gregorian or retail calendar, a company's employee organization, or the structure of a company's product lines. The structured aspect allows various approaches to be applied to optimize the data within a cube to allow for interactive analysis and reporting of very large volumes of data.

You can think of a cube as an actual cube, with each axis of the cube representing one of these units such as time, the employee organization chart, or the product organization. These axes are called *dimensions*. Each coordinate along an axis represents an important individual item in the context of that axis, for example a single day in a dimension for time or a single product in a dimension for products. These items are called *members*, also known as *categories* in IBM Cognos Transformer and IBM Cognos PowerPlay. A collection of coordinates, one from each axis, is called a *tuple*. For each tuple, there are one or more specific values for the metrics that the consumers of the cube care about, called *measures*. Typically, cubes have many more dimensions than the three implied by the cube metaphor. The measures of a cube are also organized within a dimension, called the *measure dimension*, and a tuple includes a single measure from the measure dimension. In other words, measures are the members of the measure dimension. Each cube has only one measure dimension.

IBM Cognos Dynamic Cubes are entirely in memory and query an underlying database for measure values on an as-needed basis, enabling the cube to provide fast access to the most import data with few delays. Your job as the modeler of a cube is, in part, to specify which database tables the cube should query for dimensional and measure data. To be successful, the database tables must be structured in a certain way. A single, central table contains the values for your measures (called a *fact table*), with joins to one or more other tables that contain the data for your dimensions, called *dimension tables*. This arrangement is often referred to as a *star schema*. All of your measures must be defined from a single fact table, but your dimension can be defined using a single table or multiple tables. The later arrangement is often called a *snowflake schema*.

Calculations that are written with the intention of being calculated by the cube itself are referred to as *dimensional* because they act on structures within the cube. Conversely, calculations that are intended to be run by queries to the underlying database are referred to as *relational* because they act on structures within a relational database such as tables.

The concepts of cubes, dimensions, and measures are ultimately just tools to present information to your data consumers in ways that make sense to the organization. The specific cubes, dimensions, and measures that you create should be determined by the needs of the users.

5.1.2 Dimensions

A dimension is a collection of members. A dimension can contain one or more hierarchies, each of which imposes a specific hierarchical structure to the members of the dimension. A hierarchy, in turn, is constructed of one or more levels, each containing a collection of similar members or entities. Members from one level are associated to members at lower and higher levels by parent-child and child-parent relationships.

A hierarchy can, in it simplest form, be an ordered container of all the items in the dimension that is described as a wide, flat dimension. When presented pictorially, all the members are laid out on a single level in which the only relationship is the relationship of the members to their prior and next sibling within the level. Wide dimensions are not ideal from an optimization point of view for several OLAP products, including IBM Cognos Dynamic Cubes.

Typically, however, levels are organized within a hierarchy from most general (at the top) to most detailed (at the bottom). For example, your company might sell many products. Those products might be organized by product types, which are further organized by product line. Your dimension for products has members for each product, product type, and product line, all organized into a hierarchy of product information.

However, it might not make sense for users to organize members into levels. Instead, it is simply an arrangement of members along parent-to-child and child-to-parent relationships. A dimension with such an arrangement of members is called a parent-child dimension. For example, in the reporting structure of a typical corporation, people might not easily fit into strict levels for reporting purposes, but are instead related only by to whom they report.

A dimension can contain one or more hierarchies if there are multiple ways of organizing or looking at the members of the dimension. For example, if you have a dimension for time, it is common to organize the members according to the unit of time, with levels for years, quarters, months, and days. However, it might be important when analyzing retail sales values to look at the data according to both a Gregorian calendar and a retail calendar, or one organized according to various promotions throughout the year.

Within Cognos Dynamic Cubes, a parent-child dimension only supports a single hierarchy.

Depending on the needs of the users, you might have to provide more than one hierarchy in your time dimension, one representing the calendar year and one representing the fiscal year.

Time is an important dimension and one that is involved in nearly all business analysis and reporting. Cognos Dynamic Cubes, as detailed later in this chapter and the next, provides special features related to the time dimension.

5.1.3 Levels

A level is a collection of members that belong together within a hierarchy. When you define a level, the most important thing is to define which members belong to it. You specify those members by specifying a *level unique key* or *level key,* that is, the criteria being a member within the level. In Cognos Dynamic Cubes, you typically use the values from one or more columns in a dimension table, where a member is created for each unique value.

5.1.4 Hierarchies

A hierarchy is a structure that organizes and relates the members in a dimension.

For example, a time dimension hierarchy can consist of a year level, a month level, and a day level. The highest level of the hierarchy, year, has the most abstract information classifying time. If you were examining something from the level of years, all the data would be aggregated at that level. Months are more detailed and days even more so. A product dimension hierarchy can consist of a product line level, product type level, and a product level.

It is possible to have multiple hierarchies in a dimension. For example, a time hierarchy can describe the time elements in a calendar year. Another hierarchy might organize time in a fiscal year. Each hierarchy is a way to classify data in an organizing structure.

In hierarchies that have levels, there can be a special level at the top of the hierarchy called the *all level.* An all level only has one member in it, called the *all member*, that is the ancestor for all other members within a hierarchy. You can choose to not have an all level in your hierarchy. These kinds of hierarchies are called *multiroot hierarchies* because there is more than one member at the top.

In parent-child dimensions, each member must be unique within the entire hierarchy. Therefore, you must specify the unique key on the hierarchy itself.

5.1.5 Members

Members are the fundamental building blocks of a dimension. Each member represents a single entity within a dimension that the report authors and users might care about. For example, in the sample Products dimension there are Product line members, Product type members and Product members.

5.1.6 Attributes

Attributes are data that is associated with a member and describe characteristics of the member. You can define attributes by mapping them directly to columns in a table in your data warehouse, or you can create an attribute that is an expression incorporating one or more other attributes.

There are three special kinds of attributes. The first are the attributes that you use as the keys for defining your members. Depending on how the data in your database is structured, you might be able to use only one attribute to define your members, or you might need to designate more than one attribute as keys to uniquely identify each member you want to have. You must also designate one of your attributes as the caption for your members. The member caption specifies how members appear to your users. Finally, you can designate one of your attributes as the member description. Having a member description is optional. The decision about which attributes to use as the key, caption, and description is driven by your data and your business requirements. One attribute can be used for all three, or you can have one attribute for each. For example, for years in a time dimension, you might use the same column `CURRENT_YEAR` for both the key and the caption. However, for a product in a products dimension, you might use the `SKU` column as the key because each product has a unique stock keeping unit, but use the `PRODUCT_NAME` column as the caption so that the members are meaningful to your users.

Any other attributes that you choose to have are driven entirely by your business requirements. You can have as many as you like. For example, a product can have several extra attributes such as color, size, description, and introduction date, because your report authors and users are interested in that data or want to use those attributes to refine their queries. Be aware that there is a cost associated with each additional attribute in both increased start time for the cube and increased memory usage when the cube is started. Try to avoid having redundant or unused attributes.

5.1.7 Calculated members

Calculated members are members that the modeler can create by using expressions. They are appended to the end of the list of child members of their parent. The expressions that you use are dimensional expressions and they must resolve to a single numeric value or member. Therefore, you cannot use any of the following relational constructs in expressions for calculated members:

- Value summary functions (Not Member Summary functions)
- Value Analytic functions such as rank, first, last, percentile, percentage, quantile, quartile, distinct clause, and prefilter clause (Summaries/Member Summaries)
- Value Summary functions (standard-deviation-pop, variance-pop, distinct clause, prefilter clause)
- All running- or moving- summary functions (Summaries)
- All FOR clauses in aggregate functions (Summaries/Member Summaries)
- Date and time constants (Constants)
- All business date and time functions (Business Date and Time functions)
- Like, lookup, string concat '||', trim, coalesce, cast (Common Functions)
- MOD function (Common Functions)

There are two stages of validation for expressions in calculated members. The first stage is simple syntactic validation, and is available within the expression editor. The second stage is more comprehensive and happens when a dynamic cube is started within an IBM Cognos server. If there are any semantic errors within the expression, then the calculated member is dropped from the hierarchy. These errors are reported in IBM Cognos Administration. You can see the errors by selecting **View recent messages** in the context menu for the dynamic cube. See 7.2.1, "Starting a cube" on page 205 for details about how to view the messages for a cube.

5.1.8 Measures

The measure dimension contains the fact data, or measures, that you will use in your reporting and analysis. Measures are the members of the measure dimension. A dynamic cube can have any number of regular dimensions, but only a single measure dimension based on a single fact table in the data source. If your application requires using measures from multiple fact tables, you must create a virtual cube from cubes built on individual fact tables. See Chapter 8, "Virtual cubes" on page 223to learn more about virtual cubes.

You can extend the usefulness of cubes by creating calculated measures that have custom expressions that your consumers want to use. With calculated measures, you can build measures that are complex and reference other parts of your cube. By including calculated measures in your cube, your report authors do not need to re-create those calculations in their reports.

5.1.9 Named sets

A named set is an expression that resolves to one or more members. By defining named sets of important members after they are in your cube, the report authors do not have to re-create them in their reports. You can learn more about defining named sets in 5.5.3, "Named Sets" on page 139.

5.2 Starting a dynamic cubes project

When you are satisfied that you are ready to begin modeling your application, you can begin working in any one of a number of ways. You can begin building dimensions and then insert them into a cube, or you can start with a cube and then add dimensions. You can create measures in your cube before you add dimensions to it, or afterward. If you are starting with a cube, you can create the cube and the other objects manually, or you can use Cognos Cube Designer to automatically generate a cube. There is no one correct approach and you will likely find that different approaches are useful for different situations. For example, you might have received a model with dimensions already defined in it from a coworker, and your job is to build cubes using the dimensions that are already defined in the model.

There are no limits to the number of cubes that you can have in a Cognos Cube Designer project. However, in practice, a model can become too unwieldy to work with if there are a very large number of cubes in the model.

You can also create a cube by importing objects from a published IBM Cognos Framework Manager model. See Chapter 16, "Using Framework Manager models" on page 519 for more information about reusing the work that you have already done in Framework Manager.

Keep in mind that, when you are modeling a dynamic cube, you are specifying these aspects:

- The multi-dimensional structure of the data that the users and report authors will see.
- The relationship between the dimensional structure of the cube and the physical structure of the database. Cognos Dynamic Cubes uses this relationship to issue queries to the database at run time to populate the cube.

5.2.1 Managing locales

When you create a project, it is created with a single, default language set to English. However, many applications require you to support more than one language. Cognos Cube Designer allows you to add as many additional languages as you choose. To add or remove locales to your project, complete these steps:

1. Click the **Project** node in the Project Explorer.
2. Click **Add Locale(s)** in the Supported Locales property in the Properties table for the project. Figure 5-1 shows the Properties tab for the Project.

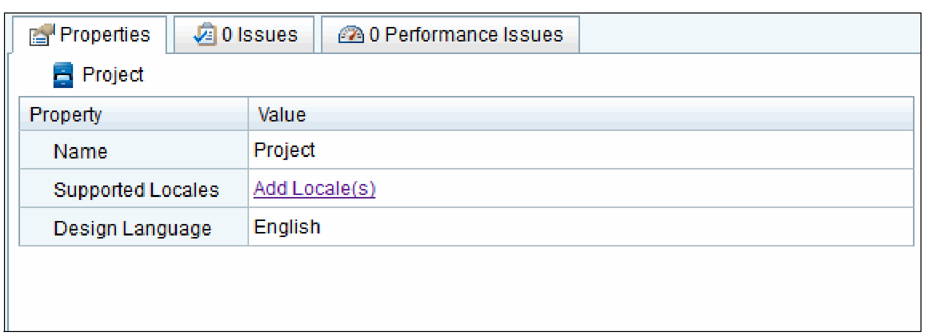


Figure 5-1 Properties tab for the Project

3. Select any locales that you want to add or remove to your project. If you remove a locale, you will be prompted to confirm it. Removing a locale removes any names in that locale from all objects in your model.

The *Design Language* is an important property to consider. Design Language is the locale used for the internal identifier of objects, and it is important to be aware of it when editing expressions. For multilingual applications, it is a common practice to choose a language that users are not interested in using to insulate the model from terminology changes while you are building it.

For any model object that appears in the studios, you can specify the names in those locales in the **Name** property of the object. Clicking the **Name** value shows a list with the names in all locales for that object. The first name in the list is the design language. Figure 5-2 shows the list of names for the Time dimension in the Cognos Cube Designer sample model.

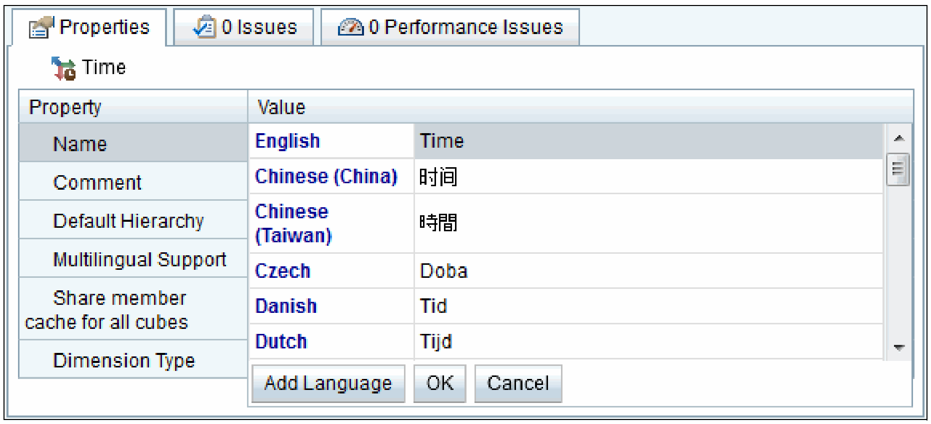


Figure 5-2 List of names for the Time dimension in the Cognos Cube Designer sample model

Click **OK** for Cognos Cube Designer to accept any changes to names, or click **Cancel** to discard the changes. Add or remove languages by clicking **Add Language** and selecting locales as desired. Changing the locales here has the same effect as though you clicked **Add Locale(s)** in the Supported Locales property of the project.

Cognos Cube Designer also supports modeling multilingual attributes, including the caption and description, for members in dimensions. For more information about modeling multilingual attributes, see 5.3.6, “Multilingual Attributes” on page 131.

5.2.2 Manually create a dynamic cube

The simplest and easiest way to create a cube is to right-click a folder or namespace and select **New** → **Cube** as shown in Figure 5-3.

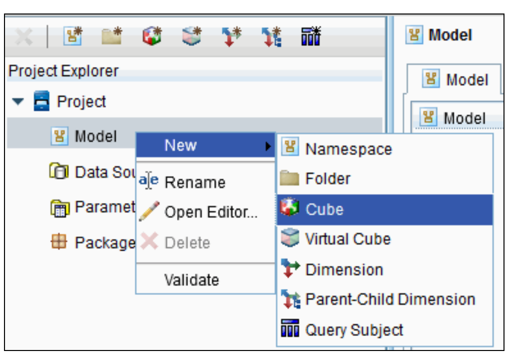


Figure 5-3 Context menu command for creating a cube

You can also select a folder or namespace and click **New Cube** in the toolbar in the Project Explorer as shown in Figure 5-4. The new cube is named New Cube and has an empty measure dimension named Measures. You can then rename the cube and begin adding dimensions and a measure dimension.

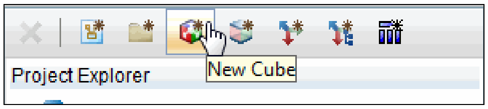


Figure 5-4 Project Explorer toolbar the New Cube in focus

5.2.3 Generate a dynamic cube

Cognos Cube Designer can generate a complete cube with dimensions based on information in your database. There are two methods for generating cubes from information in the database. Both methods rely on primary-foreign key relationships in the database to detect dimensions. If the data warehouse uses views that do not have primary-foreign key relationships, then you can only create a cube with measures, but no dimensions. In both cases, you must find your fact table in the metadata first. If you generate a cube from a table that is not a fact table, the generated cube will be of little use to you.

Two methods for generating cubes from information in the database are available:

- Generate a cube using only metadata from the database. This method looks for the primary-foreign key relationships between tables in the data source to detect and extrapolate the structure of the dimensions. In a star schema design warehouse, it creates a dimension for each dimension table that is joined to the fact table. It creates a default hierarchy with a single level. In the case of a snowflake warehouse design, it creates a dimension with a default hierarchy and a level for each table in the snowflake structure of the dimension, moving from the table furthest from the fact table inwards. Columns in each table are attributes of the level corresponding to its parent table.

  To generate a cube in this way, right-click the fact table that you want to use from the Metadata section (see Figure 4-2 on page 91) and select **Generate** → **Cube with basic dimensions** as shown in Figure 5-5.

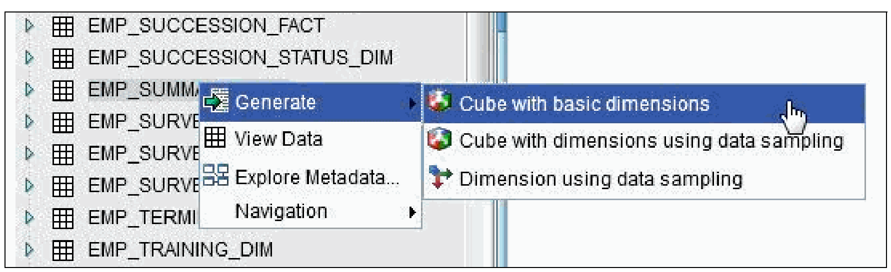


Figure 5-5 Generating a cube using only metadata from the database

- Generate a cube using data sampling. This second method differs from the first one in how the individual dimensions are generated. This method samples the data in any dimension tables it finds and infers the hierarchies and levels for the dimension from the data. With a denormalized dimension table, this method is likely to create extra levels or hierarchies in the dimension. For example, if a dimension table has columns for country, state, and city with appropriate data in the table, Cognos Cube Designer generates three levels in the dimension, corresponding to country, state, and city.

  To generate a cube using data sampling, right-click the fact table that you want to use from the Metadata section, and select **Generate** → **Cube** with dimensions using data sampling.

You can also generate individual dimensions using data sampling. See 5.3.7, “Generate a dimension” on page 132 for details.

You should always treat the generated cube as a starting point, not a finished application. Review each dimension and measure in the cube to ensure that they reflect your needs. You can rename objects, remove unneeded hierarchies or levels, and change any other property in the generated cube. Click the Issues tab for a list of issues that you have to address for each of the dimensions and measures.

Figure 5-6 shows a dynamic cube generated from the SLS_SALES_FACT table.

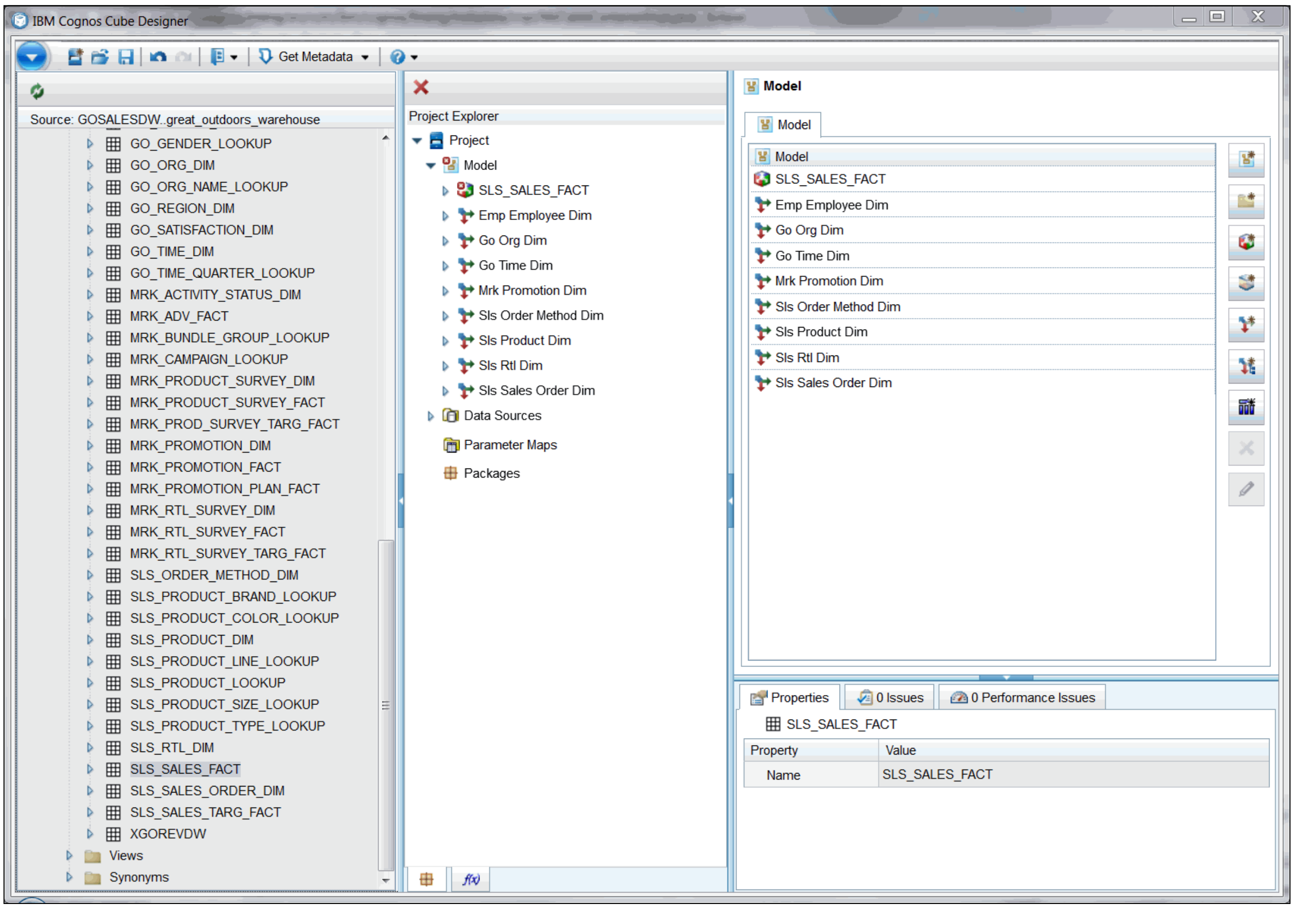


Figure 5-6 Dynamic cube generated from the SLS_SALES_FACT table

The cube inherits the name of the fact table because a dynamic cube can have only a single fact table. The dimensions inherit the names of the source tables. The names of the levels and hierarchies come from columns in the source tables. The structure of the data and the structure of the table affect results, including the order of the columns in their table. It is important to examine any generated dimension and cubes to ensure that they meet your business needs, and edit them if they do not.

5.3 Modeling dimensions

Dimensions provide context to the measures and delimit the spaces in which the measures have meaning. They have a structure that you specify to classify the data from higher to lower degrees of abstraction. A dimension provides the business context of the data.

The most common dimension is the *regular* dimension. Its defining characteristic is that it consists of one or more defined hierarchies. Each hierarchy is a set of ordered levels that relate the attributes of the dimension to each other and organize the data in them.

For example, a store dimension could be organized in a hierarchy that consists of levels of groups of countries, countries, subnational groupings such as states, provinces, or regions, cities, and stores. The branch dimension in the GO Sales sample cube model has a hierarchy of that nature. You can also have additional hierarchies that organize stores by other attributes, such as store size, store type, and so on.

You can model a dimension to specify specific attributes and members related to time. A time dimension allows your users to create reports with time-aware calculations, such as comparing the sales for the first quarter of the current year with the sales for the same quarter in the previous year. The steps to model a time dimension are described in 6.2, “Time dimensions and relative time” on page 168.

In parent-child dimensions, the organizational information is contained in the data. You must specify which member is a parent to another member. From that information, the tree of members is assembled when the dynamic cube is generated. You must specify that a member has only one parent. For more information, see 5.3.3, “Parent-child dimensions” on page 122.

In a data warehouse, dimensions are contained in either dimension tables or snowflakes:

- If the data of a dimension is contained in a single, denormalized dimension table, you must identify which columns correspond to more coarse detail and which columns correspond to more granular detail. The coarser details are good candidates for higher levels in a hierarchy whereas the granular details are good candidates for lower levels in a hierarchy. Frequently, you can identify the coarser details by the repetition of cell values.

  For example, consider the GO_REGION_DIM table in the `great_outdoors_warehouse` sample database. The column REGION_EN has duplicating values in the cells, representing coarser details that are appropriate for a higher level. However, the column COUNTRY_EN does not have any duplicating values, so it is appropriate for a lower level.
- If the dimension is in a snowflake, it is usually the case that the tables, because of the normalization of the entity relationships implied, map to levels in a hierarchy. However, a table might not represent a level, depending on what kind of data is in it. It might also be a lookup table. See 5.3.2, “Modeling hierarchies” on page 119 for more information about lookup tables. In any case, be aware of the structure of your data and the goals of your application.

You can create a dimension from objects that do not have relationships in the database. You must define relationships in the dimension. For information about how to create or modify relationships within a dimension, if necessary, see 5.3.4, “Joins” on page 123.

5.3.1 Modeling levels

The level editor contains the list of attributes that define the level, its characteristics, and the attributes that are associated with members in the level. Figure 5-7 shows the level editor for the Month level in the Time dimension of the Cognos Cube Designer sample model with the important controls identified.

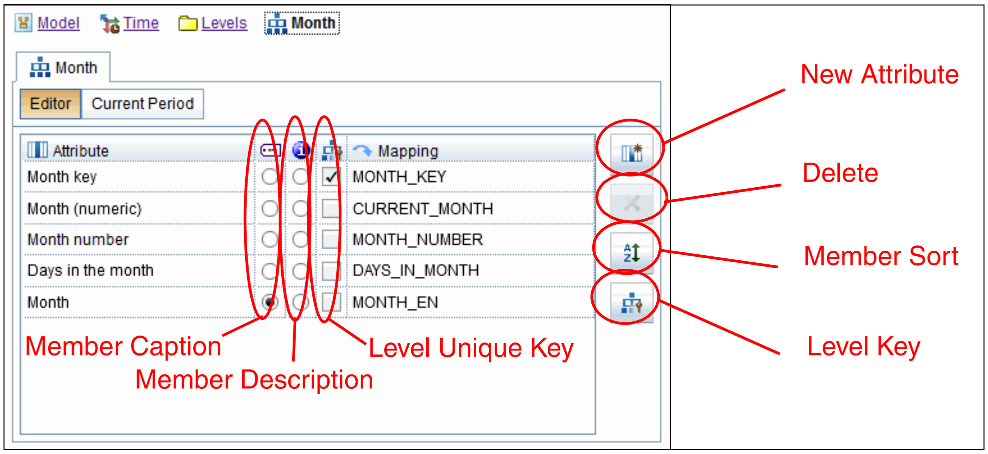


Figure 5-7 Level editor with important controls highlighted

You can create attributes by dragging columns from the Metadata tree into the level editor or by clicking **New Attribute**. You can delete attributes with the **Delete** button.

You must assign one attribute member caption for the members in the level, by selecting the appropriate radio button in the Member Caption column. It is not required that the value of the member caption be unique. However, it can be confusing for the report authors and users if they see multiple members with the same caption. If you want the member caption to be derived from multiple columns, you can create attributes for those columns, create another attribute with an expression that concatenates those attributes, and then select it to be the member caption. Such an expression is performed only once when hierarchy members are loaded into memory so they do not affect the performance of reports and analyses. If you do not assign a member caption for a level, you receive a validation error when you attempt to publish the cube.

It is optional to assign an attribute as the member description for members. If you assigned a member description to an attribute and then no longer want to have a member attribute, either click **Undo** in the toolbar (if you just did it) or delete the attribute that is assigned as the member description.

Use the Level Unique Key to identify which attributes can be used to uniquely identify members in the level, based on the value in the attributes you select. There are several options for specifying the level unique key, or level key, depending on the structure of your data. If a single attribute based on a column is sufficient, then you only need to select the check box next to that attribute. That column does not even have to be a key in the database table. The only requirement is that the value is unique when dynamic cubes queries the database. For example, in the sample database `great_outdoors_warehouse`, the column QUARTER_KEY in GO_TIME_DIM has values such as `20131`, `20132`, `20133`, and `20134` that can be used to uniquely identify members in a level for Quarters. If more than one column is necessary, select the check boxes for all that are required. Cognos Dynamic Cubes concatenates those columns internally when it is loading the dimension. You can also use an attribute that is derived from an expression based on one or more other attributes.

When you are working with denormalized dimension tables, it is common to be unable to uniquely identify the members for that level based on the attributes in that level. In those cases, click **Level Key** and add attributes from other levels. For example, the MONTH_NUMBER column in GO_TIME_DIM has values such as 1, 2, and 3. If you have data for multiple years, then the MONTH_NUMBER column does not sufficiently identify unique members in the level. However, in a time dimension with levels for Quarters and Years, you can uniquely identify members in the months level by also including attributes from the other two levels as well as shown in Figure 5-8.

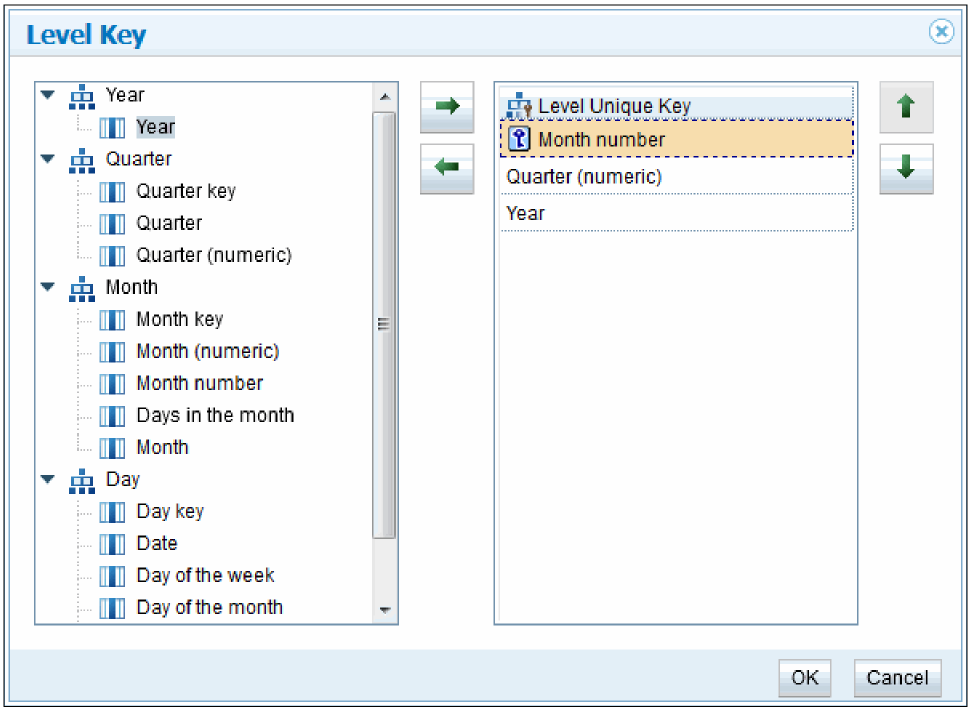


Figure 5-8 Level Key window

The first item in the level unique key list is used as the business key and is identified by the key icon next to it. You can reorder the attributes if you want to use a different one for the business key. You use the business key for constructing drill-throughs with dynamic cubes. Drill-throughs are outside the scope of this book.

If the keys for one or more of the levels of a cube are not sufficiently unique, an error message like the one shown in Figure 5-9 is displayed stating that more than one member was found with a member key when you publish the cube. See 5.8, "Deploying dynamic cubes for reporting and analysis" on page 154 for details about publishing the cube.

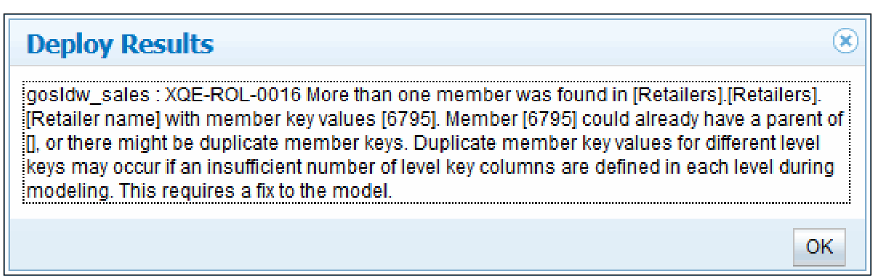


Figure 5-9 Example of the cube start error message that occurs when level keys are not unique

Usually this situation happens when you need to include the level keys from higher levels to attain sufficient uniqueness. You can usually solve this problem by starting the Level Key window for the level referred to in the error message and adding the keys from levels that are higher in the hierarchy.

You can also specify how members are sorted. Start the Member Sort window by clicking **Member Sort**. Figure 5-10 shows an example of sorting members.

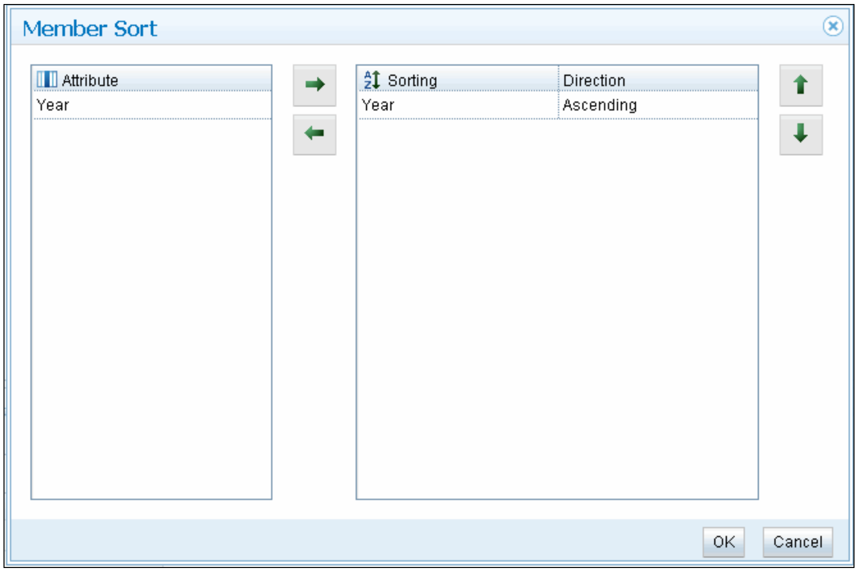


Figure 5-10 Sorting members

You can sort on multiple attributes and sort them independently. For example, you can sort an employee level by the employee surname, ascending, and then the employee start date, descending. Each sort is performed on the members, based on the preceding sort order. If there are any NULL values in the attributes used for sorting, the members are sorted last if Ascending is chosen and first if Descending is chosen.

There are two properties specific to levels: Level Type and Current Period. Both of these properties are used for creating time-aware dimensions. See 6.2, "Time dimensions and relative time" on page 168 for more information about these properties.

Considerations in Modeling Levels

Some reporting requirements can specify alternate hierarchies that organize the members based on particular attributes that are of interest to users. Be aware that the levels must identify the members as being unique within the context of every hierarchy. It is possible that these hierarchies can produce non-unique members, although the level key can produce uniquely identified members in other contexts. The Members folder in each hierarchy is useful for examining the members to ensure that they are correct. If you make any changes to levels, remember to refresh the Members folder by right-clicking it and selecting **Refresh**.

Figure 5-11 shows the menu of the Members folder in the Products hierarchy.

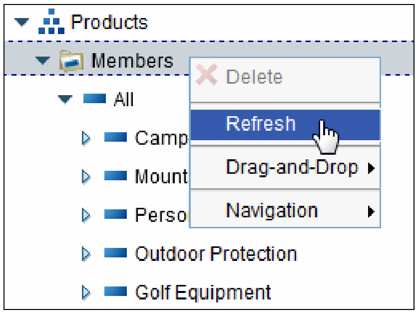


Figure 5-11 Menu of the Members folder with Refresh highlighted

In modeling for multilingual models, it is important to use level keys that do not vary from one locale to another.

Be aware of the potential number of child members that are generated for any particular member in the member tree. It can be difficult to navigate through a large number, and a very large number can impede performance. For example, consider a Time dimension with only two levels: Years and Days. Each member in the Years level has hundreds of child members, making navigation difficult. Discuss with your consumers what sort of reports they will write. With this information, you can create hierarchies that categorize members into smaller, more exact sets.

You can use that discussion as an opportunity to refine the model to meet the needs of the consumers. They might have requirements that they did not tell you about or did not request because they might not believe that modeling it is possible. This information can assist your consumers in generating reports and designing the functionality that might otherwise need to be specified in the report. Additional functionality built into the cube can speed up report processing time. It can have practical benefits of easing communications and other aspects of office politics.

An attribute expression can have a null result if one element of the expression returns a null value. For example, some records in EMP_EMPLOYEE_DIM have null values for the column Address2. An expression that uses Address2 returns null results for each record where the value for Address2 is null. If you have data with null values, you must incorporate tests for null values to handle them correctly.

A novice modeler might just include the Address1 and Address2 attributes without checking for null values as follows:

```
Address1|| ' ' || Address2
```

However, an expression that tested for null values in Address2 might look like the following:

```
Address1|| ' ' || if (Address2 is null) then ( ‘‘ ) else (Address2 )
```

If there are null values for an attribute that is being used in the level key, some members might not be generated. If the null value happens in a member caption, the member displays NULL as its caption, even if other elements of the expression that is used to define the member caption are not null values. Expressions of this type usually need the data type to be the same.

5.3.2 Modeling hierarchies

Hierarchies allow users to perform analytical drill up and drill down operations. Hierarchies can also help users to navigate to particular members they might be interested in. You can have multiple hierarchies in a dimension to provide alternate drill paths or organizations of members to suit the needs of multiple users.

Add levels to a hierarchy by dragging the level from the *Levels* folder to the hierarchy. If there is already one level in the hierarchy, you can add more levels by placing them relative to existing levels in the hierarchy. Drag-and-drop the level reference within the hierarchy to reorder the levels in a hierarchy. The avatar is colored green or red depending on whether you are able to drop the level reference at that location or not. Figure 5-12 shows reordering the *Product type* level within the Products hierarchy in the dynamic cubes sample model. The green avatar and the thick black line above the Product line shows that the Product type level can be moved there.

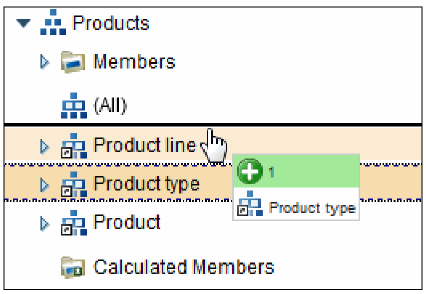


Figure 5-12 Product type level being dragged above the Produce line level in the Products hierarchy

You must always bear in mind the requirements of your users and the structure of the data when ordering levels within a hierarchy. For example, although Cognos Cube Designer allows you to order a Months level above a Years level within a time hierarchy, this arrangement will likely confuse your users (if it works at all).

Figure 5-13 shows what happens if the *Product type* level is dragged above the *(All)* level. The avatar turns red because the (All) level must always be the highest level in the hierarchy.

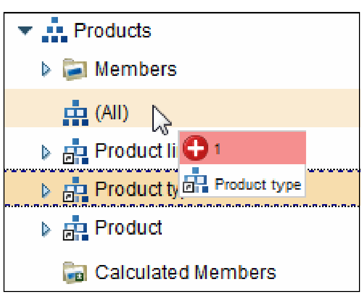


Figure 5-13 Attempting to reorder a level above the (All) level

The (All) level is a predefined level that contains a single member known as the ALL member. The ALL member is the ancestor member for all other members in the hierarchy and usually the starting point for users to navigate into the hierarchy in the Cognos studios. Although you cannot move the (All) level, you can rename it the same way that you rename any other object in the model. It supports multiple locales as well, if your application requires that. You can also change the name and caption of the ALL member through the Root Member and Root Caption properties of the hierarchy. The Root Member property specifies the internal name of the member that is used in calculations in reports, whereas the Root Caption is a multilingual property that specifies how users will see the ALL member in different locales. Figure 5-14 shows the members of the Time hierarchy with the All member at the root.

Figure 5-14 The members of the Time hierarchy with the All member at the root.

There are situations where it does not make sense to aggregate values to a single root member. For example, assume that you have a budget dimension. In the main hierarchy, you want to have three root members that represent three different budgeting scenarios: Planned, budget, and actual. In that case, it makes no sense to aggregate values for those three members. To change a hierarchy to a multi-root hierarchy, set the **Multiple root members** property of the hierarchy to `true`.

Figure 5-15 shows the Time hierarchy from the dynamic cubes sample model with the **Multiple root members** property set to `true`. Notice that the (All) level has been removed from the hierarchy.

Figure 5-15 Members of the Time hierarchy with multiple root members

Changing the Multiple root members property back to `false` inserts a default (All) level with a default ALL member.

As described in 5.1.1, "Cubes" on page 106, a *tuple* is a point in a cube and is composed of some specified members as well as, implicitly, the default member from every hierarchy in every dimension in the cube. You can specify the default member of a hierarchy by dragging a member from the Members folder of the hierarchy to the Default Member property of the hierarchy. For a hierarchy with an (All) level, the ALL member is the default member for the hierarchy if none is specified. For a multi-root hierarchy, the first member encountered in the top level is the default member if none is specified. Changing the default member of a hierarchy can affect every aggregation that is performed by the cube, so you should test your application to ensure that you are receiving the results that you expect.

Depending on the structure of your data, there might be missing members in a hierarchy. For example, consider a Geography dimension with levels for Country, State or Province, and City. In such a dimension, The Vatican is usually modeled as a member in the City level whose parent is a member in the Country level. Dynamic cubes do not allow having gaps like this one in a hierarchy, and automatically inserts padding members to ensure that there are no gaps in the hierarchy. The hierarchy properties **Show Extraneous Padding Members** and **Caption of Padding Members** can be used to specify whether the padding members are shown to users or not, and whether they have a visible caption or not.

5.3.3 Parent-child dimensions

Parent-child dimensions do not have defined levels. The data of the parent and child attributes determine how the members are generated and assembled into a hierarchy. Parent-child dimensions do not have level keys. The child attribute uniquely defines every member in the hierarchy. Parent-child dimensions can have only one hierarchy. Parent-child dimensions are useful when the items you want to report on are not easily organized into discrete levels. The most common use of a parent-child dimension is to model the personnel chart of a typical organization. Figure 5-16 shows the attributes for the *Employee by manager* dimension in the dynamic cubes sample model.

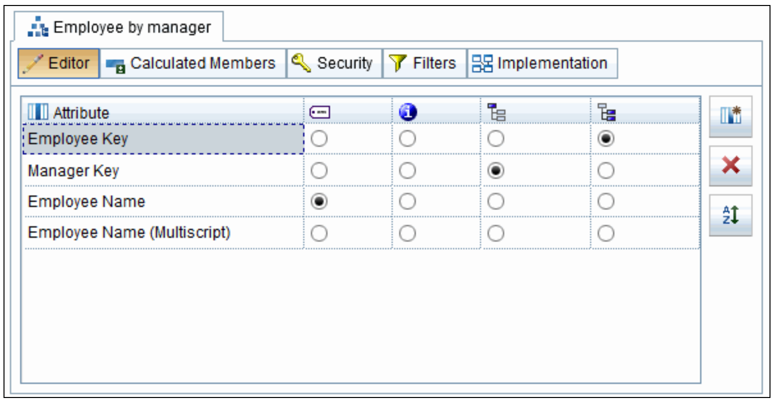


Figure 5-16 Attributes of the Employee by manager dimension in the sample model

The attributes assigned as Member Caption and Member Description function identically as they do for members in a level. The other two columns of radio buttons are for specifying the parent and child attributes. In the Employee by manager dimension, the parent attribute is Manager Key, which is mapped to the MANAGER_KEY column. The attribute that defines unique members is Employee Key, which is mapped to the EMPLOYEE_KEY column. In the underlying table, the MANAGER_KEY column is a foreign key to the EMPLOYEE_KEY column, which is the primary key of the table. It is most useful to model tables with these kinds of self-joins as parent-child dimensions. You can use attributes built from expressions as the parent and child attributes, but you must ensure that the values match or you will get a wide, flat hierarchy that is difficult for the users to use.

If the parent-child dimension is a slowly changing dimension, the parent and child attributes must be surrogate keys. You can read more about slowly changing dimensions in 6.3, “Slowly changing dimensions” on page 179.

Data members are members that contain values that are associated with the parent member itself. The data member allows you to see those values. If the data member is not specified, the values are still aggregated into the parent member, but there is no indication of the source of that value. For aggregations such as *sum*, it is fairly readily apparent that the parent member has values that are associated with it. For others, such as *average*, it is less apparent. It is a good idea to include them by setting the **Show Data Members** property value to `true` and setting the Caption of Data Members property value to Parent’s caption.

Figure 5-17 shows how a parent-child dimension looks in IBM Cognos Report Studio when the **Show Data Member** property is set to `true` and the Caption of Data Members property is set to Parent's caption.

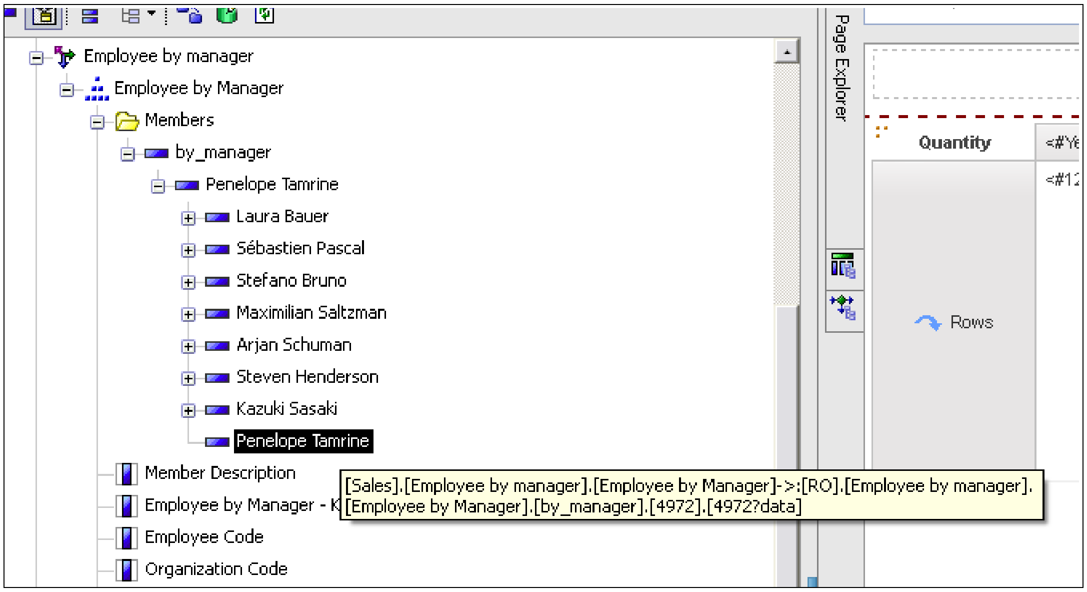


Figure 5-17 Parent-child dimension

If specified, the Root Member property is the name of an artificial member that is the ancestor of all other members in the hierarchy. By default, it is empty, so no additional member is inserted at the root and the root or roots of the hierarchy are derived from data. The Root Caption property is only applicable if the Root Member property has a value, and otherwise behaves the same as the property does in a hierarchy with levels. The Default Hierarchy property also behaves the same as the property does in a hierarchy with levels. Caption for the member as displayed in the member tree in the Cognos studios is the name of the dimension.

Functions that reference a member in the context of a level or descendants return each child member, that is many member nodes down the tree. If you have an unbalanced hierarchy, the member tree is generated without any additional modeling required.

5.3.4 Joins

When you are building a dimension, it is common to want to build a dimension using columns from more than one table. Cognos Cube Designer adds and removes tables automatically as you map and unmap attributes to columns from your database. Cognos Cube Designer attempts to create the joins between those tables automatically, based on any keys in the tables and how those tables are used in levels. However, there might be situations where Cognos Cube Designer cannot automatically determine how two tables are joined, and you must specify the relationship yourself. The tables and relationships for a dimension can be found in the Implementation tab of the dimension. You edit the relationships within a dimension in the Implementation tab.

Figure 5-18 shows the *Implementation* tab for the Branch dimension. The modeler uses columns from each table to define level keys in the dimension.

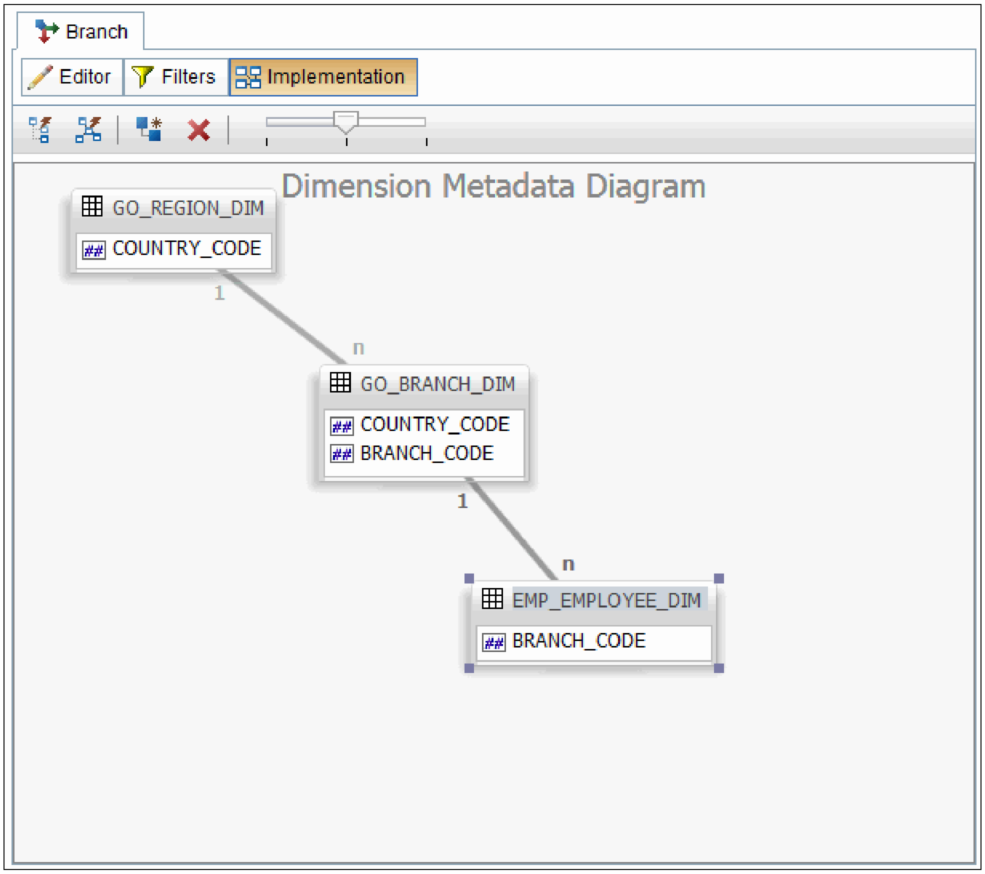


Figure 5-18 Implementation tab for the Branch dimension

To edit a join between two tables, double-click the line connecting them to open the Join Editor. Figure 5-19 shows the join between the GO_REGION_DIM and GO_BRANCH_DIM tables used by the Branch dimension.

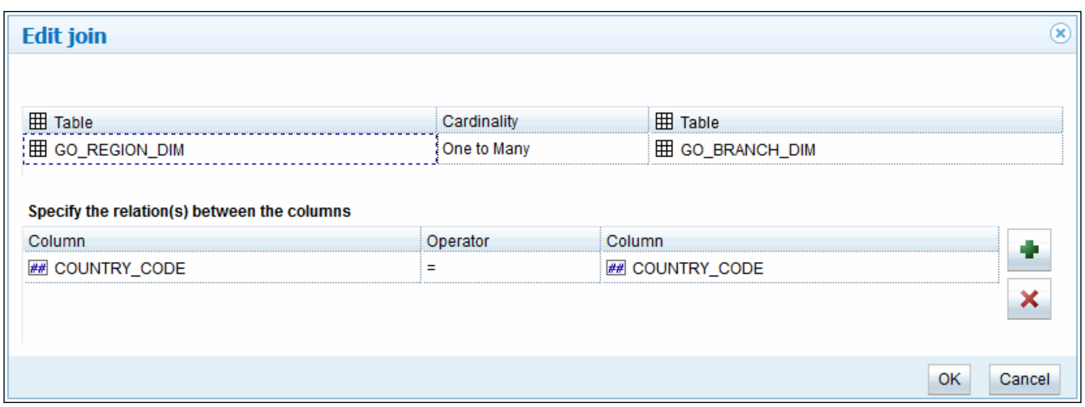


Figure 5-19 The Edit join dialog.

You specify which columns are used to join the two tables in the bottom part of the Edit Join window. The left and right side of each row contains drop-down lists with all of the columns in the table for that side. Select which columns are to join on. If the join requires more than one set of columns, you can add more sets by clicking **Add join to expression**. Similarly, if there are extra sets of columns specified, you can remove them by selecting the row and then clicking **Delete**. The columns that you join on do not need to be primary or foreign keys in the table. In fact, if your database does not have referential integrity, you must specify how the columns are related in each join yourself.

The top part of the editor shows the two tables that the join is for and the cardinality of the join between them. You can change the cardinality according to the needs of your modeling situation, and can use different cardinalities for different situations in your data warehouse. There are five possible options: *Many to One, One to Many, One to One, Left Outer*, and *Right Outer*. It is important to consider which table is on which side of the join because for most of these cardinalities, the side is meaningful.

Many-to-one and one-to-many cardinality

The most common cardinalities that you are likely to use are one-to-many or many-to-one. In a snowflake schema, these cardinalities usually indicate that the two tables have level keys for two separate levels (although each individual table can be used to populate the members of more than one level, depending on how your data is structured). In fact, Cognos Cube Designer requires that two tables have a one-to-many join if both tables have level keys in them. The Branch dimension that is shown in Figure 5-18 on page 124 is a good example of a dimension built on multiple tables, where each table contributes level keys.

One-to-one

Use one-to-one joins to describe situations where the columns of a table do not contribute any level keys to defining the level, but the table does contain additional attributes that you want to make available to your users. These tables with additional attributes are called *lookup tables*.

Typically, a lookup table has a join (either logical or physical) to a table with level keys but it does not have any level keys itself. Instead, the lookup table has extra attributes for the members that are defined from the table with level keys. You should ensure that each row of attribute data in the lookup table corresponds to a row from the table with level keys.

Figure 5-20 shows an example of using lookup tables in a dimension. The table SLS_PRODUCT_DIM is the single dimension table where all the level keys for each level come from. However, the captions for each Product, Product Type, and Product Line are stored in separate lookup tables.

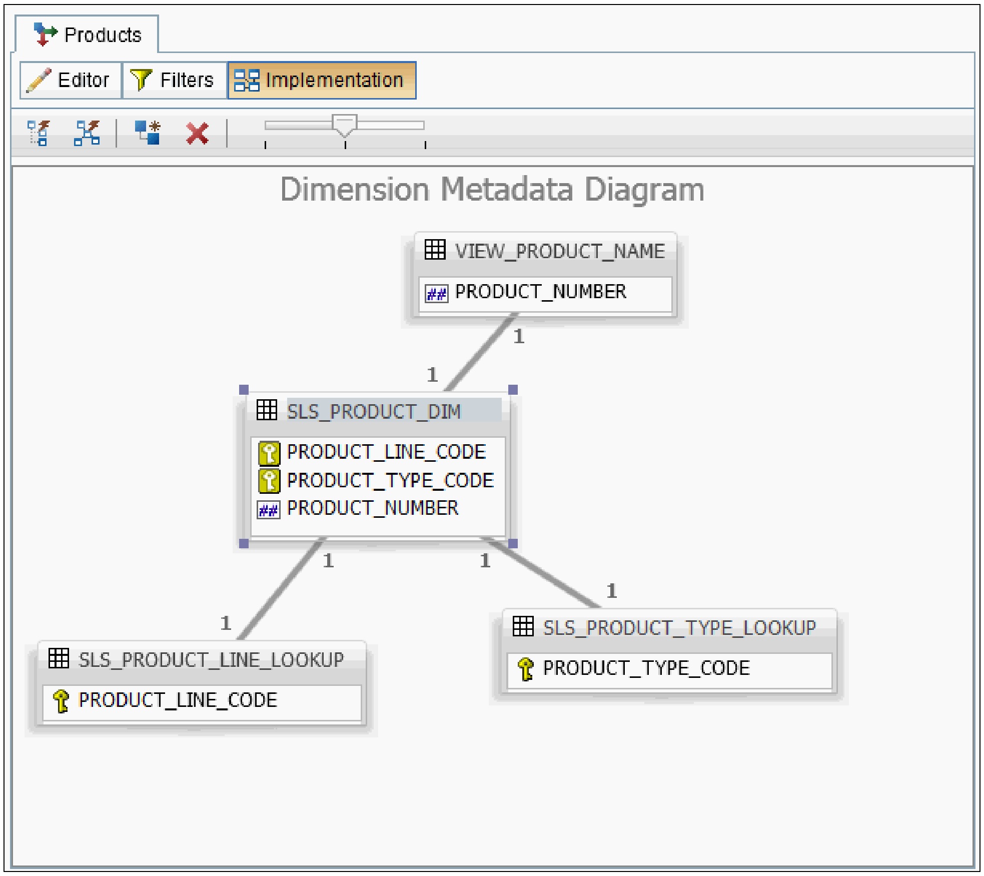


Figure 5-20 Products dimension showing lookup tables

Left outer and right outer joins

You might want to create an attribute from a column in a lookup table, but the corresponding dimension table is incomplete. That is, there are rows in the dimension table that do not have the corresponding rows in the lookup table. For example, suppose that you have the following tables and want to build a Customer dimension. Table 5-1 shows the Customers table.

Table 5-1 Data for the Customers table

Customer_No	**Demographics_ID**
001	002
005	Null

Table 5-2 shows the data in the Demographics table.

Table 5-2 Data for the Demographics table

Demographics_ID	**Gender**
001	Female
002	Male

In the Customers dimension, add an attribute for Demographics. Cognos Cube Designer creates a join between the two dimensions for you, with Customers on the left side and Demographics on the right side. If you specified a one-to-one cardinality for the relationship, the query that Cognos Dynamic Cubes issues to the database uses an inner join, so any customer that had a NULL value for the Demographics_ID column is not returned. The solution is to specify left outer for the cardinality. The query that Cognos Dynamic Cubes issues to the underlying database uses a left outer join to join the two tables together. All rows from the Customers table are retrieved, as well as any row with a corresponding Demographics_ID value from the Demographics table.

5.3.5 Dimension filters

Dimension filters allow you to apply a filter to the data stream that is used to populate the members in your dimension during cube start. You can use them to restrict the type of member that is loaded to a more useful subset. Having fewer members in the dimension also improves cube start time and reduces the memory footprint required for the cube. It is important to note that filters act on the data that all users of a cube see. If you want to have filters that only apply to some of the users, you must use security filters.

Filters are components of the relational query that are used to fetch data for the cube, so you can only use relational expressions in a filter. In your expression, you can refer to any attribute in the dimension that you are defining the filter for, or any measure in a cube that uses the dimension.

Consider the *Employee by region* dimension in the Cognos Cube Designer sample model. Figure 5-21 shows the levels and hierarchy of the Employee by region dimension.

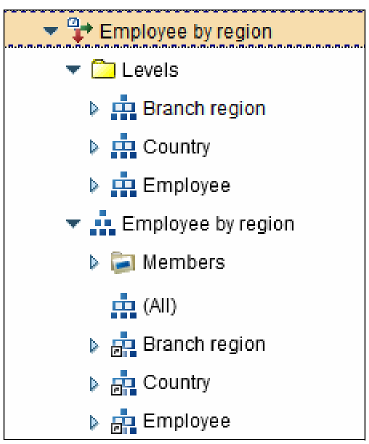


Figure 5-21 Employee by region dimension

As it is constructed, the Employee by region dimension includes all employees, past and present. However, if the objective of your application is to track the sales targets of employees, it does not make sense to include people who are no longer employees. The Termination Code attribute of the Employee level can be used to filter out past employees.

To create the filter, complete the following steps:

1. Double-click the **Employee by region** dimension in the Project Explorer to open the dimension editor.
2. Expand the **Employee by region** dimension, the **Levels** folder, and the **Employee** level.
3. If necessary, scroll down in the Project Explorer until the Termination Code attribute is visible.

4. Select the **Filters** tab in the dimension editor.
5. Click **New Filter** on the right to create a new filter.
6. Click **New Filter** in the list of filters. The properties of the filter are shown in the Properties tab of the Details window.
7. Change the name of the filter to **Active Employees**.
8. Double-click the **Expression** property to open the expression editor for the filter.
9. Drag **Termination Code** from the Project Explorer to the **Expression window** and enter:

   = '150'
10. Leave the **Exclude Facts Without Corresponding Dimension Keys** property set to `true`.

When you finish, the filter should look like the Active Employees filter shown in Figure 5-22.

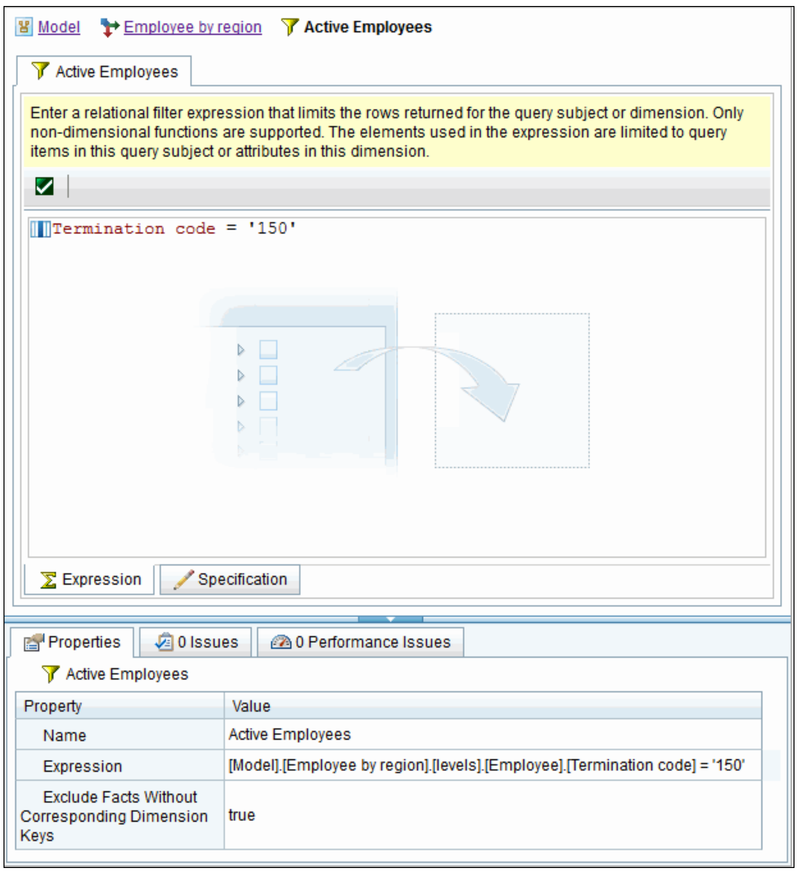


Figure 5-22 The Active Employees filter

There are a couple of ways you can test a filter to ensure that it is working correctly:

- If you know of members that should no longer appear in the hierarchy, you can refresh the Members folder of the hierarchy and navigate to where the members were before. In the previous example, the employee Vern Ritchie voluntarily left (a Termination code of 152), so Vern should no longer appear in the list of members under the United States.
- Another way to verify your filter is to right-click the level whose attributes the filter uses, and select **View Data**. In the previous example, you can view the data for the Employee level and verify that the values in the Termination Code column are all 150.

Each method has advantages and drawbacks. By browsing the hierarchy members, you can see every member that is in the hierarchy, but no other details. By viewing the data of a level, you can see every attribute modeled for the level, but only a subset of all the data rows that will be used to populate the level. You are free to use whichever method best suits your needs, or use both of them.

Figure 5-23 shows the filter applied to the Employee by region dimension.

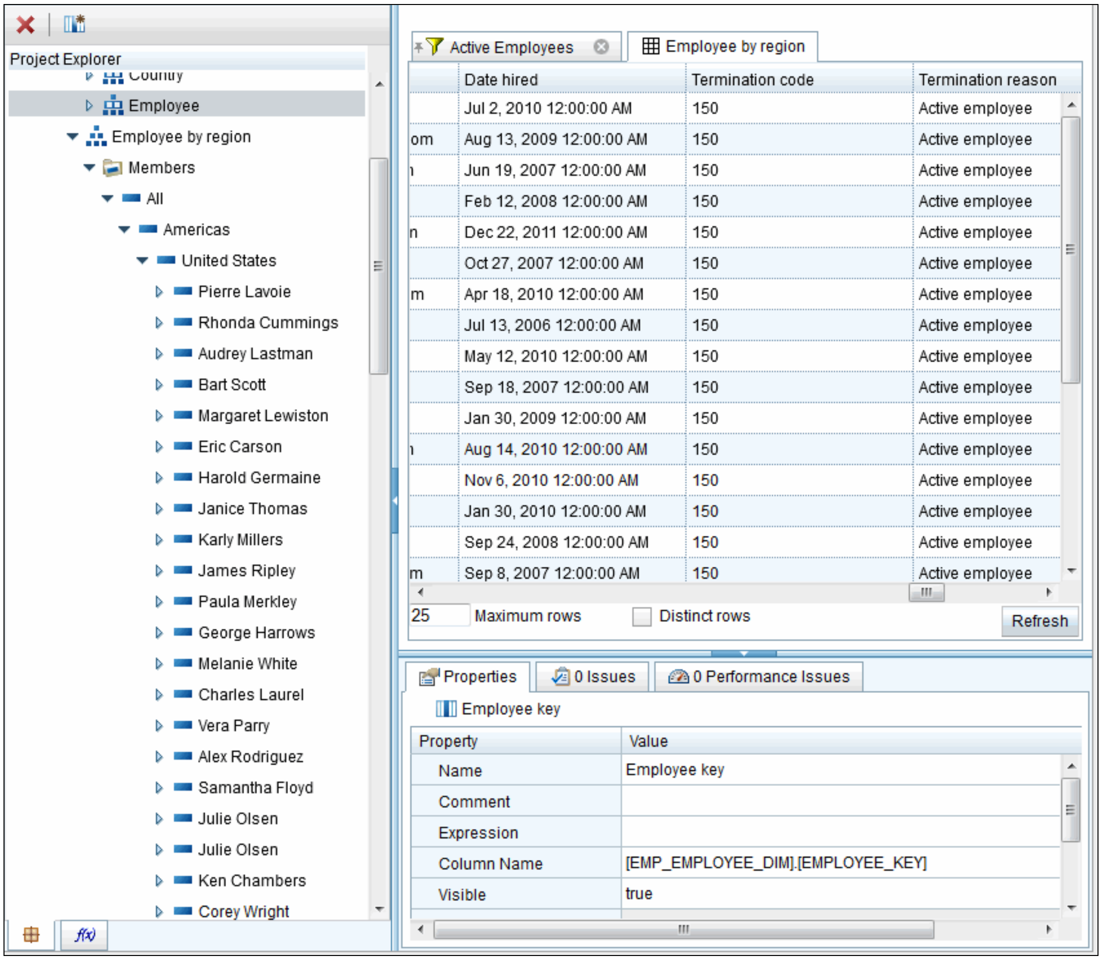


Figure 5-23 Employee by region dimension with the Active Employees filter applied

Exclude Facts Without Corresponding Dimension Keys is an important property to understand. If this property is set to `true`, then a data filter is applied to your fact table to ensure that only fact rows that have a corresponding key in the dimension table are used. Cognos Cube Designer generates this additional data filter when the cube is published, based on the relationship between the dimension and the measure dimension. If you define multiple filters in your dimension, and even one filter has the **Exclude Facts Without Corresponding Dimension Keys** property set to true, then the fact data is applied.

However, there are scenarios where you should set **Exclude Facts Without Corresponding Dimension Keys** to `false`. If the filter applies to data that is not used by your members, or the filter uses fact table data to filter your dimension, then you should set this option to `false`.

Examples of both of these scenarios are shown in "Using a data filter with a lookup table" on page 130 and "Excluding dimension members without fact data" on page 131. It is critical to understand your data so that you know whether you should exclude fact data or not.

Using a data filter with a lookup table

There can be situations where a normal join to a lookup table can result in a query that returns more rows than needed by the cube. This is an efficiency problem that can be addressed by filtering the data in the query, instead of retrieving unneeded rows. In these situations, **Exclude Facts Without Corresponding Dimension Keys** should be set to `false` because the filter does not affect the number of members that are generated for the level. Having **Exclude Facts Without Corresponding Dimension Keys** set to `true` in this case causes the Cognos server to issue a more complicated query than necessary.

Consider the table SLS_PRODUCT_LOOKUP in the `great_outdoors_warehouse` sample database. The primary key for the table is composed of both the PRODUCT_NUMBER and the PRODUCT_LANGUAGE columns. There are rows for each combination of product number and language. If you wanted to have a unilingual cube, returning the names for all languages is inefficient. Instead, you can create a filter to exclude rows in other languages as follows:

1. In a Products dimension, create a Product level with the PRODUCT_NUMBER column from SLS_PRODUCT_DIM and the PRODUCT_LANGUAGE and PRODUCT_NAME columns from SLS_PRODUCT_LOOKUP. Set Product Name to be the **Member Caption** and Product Number to be the **Level Unique Key** as shown in Figure 5-24.

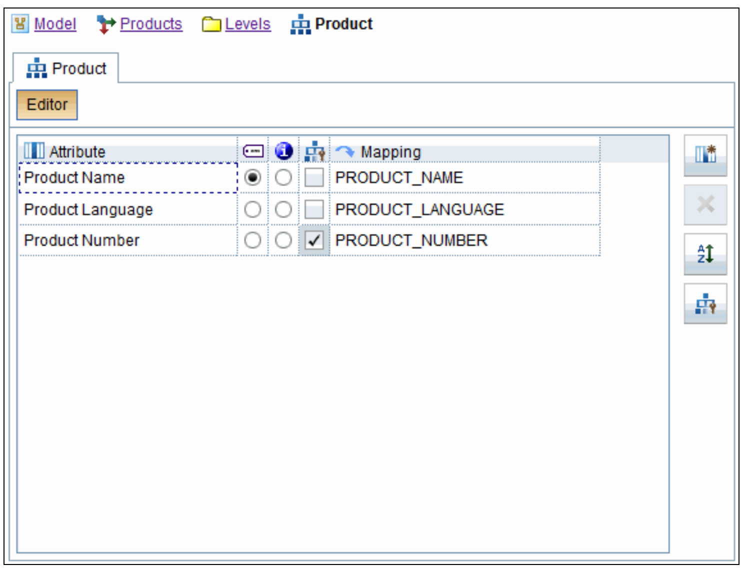


Figure 5-24 Product level.

2. Select **Product Language** and set the Visible property to `false`.
3. Double-click the **Products** dimension in the **Project Explorer** and select the **Implementation** tab.
4. Double-click the line connecting the SLS_PRODUCT_DIM table and the SLS_PRODUCT_LOOKUP table, set Cardinality to **One to One** and click **OK**.
5. Select the **Filters** tab and create a filter.
6. Click **New Filter** in the Filters list.
7. Set the Name of the filter to **English Only**.
8. Set **Exclude Facts Without Corresponding Dimension Keys** to **false**.
9. Double-click **Expression**.

10. Drag the **Product Language** attribute from the Project Explorer into the Expression Editor.

11. Enter `= 'EN'` in the Expression Editor. The expression should look like the one shown in Figure 5-25.

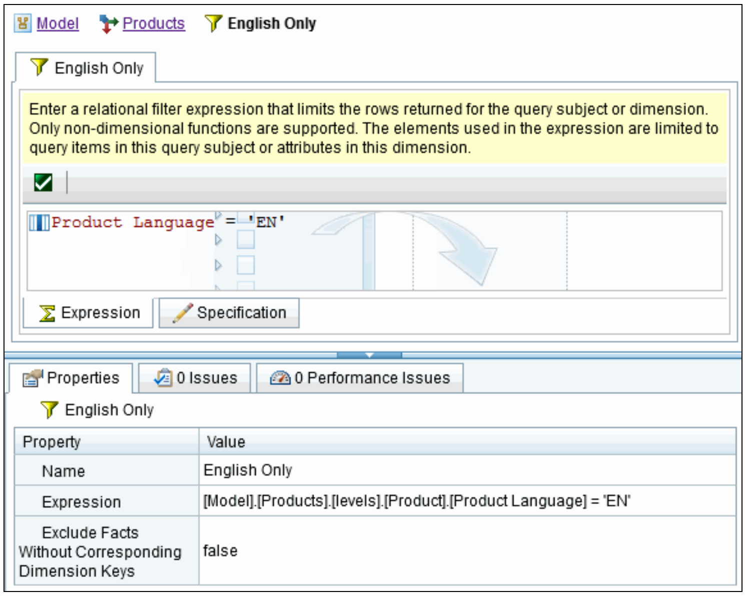


Figure 5-25   English Only filter.

Now when the Products dimension is populated by Cognos Dynamic Cubes during cube start, Cognos Dynamic Cubes only fetches rows from the SLS_PRODUCT_LOOKUP table that have a product language of `EN`, rather than fetching all rows.

Excluding dimension members without fact data

You can exclude members from your dimension that do not have the corresponding fact data in the fact table by using a filter. An example situation is with a cube for sales data. You might not want to include products that do not have any sales from the cube, for example because they are new products that have not been released yet.

To exclude dimension members without fact data, you must first create a measure in the measure dimension of the cube on the key column that joins to the dimension. Set its **Visible** property to `false` and then create a filter using that key column. You must set the **Exclude Facts Without Corresponding Dimension Keys** property on the filter to `false`, or you might have problems with circular references in your queries to the database.

5.3.6 Multilingual Attributes

Unlike model objects that you define in the model, members are derived from the data in the database. If you want to have multilingual attributes for members, the attribute values must come from the database. To model a multilingual attribute, complete these steps:

1. Select the dimension for which you want to model multilingual attributes in the Project Explorer.
2. Click the **Multilingual Support** property in the Properties tab for the dimension and select **By Column**.
3. Select the attribute that you want to model in your dimension within the Project Explorer.

4. In the Properties tab for the attribute, change the **Multilingual** property to `true`.
5. In the same Properties tab, select the **Column Name** property. Cognos Cube Designer shows a list of languages and values similar to the one for multilingual model objects. However, instead of names, the values are column references from your model.
6. Locate the column in the metadata with the values you want to use for the attribute in that locale, and drag it to the corresponding locale property in the list.

Figure 5-26 shows the columns that are mapped to the Weekday attribute from the Time dimension in the same model. The columns come from the GO_TIME_DIM table in the `great_outdoors_warehouse` sample database.

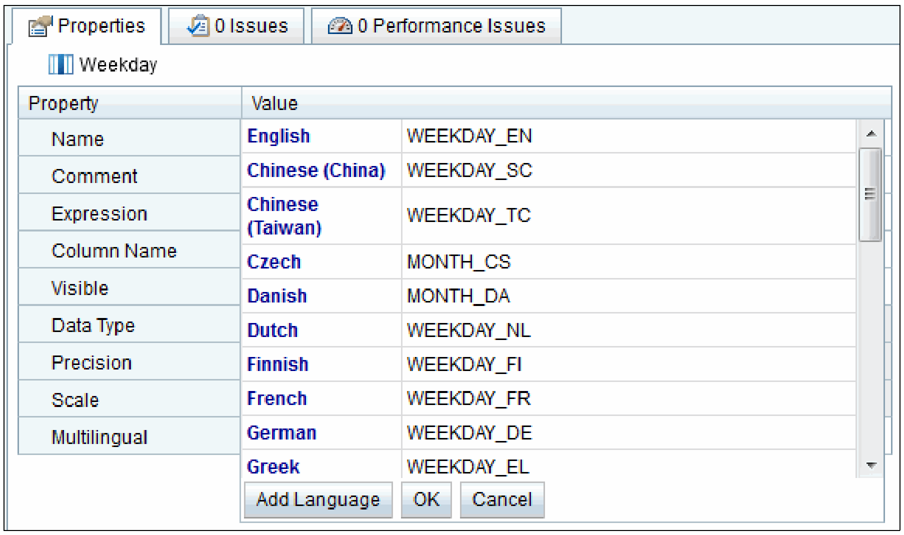


Figure 5-26 Columns used for the Weekday multilingual attribute

5.3.7 Generate a dimension

You can generate individual dimensions from the data in the database similarly to how you can generate complete cubes. Generating a dimension works best with fully denormalized dimension tables. If your data warehouse uses snowflake dimension tables, you are better off creating your dimensions manually.

To generate a dimension, complete these steps:

1. Select the appropriate dimension table in the metadata
2. Right-click the dimension table
3. Select **Generate** → **Dimension using data sampling**.

5.4 Modeling measures

Dimensions in a cube give context to measures. Measures are also known as metrics, facts, or KPIs. They are the values that the decision makers in an organization use to make key decisions concerning the organization.

5.4.1 Creating a measure

The simplest way to create a measure is by dragging columns from a fact table in the metadata to the measure dimension editor in the cube. You can select individual columns and only drag those if you choose, or you can drag the entire table.

Figure 5-27 shows dragging three columns from the SLS_SALES_FACT table to the measure dimension editor.

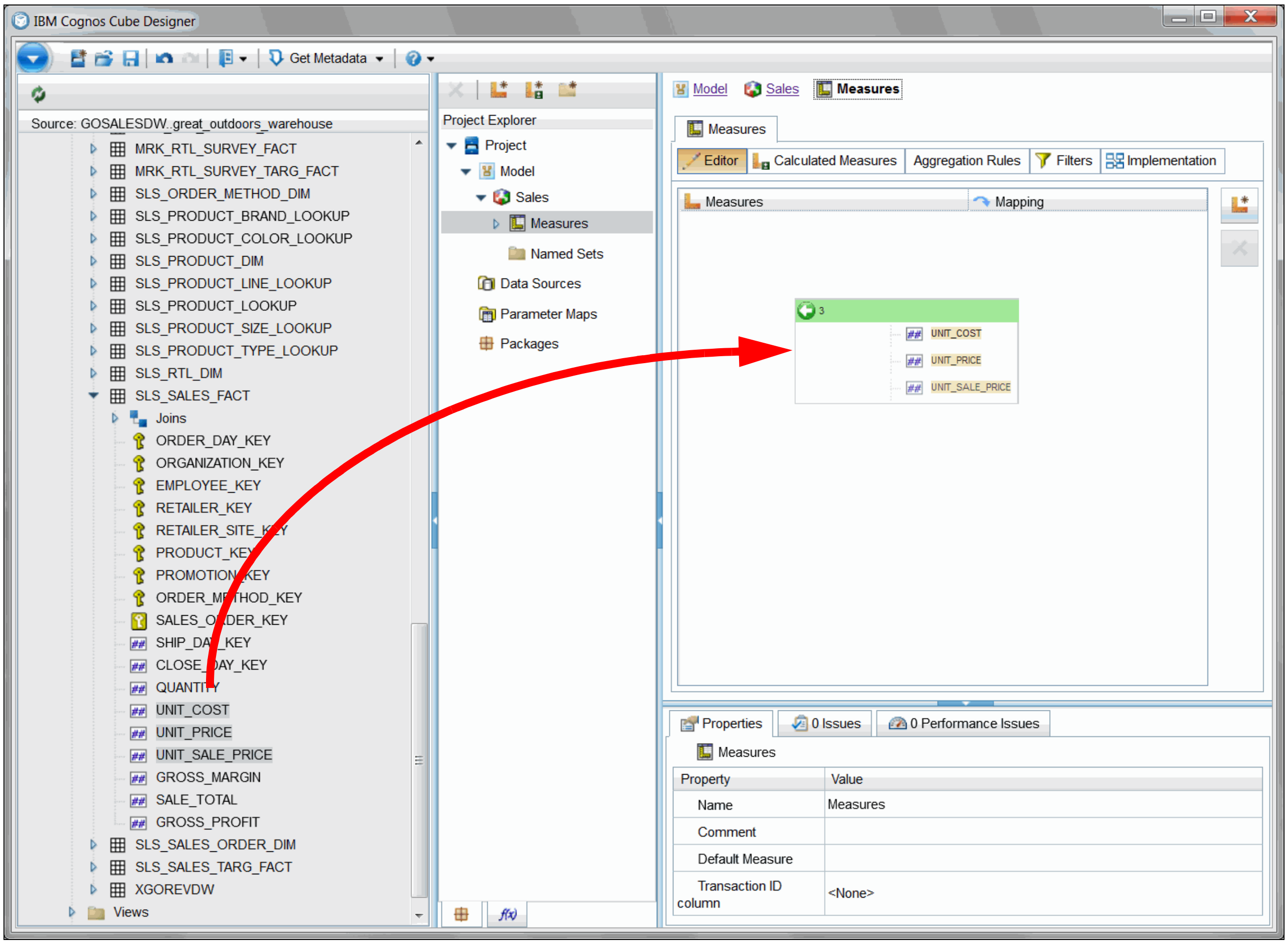


Figure 5-27 Dragging three columns from the SLS_SALES_FACT table to the measure dimension editor

When you create measures by dragging individual columns into the editor, the column names are used as the basis for the measure names. One measure is created for each column you drag into the editor and certain properties for the measure are assigned automatically. If you drag an entire table into the measure dimension editor, Cognos Cube Designer only creates measures based on columns that appear to contain fact data. Any column that is a primary key, foreign key, index, or has a non-numeric data type is ignored.

When you create measures, if the data type of the underlying column is numeric (for example, an integer, floating-point, or decimal), then the **Regular Aggregate** property of the measure is set to `Sum`. If the underlying data type of the column is not numeric, or if the column is a primary key, foreign key, or index, then the **Regular Aggregate** property of the measure is set to `Count Distinct`.

Three other properties are specified when you drag a column into the measure dimension editor: Data Type, Precision, and Scale. These properties are imported from the database itself and are read-only.

Regular Aggregate

The **Regular Aggregate** property is an important property in Cognos Dynamic Cubes. It specifies how the cube summarizes the detailed fact data in the fact table.For numeric data, sum is the most common regular aggregation and therefore is the default. However, you must consider your business requirements. Sometimes sum is inappropriate and you should change it to a more suitable aggregation.

The regular aggregation values Average, Count, Count Distinct, Maximum, Median, Minimum, Standard Deviation, Sum, and Variance simply indicate the mathematical operations that are performed on a set of measure values when they are aggregated.

Use the Custom regular aggregation to indicate to the cube that the value is computed by an external business process. In that case, the cube will always query the underlying database for fact values instead of computing them.

The Calculated regular aggregate value is used when a measure is based on an expression rather than mapped to a database column, and is used for governing the order of operations. The Calculated regular aggregate is discussed in more detail in 5.4.2, "Measures with expressions" on page 134. The Count Non Zero regular aggregate is special and is also discussed in that section.

5.4.2 Measures with expressions

Measures can also be used to compute values based on other measures. When you are working with measures in a dimensional context, measure values are frequently both, summarized (according to the Regular Aggregate) and computed (according to the expression).

Depending on the expression, a query can resolve to substantially different values depending on whether the values are summarized, and then the expression is computed or the detail values are computed and then summarized.

If the **Regular Aggregate** property is set to `Calculated`, then the individual components of the expression are aggregated according to their regular aggregate property and after that the expression is evaluated using the aggregated values.

If any other regular aggregate property is set, then the expression is applied to the individual component values, and the individual results are aggregated according to the regular aggregate you specified. For example, consider a cube with the data shown in Table 5-3, which shows the sample data for each quarter and the entire year.

Table 5-3 Sample data for each quarter and the entire year

Time	Sales (Sum)	Returns (Average)
Q1	10	2
Q2	30	4
Q3	60	6
Q4	120	8
2015	220	5

The values for 2015 are the aggregated values of each quarter in the year, according to the regular aggregate specified for that measure.

Assume an additional measure called Weighted Sales that is the Sales measure divided by Returns (this is not a meaningful calculation and is only used to illustrate the example). If the regular aggregate for the measure is set to `Sum`, then the value comes from calculating the expression on each row, and then applying the regular aggregate as follows:

1. Calculate: 10/2 = **5**, 30/4 = 15, 60/6 = **10** and 120/8 = **15**.
2. Aggregate: 5 + 15 + 10 + 15 = **65**.

However, if the regular aggregate property of a measure is `Calculated`, then the values are first aggregated, then calculated as follows:

1. Aggregate: 10 + 30 + 60 + 120 = **220**, average(2, 4, 6,8) = **5**.
2. Calculate: 220 / 5 = **44**

Count Non Zero behaves as though the measure (whether mapped to a column or containing an expression) is substituted for X into the following expression, with the regular aggregate set to sum as follows:

```
If(X = 0) THEN (0) ELSE (1)
```

If fact, you can even see this expression in the SQL statementthat is sent to the database. To see the SQL statement, you must enable logging.

5.4.3 Calculated measures

A calculated measure has most of the same properties as a regular measure with expression, including all of the same Regular Aggregate values. The key difference between a calculated measure and a measure with an expression is that the calculated measure takes a dimensional expression, rather than a relational one, and is always resolved by the dynamic cube itself. The calculation measure must resolve to a value. In general, a measure with an expression performs better than the equivalent calculated measure. However, there are a wider variety of expressions available for calculated measures, such as using relative time members (see 5.3.5, “Dimension filters” on page 127).

5.4.4 Measure folders

You can organize your measures and calculated measures in folders. The measure dimension can contain as many folders as you want, and individual folders can contain other folders. Measure folders are purely for organization purposes and putting a measure in a folder has no effect on the value of the measure.

5.4.5 Measure filters

You can apply data filters to the fact table that your measures are mapped to. Specify filters in the measure dimension, in the Filters tab, because filters apply to the fact table as a whole, not individual measures. In general, there are not many reasons to apply a filter directly to the measure dimension. A reason to apply a filter might be if the fact table has a status row and only rows with a certain status are valid. In this scenario, you could apply a filter and only use the valid data.

Measure filters do not have the **Exclude Facts Without Corresponding Dimension Keys** property. If you create a filter for your measures, it is important to understand your data and the implications of applying the filter, or you might get invalid results from the cube.

5.4.6 The default measure and transaction ID

The measure dimension has a few properties of interest. The Default Measure is the measure that is used for queries when the user has not chosen a measure. You can change the default measure to be any measure or calculated measure defined in the cube.

The Transaction ID column is part of the near real-time updates technology built into Cognos Dynamic Cubes. With near real-time updates, a dynamic cube can accommodate additions to the fact table without having to restart the cube. Use the **Transaction ID** column property to specify which column in the fact table is updated when new data is added. Cognos Dynamic Cubes can then monitor this column for additions to the fact table. For more information, see Chapter 12, “Near real-time updates” on page 427. A value of <None> disables near real-time updates for the cube.

5.5 Bringing dimensions and measures together in a cube

To restate, a cube is a data source that is defined by one or more measures interacting with one or more dimensions. There are different ways that you can specify and modify how the measures relate to dimensions.

5.5.1 Relationships

You must specify a *relationship* between the measure dimension and each participating dimension in the cube. Cognos Cube Designer is able to derive the relationship based on the keys in the dimension tables and fact table. However, if the data warehouse only uses views, you must specify the relationship yourself. You can see the list of dimensions that are used by a cube in the editor for the cube itself. Figure 5-28 shows the Editor for the `gosldw_sales` cube in the sample model.

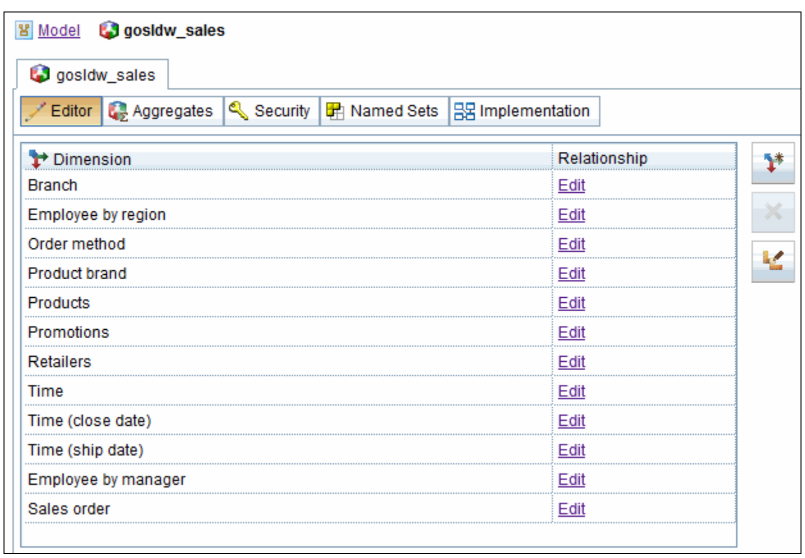


Figure 5-28 Relationship editor for the gosldw_sales cube in the sample model

You can remove a dimension from the cube by selecting the cube and clicking **Remove**. Removing a relationship removes the corresponding dimension from the cube.

You can edit the relationship to each dimension by clicking **Edit** in the Relationship column next to the dimension for which you want to edit the relationship. Clicking **Edit** starts the dimension relationship editor as shown in Figure 5-29. The figure shows the relationship editor for the relationship between the Branch dimension and the measure dimension in the `gosldw_sales` cube.

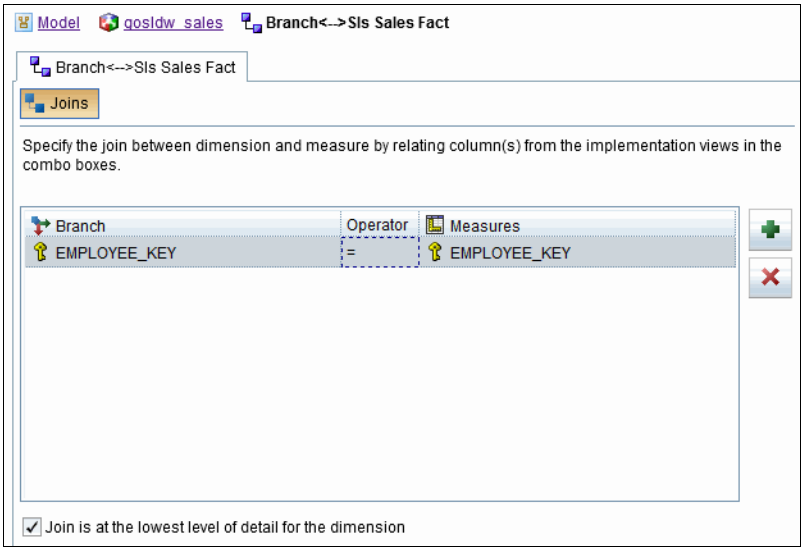


Figure 5-29 Relationship editor for the relationship between the Branch and measure dimensions

The functionality of the relationship editor is limited compared to the functionality of the intra-dimension join editor described in 5.3.4, “Joins” on page 123. You can add or remove key columns and specify the operator that is used to join the columns. The cardinality of all joins is assumed to be one-to-many.

If the data in the data warehouse does not correspond to a one-to-many join from the dimension to the measure dimension, you must apply the appropriate filters to either the dimension or the measure dimension. See 5.3.5, “Dimension filters” on page 127 and 5.4.5, “Measure filters” on page 135 for details about specifying data filters.

If you created a dimension that is shared between multiple cubes, the grain of the fact data can be at different levels for different cubes. For example, in the sample model, the Time dimension is defined down to the Day level. The data in the fact table for the `gosldw_sales` cube is also specified at the Day level. However, the data in the fact table for the `gosldw_target` is only defined at the Month level of the Time dimension. In cases like this one, clear **Join is at the lowest level of detail for the dimension** for the relationship. When this check box is cleared, Cognos Cube Designer generates additional model information to ensure that the dimension is used correctly in the context of the corresponding fact table.

If you are familiar with Framework Manager, you might have worked with measures that have different grains of detail. You handle different grains of detail by specifying measure scope for the measures. Cognos Dynamic Cubes does not support measure scope. Therefore, you must ensure that every fact within a fact table in the data warehouse has the same grain of detail for all dimensions you will use it with. For more information, see 6.5, “Multiple fact and multiple fact grain scenarios” on page 181.

5.5.2 Aggregation rules

In general, measures are aggregated according to the **Regular Aggregate** property of the measure. However, there are situations where you might want a measure to have different aggregation rules according to the dimension that is in scope.

For example, consider an application for tracking inventory levels. There is a measure for the specific inventory level, and dimensions for products that the inventory levels track, and geography (the inventory level in different regions). For these dimensions, setting **Sum** as the regular aggregate gives you useful values when reporting on those dimensions. However, using a time dimension in a query gives grossly misleading values. The inventory level for a month is *not* the sum of inventory levels for all days in the month.

In situations like this one, you can specify aggregation rules to use when particular dimensions are in context for a query. It is not necessary to define a dimension as a time dimension (such as in 5.3.5, “Dimension filters” on page 127) to specify aggregation rules.

You can specify aggregation rules for any dimension in a cube using the following steps, which use a time dimension as an example:

1. Double-click the **measure dimension** in a cube in the Project Explorer. The measure dimension is usually called *Measures*, but can have a different name.
2. Select the Aggregation Rules tab in the editor for the measure dimension.
3. Select a measure in the Measures list on the left and select the check box in the **Include** column for each dimension you want to specify an aggregation rule.
4. For each dimension, select the dimension in the dimension list and then select the aggregation in the **Aggregation Rule** drop-down menu.

Figure 5-30 shows the Aggregation Rules tab for the measures in the `gosldw_sales` cube in the sample model. The Quantity measure is assigned Count aggregations for the Employee by region and Branch dimensions.

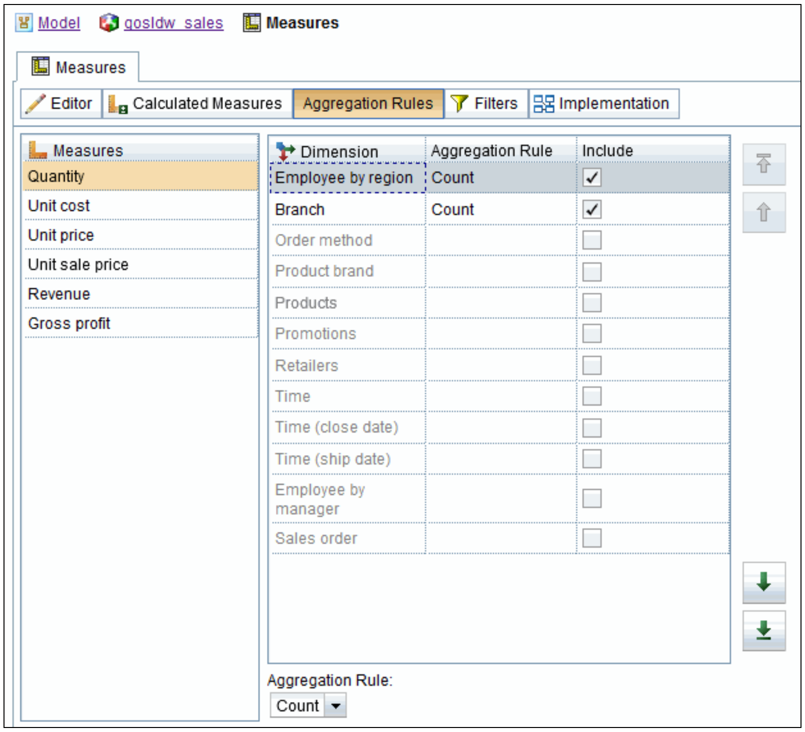


Figure 5-30 Quantity measure assigned the aggregation rule Count for the Employees by region and Branch dimensions

5.5.3 Named Sets

A named set is a dimensional expression that resolves to a set of members from a single hierarchy in one of the dimensions. A named set can reference any model object that is valid in a dimensional expression, including other named sets.

Named sets are checked for semantic correctness when the cube is started, but are run in their context within a report. If you refresh the member cache, then any changes to the members are represented in the evaluation of a named set (if there is an impact). The same named set can return a different set of members depending on where it appears within a report. For example, a TopCount of the top 10 products returns a different set of 10 products if nested within the regions of a geography dimension (a different set of 10 products for each region).

> **Tip:** The advantage of a model-defined named set versus a query-defined set is that it can be authored once and reused in different reports.

Named sets are defined in a folder called `Named Sets` in a cube. The `Named Sets` folder is always last in the list of objects under a cube in the Project Explorer after all the dimensions.

You can organize the named sets by creating subfolders under the root folder. When a dynamic cube is published, named sets are available as data items in the `Named Sets` folder within both the Metadata Tree and Members Tree in the IBM Cognos studios. In the studios, a named set can be inserted directly into a report, or it can be used in a dimensional function anywhere that a set expression is required.

Creating a named set

To create a named set, complete the following steps:

1. From the Project Explorer tree, expand the cube.
2. Right-click the **Named Sets** folder and then select **New** → **Named Set**. Rename the named set as wanted.
3. Double-click the named set to open the expression editor.
4. Enter a dimensional set expression.
5. Validate the expression. Syntax errors are displayed on the Issues tab.
6. Optional: Right-click the **Named Sets** folder and then click **New** → **Named Set Folder**. Organize named sets by renaming folders, creating additional folders, and dragging named sets from one folder to another.

Note: Expression validation checks for syntax errors only. It does not check semantic correctness. Semantic correctness is checked during cube start. Named sets are evaluated during cube start using the cube default context. Invalid named sets do not show up in the studios. They are dropped during cube start and an error is recorded in the `xqelog` file. Invalid named sets do not prevent the cube from starting.

Tip: For complex expressions, you might find it easier to author and test in report studio before creating in the model.

Use case: Creating named sets for analysis of products by revenue

Suppose that the report authors want to analyze the top products by revenue in different countries in their reports. You can provide a named set that contains the top products that report authors can reuse in their reports, without having to reauthor the expression.

Perform the following steps:

1. Create a named set `Top N Products by Revenue`, where N is a prompt macro with a data type of integer and a default value of five (5).

   Figure 5-31 shows the named set in the Project Explorer and the following expression:

   ```
   topcount([Model].[Products].[Products].[Product], #prompt('Enter N: Top
   N','integer','5')#, [gosldw_sales].[Measures].[Revenue]).
   ```

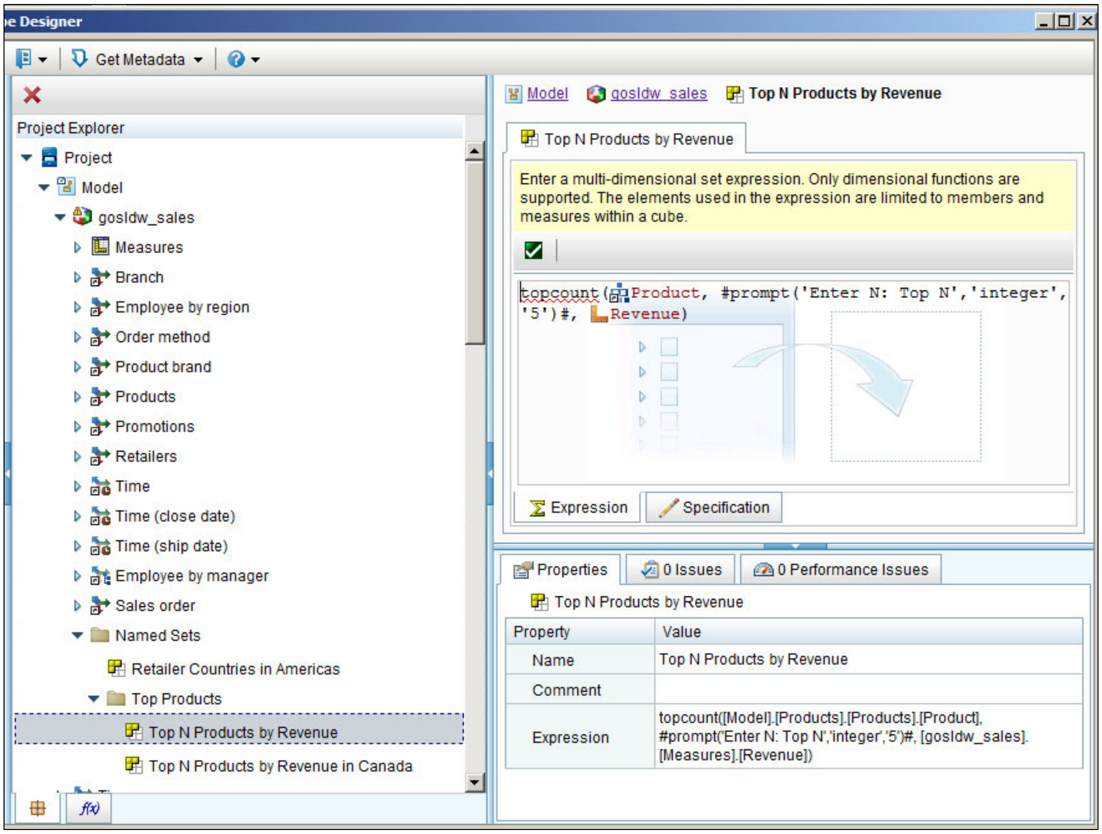


Figure 5-31 Named set Top N Products by Revenue

2. Create the named set `Top N Products by Revenue in Canada`, where N is a prompt macro with a data type of integer and a default value of five (5), and Canada is a member from the Retailer country level of the Retailers hierarchy.

Figure 5-32 shows the named set in the Project Explorer and the following expression:

```
topcount([Model].[Products].[Products].[Product], #prompt('Enter N: Top
N','integer','5')#, tuple([gosldw_sales].[Measures].[Revenue],Canada))
```

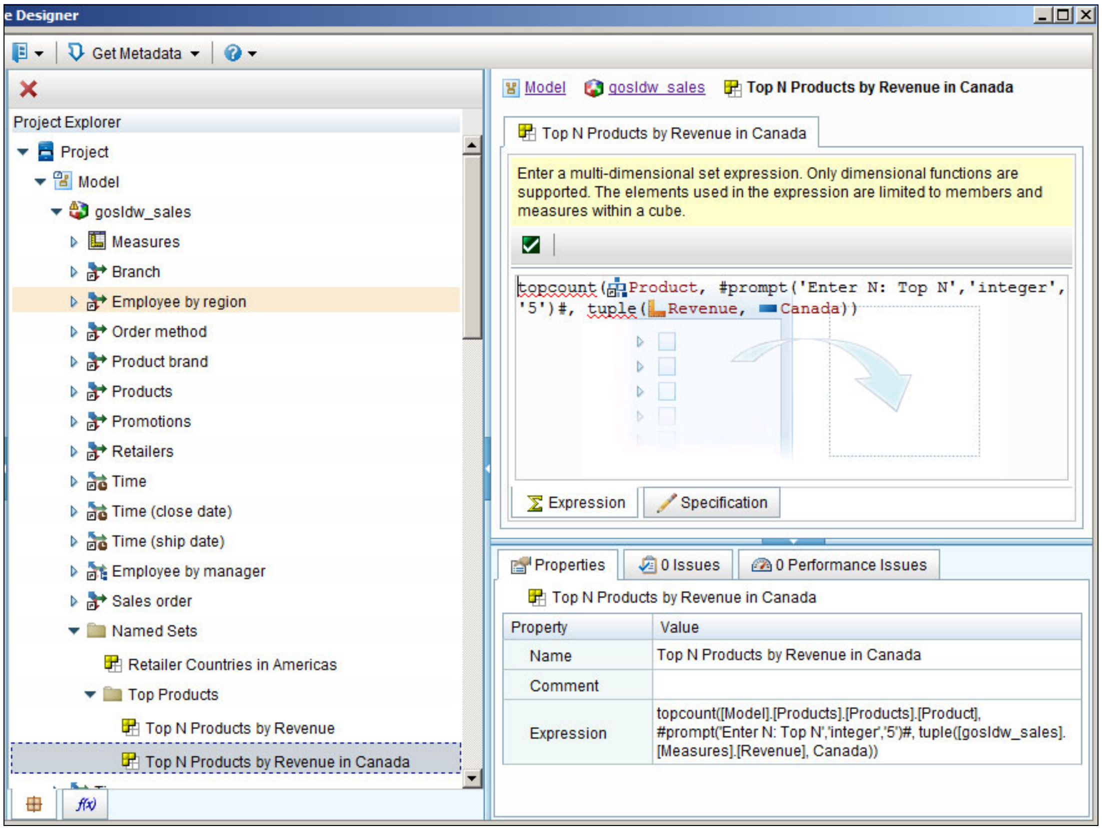


Figure 5-32 Named set Top N Products by Revenue in Canada

3. Create the named set `Retailer Countries in Americas` containing all countries in the region Americas from the Retailers hierarchy.

   Figure 5-33 shows the named set defined as: `children(Americas)`.

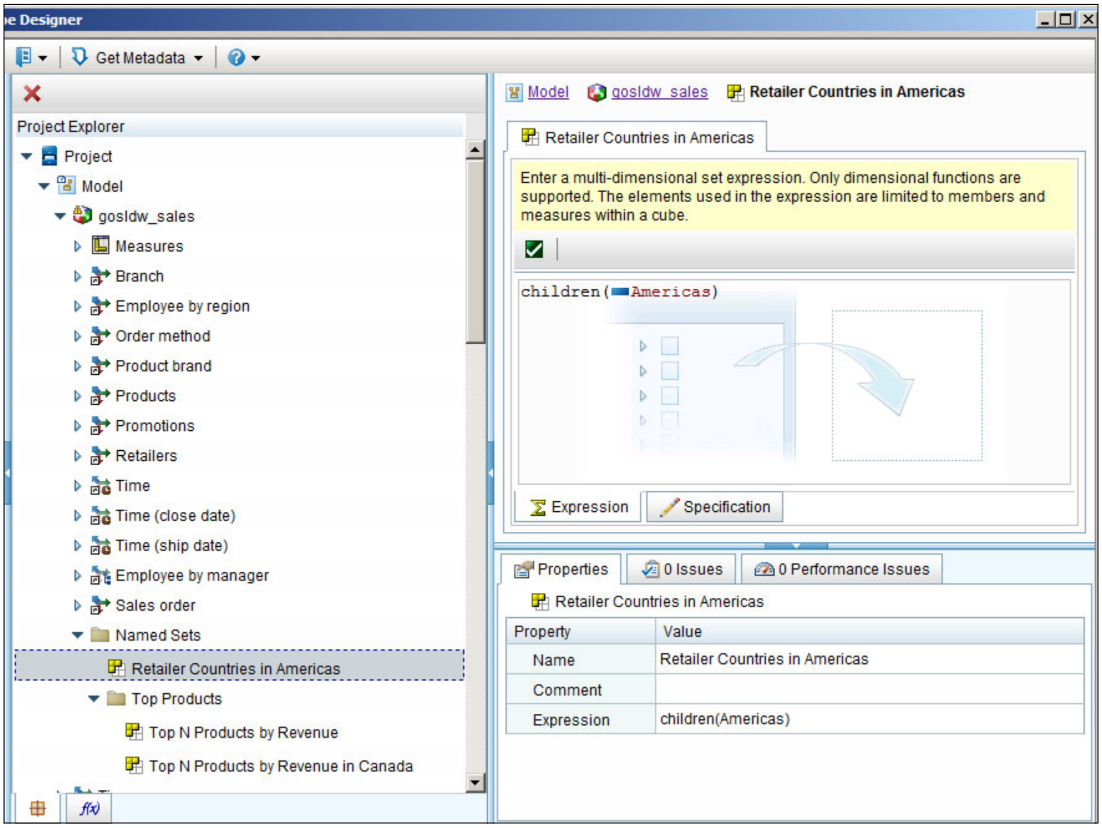


Figure 5-33 Named set Retailer Countries in Americas

4. Create calculated member `Aggregate of Top N Products by Revenue` in the Products hierarchy that aggregates the current measure for the `Top N Products by Revenue` named set.

   Figure 5-34 shows the calculated member with the following expression:

   ```
   aggregate(currentMeasure within set Top N Products by Revenue)
   ```

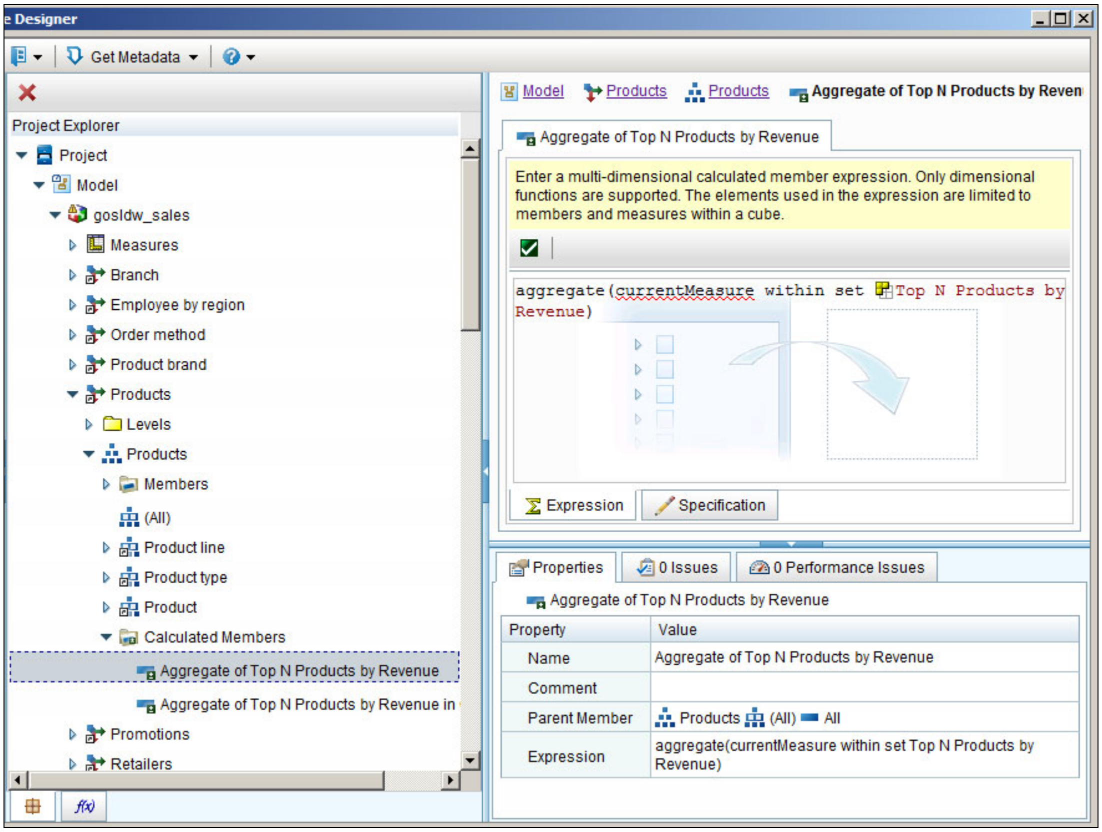


Figure 5-34 Calculated member Aggregate of Top N Products by Revenue

5. Create calculated member `Aggregate of Top N Products by Revenue in Canada` in the Products hierarchy. The expression locks the Retailer context to Canada so that the aggregate for Canada can be compared to the aggregate of other countries.

   Figure 5-35 shows the calculated member expression:

   ```
   tuple(member(aggregate(currentMeasure within set Top N Products by Revenue in
   Canada)), Canada).
   ```

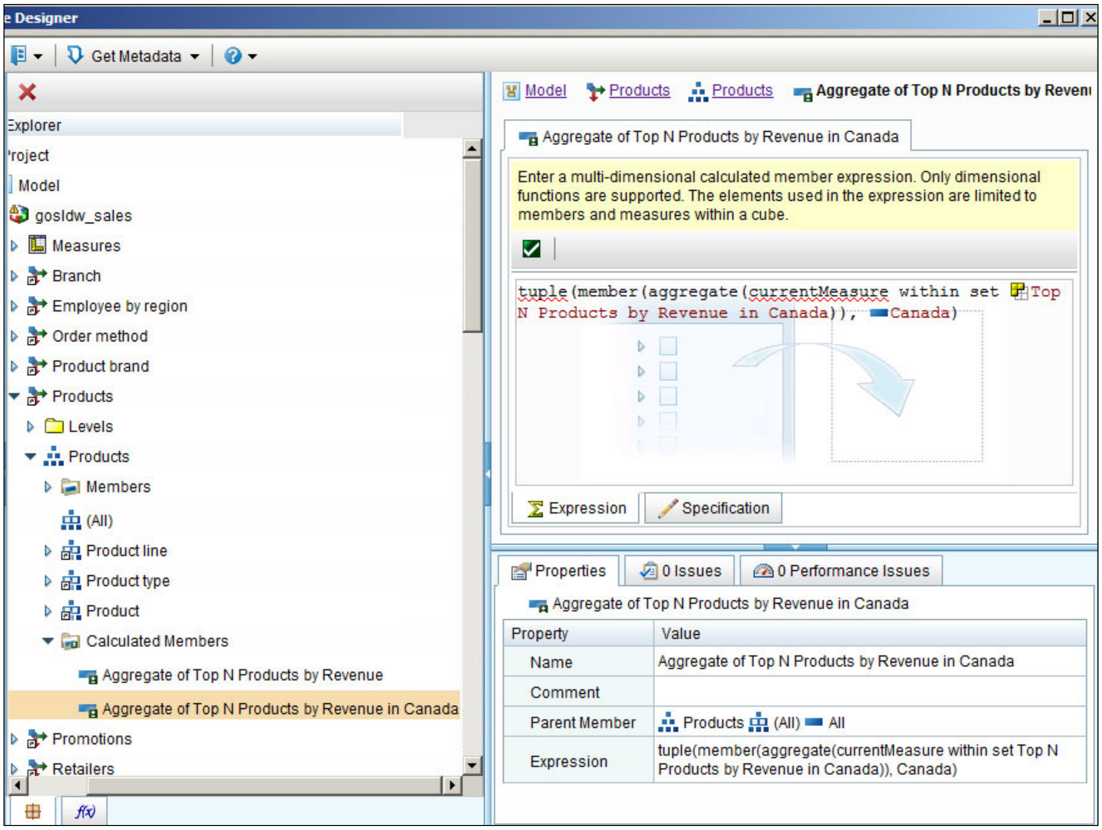


Figure 5-35 Calculated member Aggregate of Top N Products by Revenue in Canada

6. Publish and start the cube.

7. Open the package in Workspace Advanced, and create a crosstab.

   Figure 5-36 shows the Page Design view of the finished report in Workspace Advanced.

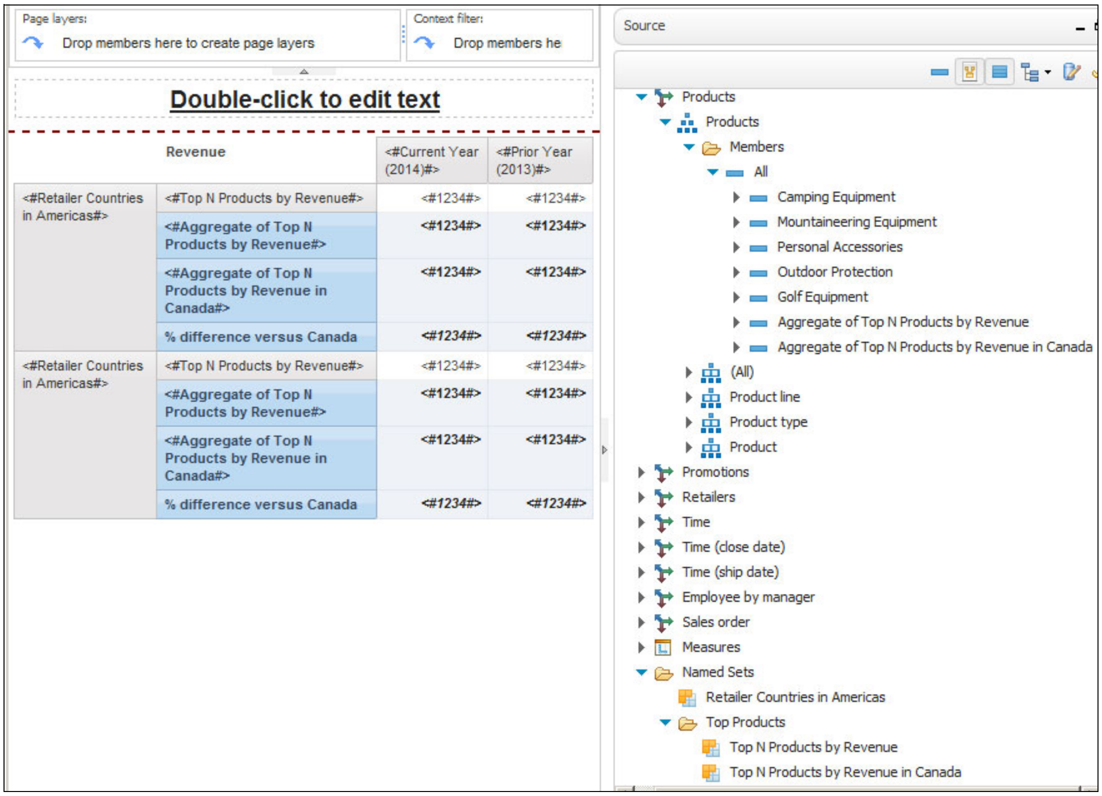


Figure 5-36 Report Page Design view in Workspace Advanced.

8. Show the top N products by revenue for different countries in the Americas by dragging named set **Retailer Countries in Americas** to the row, and nesting named set **Top N Products by Revenue** under it.

9. Drag calculated members **Aggregate of Top N Products by Revenue** and **Aggregate of Top N Products by Revenue in Canada** as siblings of **Top N Products by Revenue**.

   Aggregate of Top N Products by Revenue is a summary of the **Top N Products by Revenue** named set. **Aggregate of Top N Products by Revenue in Canada** is a summary of the **Top N Products by Revenue in Canada** named set, regardless of which country it is rendered under.

10. Create a query calculated member `% difference versus Canada` in the Products hierarchy, formatted as a percent that compares the top products revenue in Canada to other countries. Expression:

```
([Aggregate of Top N Products by Revenue] - [Aggregate of Top N Products by
Revenue in Canada] ) / [Aggregate of Top N Products by Revenue in Canada]
```

11. Drag in **Revenue** as the measure, and drag the built-in relative time members **Current Year** and **Prior Year** into the column.

12. Run the report.

13. At the prompt, click **OK** to use the default value of five (5), or enter a different integer value to change the size of the top products set.

Note: The two named sets that use the prompt macro use the same prompt name `Enter N: Top N`. As a result, only one prompt is generated when the report is run, and the value entered is used for both named sets.

Figure 5-37 shows the report output.

Revenue		Current Year (2014)	Prior Year (2013)
United States	Zone	5,641,953.50	10,117,895.60
	Star Lite	6,018,931.09	8,445,996.59
	Star Gazer 2	5,367,563.30	7,638,306.50
	TX	3,532,022.10	6,095,986.50
	Inferno	6,032,958.50	7,984,393.10
	Aggregate of Top N Products by Revenue	**28,245,642.35**	**41,952,566.19**
	Aggregate of Top N Products by Revenue in Canada	**10,588,389.93**	**14,753,560.80**
	% difference versus Canada	***167%***	***184%***
Canada	Star Lite	2,586,981.92	3,292,608.73
	Zone	1,822,064.70	3,199,829.45
	Star Gazer 2	2,388,042.80	3,034,297.20
	TX	1,238,638.60	2,272,017.20
	Hailstorm Titanium Woods Set	2,021,950.11	2,485,540.02
	Aggregate of Top N Products by Revenue	**10,588,389.93**	**14,753,560.80**
	Aggregate of Top N Products by Revenue in Canada	**10,588,389.93**	**14,753,560.80**
	% difference versus Canada	***0%***	***0%***
Mexico	Star Lite	1,631,908.24	2,396,737.13
	Star Gazer 2	1,389,336.30	2,072,661.80
	Hailstorm Titanium Woods Set	1,325,319.41	1,782,121.53
	Canyon Mule Journey Backpack	948,720.78	1,605,596.03
	Star Dome	867,152.95	1,319,850.32
	Aggregate of Top N Products by Revenue	**6,224,507.15**	**9,328,551.77**
	Aggregate of Top N Products by Revenue in Canada	**10,588,389.93**	**14,753,560.80**
	% difference versus Canada	***-41%***	***-37%***

Figure 5-37 Report output for the first three countries

Observations:

- The outer named set on the row, **Retailer Countries in Americas**, is a static set. The expression is independent of context and produces the same set of members regardless of placement in the report.
- The inner nested named set, **Top N Products by Revenue**, is a dynamic set. The Topcount is context-sensitive and can resolve to a different set of products depending on the context provided by the outer nested item.

Note: Current Year and Prior Year on the column do not affect the evaluation of the named sets on the row. Data items that are outer nested, or in the context filter, provide context for inner nested items. The default member of unprojected hierarchies is implicitly in context.

- The value of calculated member **Aggregate of Top N Products by Revenue in Canada** is independent of the nested country because the expression explicitly locks the context to member **Canada** in the **Retailers** hierarchy. Locking the context also allows for comparison of the summarized value for Canada to the other countries.

Security

Named set objects cannot be secured. However, member security affects the set. At report execution time, the combined security views of the logged in user are used to compute the named set. Security is applied before the set is computed. If the named set is a topcount, the topcount is based on the set of accessible members for the specified user.

At cube start time, named sets are evaluated using the default member from each hierarchy without regard to security. At cube start time, named sets that resolve to an empty set and empty named set folders are removed and not displayed in the studios.

Using the preceding "Use case: Creating named sets for analysis of products by revenue" on page 140 as an example, a security filter is defined on the **Products** hierarchy to deny access to all product lines except **Golf Equipment**. The filter is added to a security view and user `scarter` is assigned to the view.

Figure 5-38 shows the **Products** hierarchy member tree for user `scarter`. Only the product line **Golf Equipment** is accessible.

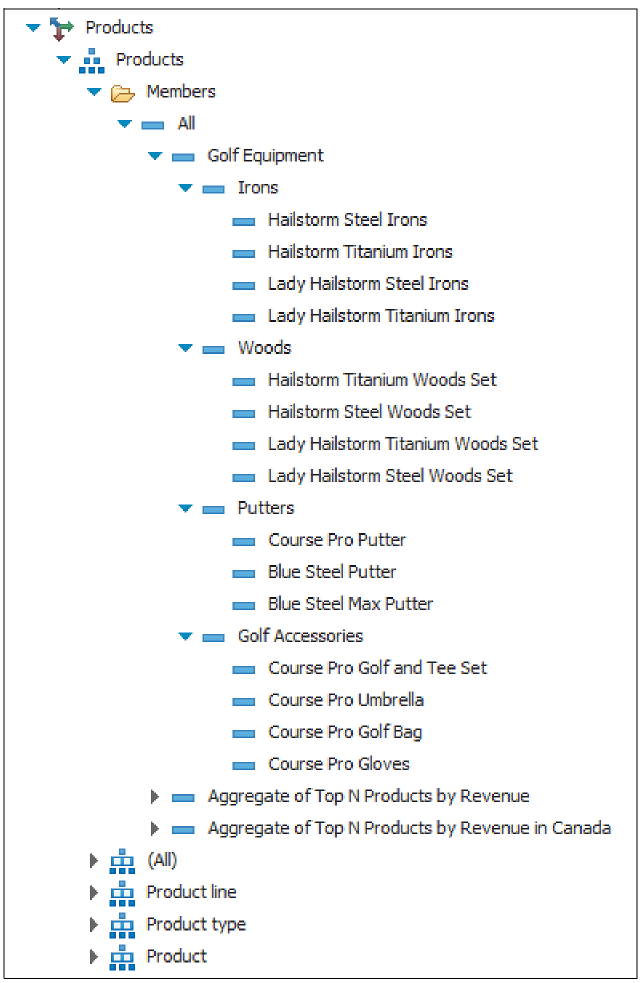


Figure 5-38 Products hierarchy member tree for user scarter

Figure 5-39 shows a partial report output for a report run by user `scarter`. Note the **Top N Products by Revenue** named set only contains product members that are descendants of **Golf Equipment**.

Revenue		Current Year (2014)	Prior Year (2013)
United States	Hailstorm Titanium Woods Set	5,184,235.96	6,232,767.70
	Hailstorm Titanium Irons	3,889,019.74	4,858,977.23
	Hailstorm Steel Woods Set	3,092,989.20	3,748,057.44
	Lady Hailstorm Titanium Woods Set	3,043,315.88	3,754,502.32
	Lady Hailstorm Titanium Irons	2,282,872.71	3,091,161.74
	Aggregate of Top N Products by Revenue	**17,492,433.49**	**21,685,466.43**
	Aggregate of Top N Products by Revenue in Canada	**6,930,235.12**	**8,780,072.79**
	% difference versus Canada	***152%***	***147%***
Canada	Hailstorm Titanium Woods Set	2,021,950.11	2,485,540.02
	Hailstorm Titanium Irons	1,490,005.88	1,926,970.38
	Lady Hailstorm Titanium Woods Set	1,252,978.10	1,560,409.84
	Hailstorm Steel Woods Set	1,220,227.20	1,510,737.60
	Lady Hailstorm Titanium Irons	945,073.83	1,296,414.95
	Aggregate of Top N Products by Revenue	**6,930,235.12**	**8,780,072.79**
	Aggregate of Top N Products by Revenue in Canada	**6,930,235.12**	**8,780,072.79**
	% difference versus Canada	***0%***	***0%***

Figure 5-39 Partial report output for a report run by user scarter

5.5.4 Estimating hardware requirements

When your dynamic cube is reasonably close to being ready for deployment, Cognos Cube Designer can estimate the amount of memory, number of CPU cores, and amount of hard disk space needed for your cube. Right-click the cube and select **Estimate Hardware Requirements**.

Figure 5-40 shows the Estimate Hardware Requirements window.

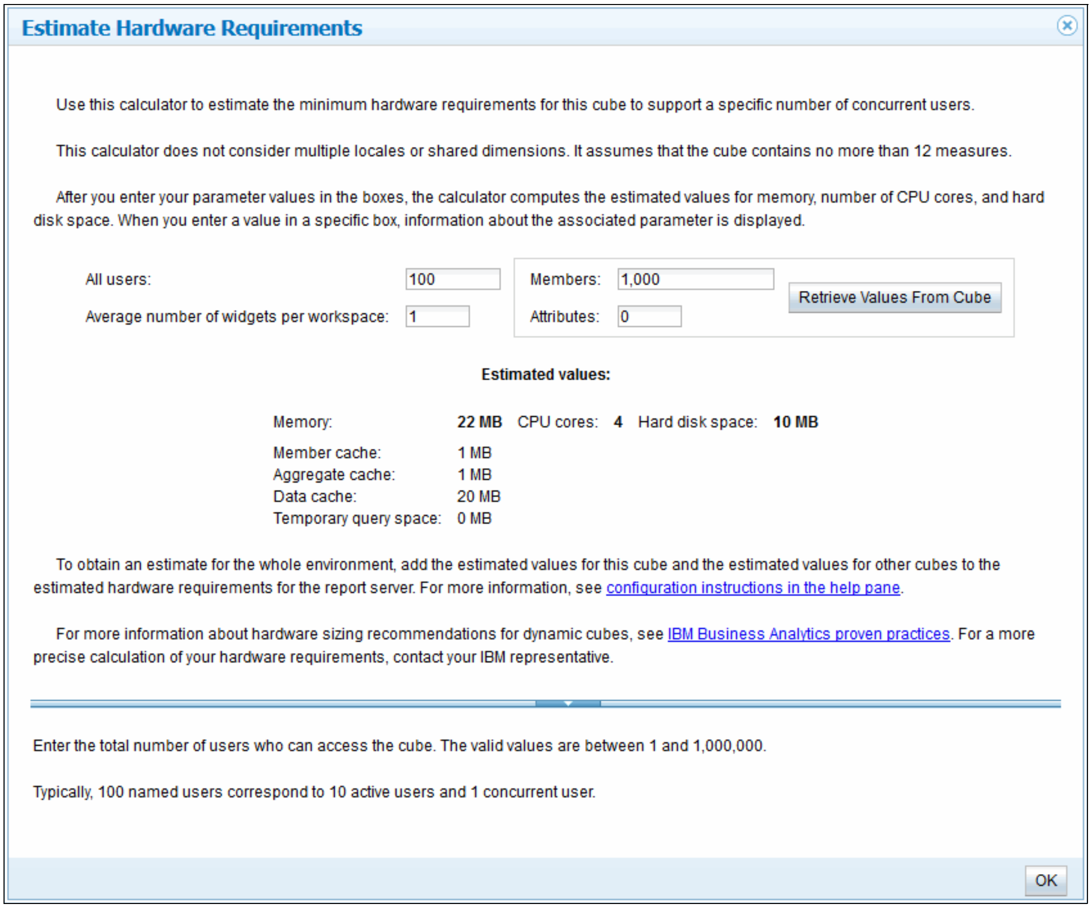


Figure 5-40 Estimate Hardware Requirements page

You must provide estimates in the **All users** and **Average number of widgets per workspace** fields. Click **Retrieve Values From Cube** to retrieve values for **Members** and **Attributes** from the cube itself. These values should be treated as a starting point, and optimizing a dynamic cube can be a complex process. See Chapter 11, “Optimization and performance tuning” on page 341 for more information about optimizing your dynamic cube.

5.6 Parameter maps

Parameter maps substitute keys in relational expressions with corresponding values. Parameter maps are useful if certain characteristics of your cube are defined by external tables or data sources. You can define parameter maps manually entering key-value pairs, by importing a comma-separated value (CSV) file into the model, or by identifying columns in a database to retrieve key and value pairs from.

You can create parameter maps by clicking the **Parameter Maps** folder in the model and either clicking the appropriate toolbar button or selecting the appropriate item from the menu, as shown in Figure 5-41.

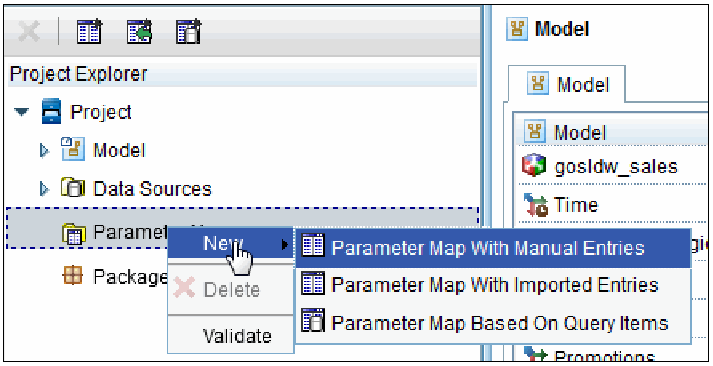


Figure 5-41 Toolbar and menu items for creating parameter maps

You can manually add more key-value pairs to a parameter map after importing the values from a CSV file. You can export a manually entered parameter map as a CSV file, edit it in another application and then import it back into the parameter map. The imported key-value pairs replace any existing entries in the parameter map. Figure 5-42 shows a parameter map with several entries.

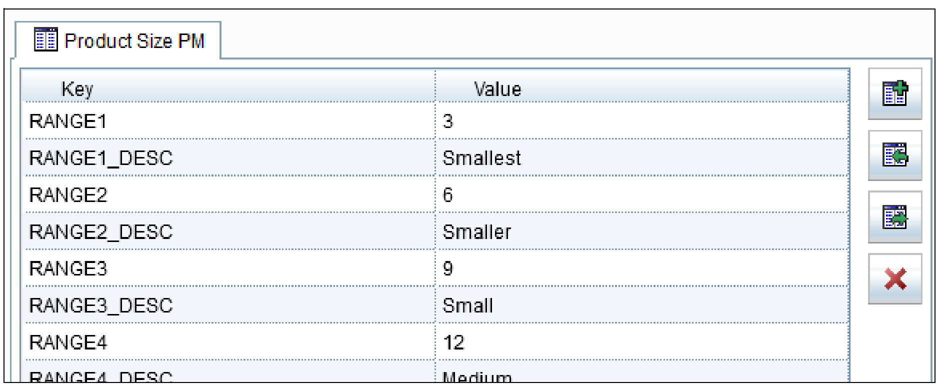


Figure 5-42 A parameter map with several entries.

Parameter maps based on query items provide a rich interface for querying key-value pairs from a table. You can even use expressions or provide filters if necessary. Figure 5-43 shows a parameter map based on query items. The key values are mapped directly to an EMPLOYEE_KEY column and the values are taken from a query item with an expression.

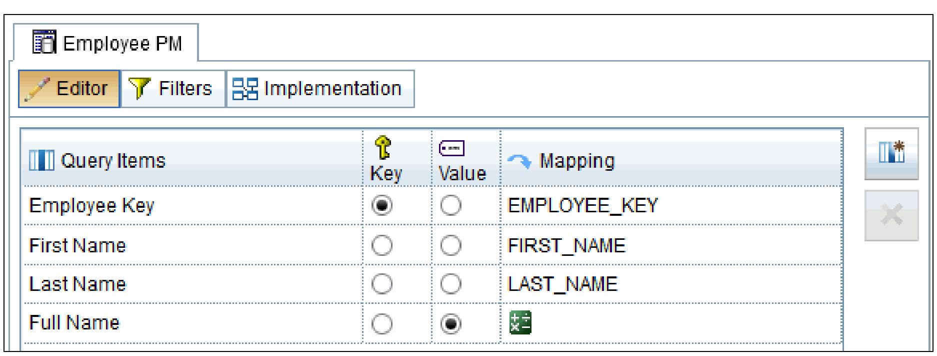


Figure 5-43 Parameter map based on query items

You can test a parameter map based on query items by right-clicking the parameter map and selecting **View Data**.

Parameter maps are evaluated during cube start and evaluated in the context of the user account you assign to it when you publish the cube. If you are familiar with Framework Manager, you might have used parameter maps with macros to enable multilingual applications by creating expressions that use different attributes depending on the locale of the user. For more information about creating multilingual applications with dynamic cubes, see 5.3.6, "Multilingual Attributes" on page 131. The method for modeling multilingual attributes in a dynamic cube is different from the method for modeling multilingual attributes in a Framework Manager model.

If you do not supply a default value for a parameter map and a key is encountered that does not have a value associated with it, a null value is used in the relational query to the underlying database. The null value might produce query errors or it might cause incorrect data to be returned.

The following example shows how to create an attribute hierarchy in a Products dimension based on the size of each product using parameter maps. Instead of having a single level with all products in it, it is helpful to organize the products into different groups based on their size. You can do that by creating an extra level above the products with a calculated attribute as shown in Figure 5-44, and specifying it as both the caption and level key attribute.

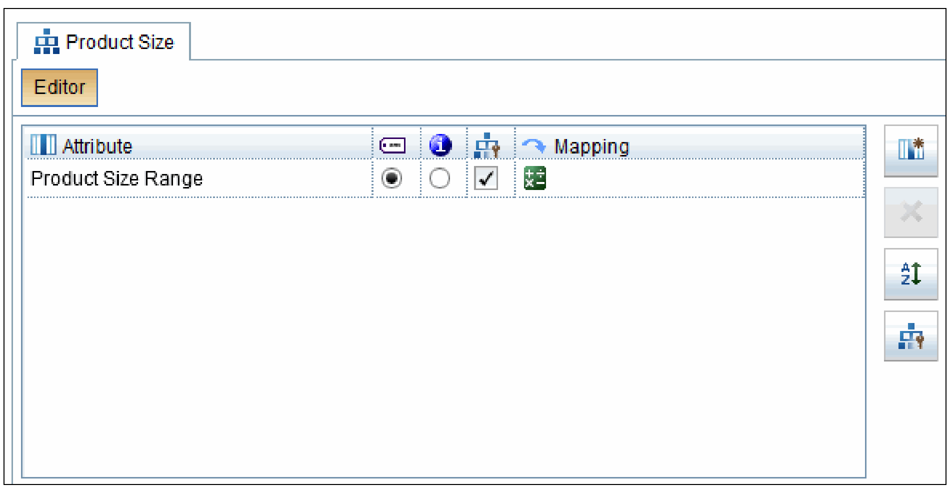


Figure 5-44 A level with a single attribute with an expression

Then, specify the Product Size Range expression as a series of case statements that reference both the product size values and the description (to use as the member caption) from the parameter map, as shown in Figure 5-45. Doing so allows you to define the attribute now, but modify the individual groups later without having to edit the model, by merely changing the values associated with the relevant keys in the parameter map. The product size range values (the values associated with RANGE1, RANGE2, and so on) and the caption used for those members (RANGE1_DESC, RANGE2_DESC, and so on) are defined in the parameter map.

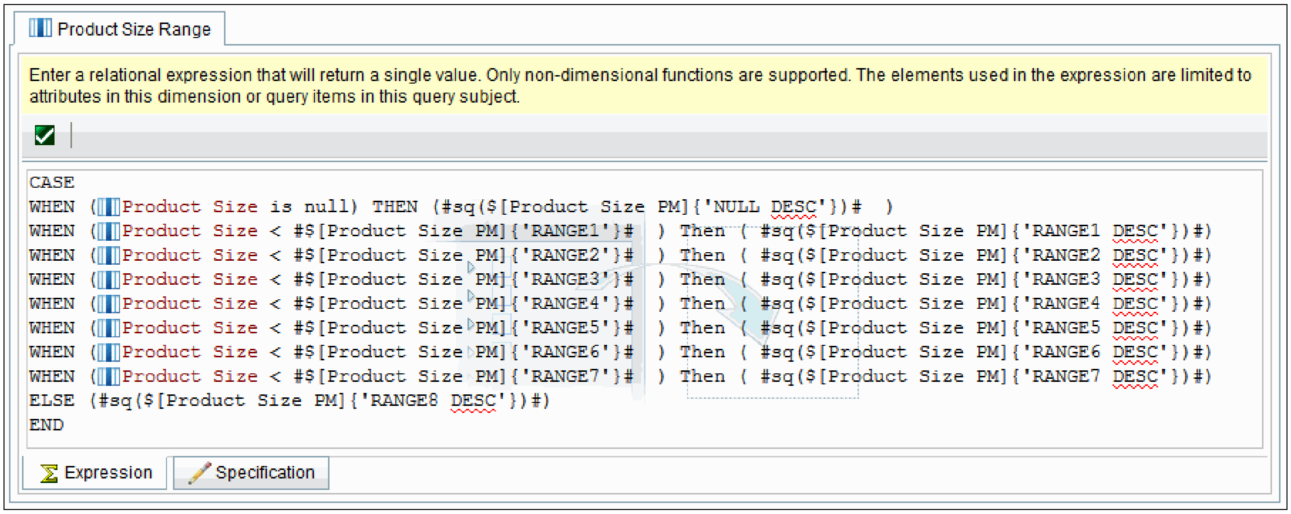


Figure 5-45 Expression for an attribute that references a parameter map for both the value ranges and the resulting text

Be aware that all parameter maps based on query items must have no errors in the Issues tab before you can browse the members of a dimension or publish a cube, even if the particular dimension or cube does not use any parameter maps.

5.7 Validation

Cognos Cube Designer analyzes the model to check for modeling errors. The analysis comprises validation checks for model integrity, which attempts to assist the modeler to follow modeling best practices, and identify situations that can affect cube performance.

The results of model validation are displayed in the Issues tab in the Details window. You can click each issue to see an expanded description of the problem and, depending on the issue, a list of possible solutions to the problem. Issues are organized according to their severity. Errors always appear at the top of the list of issues. All errors that are identified in a cube must be resolved before you can publish the cube. Warning messages appear lower in the list of issues, below errors messages. You should consider each warning message, but they might not necessarily apply to your application and can be ignored if they do not apply.

Some examples of the model integrity validation checks include identifying if levels do not have attributes in them, identifying if an attribute does not have an expression or object defining it, and identifying if a level has had the required roles defined. Other tests include checking to see whether the relationship between a dimension and the measure dimension is defined. If there is role-playing between a dimension and a fact table, there might be more than one relationship in the database. Validation detects this state. You need to determine which relationship you want to use.

One test of modeling best practices is to ensure that the relationship cardinality flows down to the fact table. If an object that has a relationship to another object in the dimension goes against this flow, Cognos Cube Designer flags it as an issue.

An example of a warning message exists in the sample model. The cube `gosldw_sales` has an aggregate defined. There is a filter in the Time dimension. The warning message identifies this state. The combination of the two (aggregate defined and filter in a dimension) is not necessarily a problem. You need to verify that the dimension filter does not filter out data that exists in the source of the aggregate. If that was the case, then a query that is routed to the aggregate returns a result that differs from a query routed to the detail fact table. This warning can be confusing if you are unfamiliar with aggregates. For more information about aggregates, see 11.2, “Aggregate awareness in Cognos Dynamic Cubes” on page 342.

Another example of an issue with a severity of warning is a level that exists in a dimension but is not used by any hierarchy in the dimension. This situation does not prevent you from publishing a cube that uses this dimension. The level will simply not appear to the report authors. However, if the level is important to your application, then you should pay attention to the warning message and use it in a hierarchy.

Cognos Cube Designer also attempts to identify potential performance issues with the cube. Performance issues are listed in the Performance tab. They identify model and database issues that might cause slower queries or greater memory usage. You should consider the issues that are identified, but if they are not relevant for your application, you can ignore them.

5.8 Deploying dynamic cubes for reporting and analysis

This section explains how to quickly or manually deploy Cognos Dynamic Cubes, and work with multiple cubes in a package.

5.8.1 Quick-deploy options in Cognos Cube Designer

After you finish modeling, you can quickly publish a single cube to IBM Cognos Connection in the IBM Cognos BI server environment so that report authors and others can begin their reporting and analysis. To perform this task, right-click the cube that you want to deploy from the Project Explorer and select **Publish**.

By default, the Publish window publishes the cube model to the content store as a data source. All other tasks must be done by an administrator from IBM Cognos Administration.

Clicking **Additional Options** causes the window to expand to provide more options for deploying a cube, as shown in Figure 5-46.

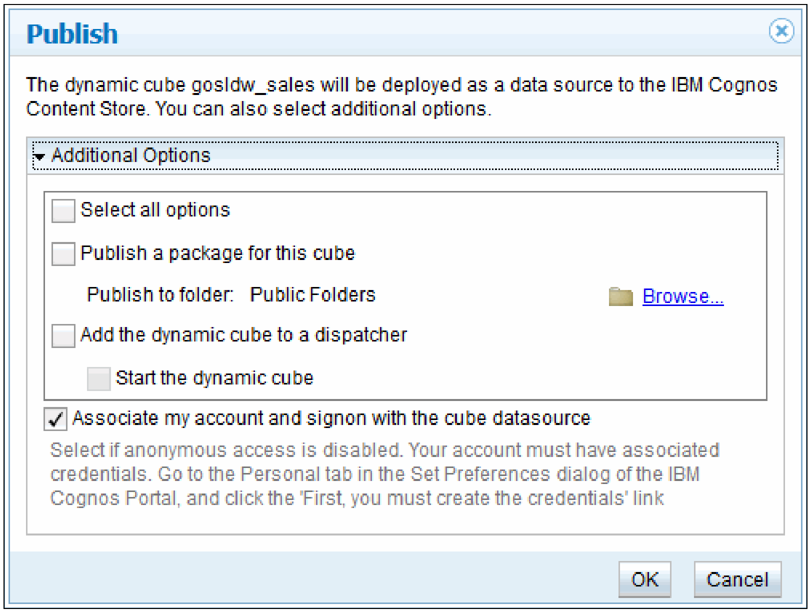


Figure 5-46 The Publish window with Additional Options expanded

The following options are displayed:

- Selecting **Publish a package for this cube** creates a package for the cube in the content store directory.
- The **Browse** link starts a window that allows you to select a different folder in the directory to which you want to publish the package of the cube.
- If selected, the **Add the dynamic cube to a dispatcher** option adds the cube to a dispatcher. This option is fine if you are working in a single-server environment such as a development server and want to test a cube. However, it can have unpredictable behavior in multiple-server environments.
- If **Start the dynamic cube** is selected, Cognos Cube Designer attempts to start the dynamic cube.

  Cognos Cube Designer waits for a short time for the cube to start before returning control of the interface to the user. If the cube has not started by then, Cognos Cube Designer shows an alert message. This message does not necessarily mean that anything is wrong. It simply means that you need to check Cognos Administration to verify that the cube has started.
- Click **Select all options** to select all options.
- If you click to clear **Associate my account and signon with the cube datasource**, you must manually associate the dynamic cube with another data source in Cognos Administration.

After this process completes, begin your reporting and analysis on the dynamic cube package. If you then want to make this dynamic cube package available to other report authors or users, copy the package and paste it into the location of your choice in `Public Folders` in Cognos Connection. The package contains a reference to the underlying cube and data source, so copying or moving the package does not affect the link between the package and the cube.

5.8.2 Manually deploying a dynamic cube

Another way of deploying your dynamic cube to is to manually deploy the cube from Cognos Cube Designer and create and publish a dynamic cube package using Cognos Framework Manager.

Although a number of steps are required to manually deploy the cube, this approach might be the correct approach to use if you are publishing to an environment other than a development or test environment. Acceptance testing or production environments are tightly controlled by administrators, so using the quick-deploy options might not be allowed.

The first required step is to publish the cube from Cognos Cube Designer without the additional options selected. This approach creates a cube data source in Cognos Connection and publishes the cube model to the Cognos content store.

The Publish page contains an option that must be set if your environment has security defined. Most environments, other than a sandbox environment, have defined some kind of security. The cube that you publish to Cognos Connection requires an account to be specified if security is defined. If you do not specify it, the cube does not start.

You can have Cognos Cube Designer associate your account with the cube by selecting A**ssociate my account and signon with the cube data source**. You must perform an extra step to generate credentials. To generate the credentials, click **First, you must create the credentials** in the Personal tab of the preferences window in IBM Cognos Connection.

If a cube of the same name exists in the portal, you are prompted to confirm that you want to continue to deploy the cube and overwrite the existing cube.

The remaining steps to deploy the dynamic cube are performed in IBM Cognos Administration in Cognos Connection, so you might need to work with your system administrator. Detailed steps to add a cube to a query service and start the cube are described in Chapter 7, “Administering dynamic cubes” on page 199.

After the cube is started, you then create a package in Framework Manager to publish to the content store to make the cube available for reporting and analysis. To perform this task, create a Framework Manager project and, when prompted for a data source to use, select the dynamic cube data source that was created during the publish step in Cognos Cube Designer.

The data source inherits the name of the cube you published. The cube is imported into Framework Manager as a cube object, so you cannot expand the cube to see the dimensions and levels in the cube.

Create a package and add the cube object, and then publish the cube to the location of your choice in Cognos Connection. This process is identical to creating a Framework Manager package using any other supported online analytical processing (OLAP) source such as IBM Cognos TM1 or IBM Cognos Transformer PowerCubes.

5.8.3 Working with packages

Cognos Cube Designer allows you to create packages with multiple cubes in it. This is useful for dashboard applications where the users include multiple charts in a single report that come from multiple data sources. Be aware that having multiple cubes in a package does not allow you to use them together in a single query. If you want to use multiple cubes in a single query, you must first merge them together into a virtual cube.

To create new packages, select the **Packages** object in the Project Explorer and click **New Package** on the toolbar.

To add cubes to the package, drag the cubes from the Project Explorer and drop them into the package editor.

Figure 5-47 shows a package with two cubes in it from the sample model.

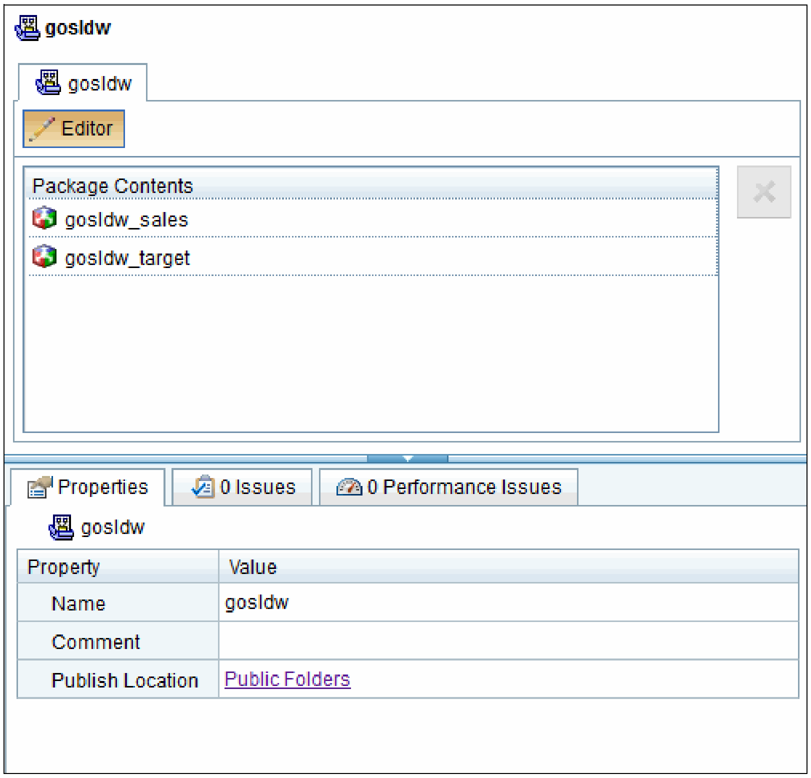


Figure 5-47 The gosldw package with two cubes

Specify to where you want to publish the cube in the **Publish Location** property of the package. Publishing a package is then a simple matter of right-clicking the package in the Project Explorer and selecting **Publish**.

5.9 Managing data sources

While developing a dynamic cube, you will occasionally need to change the data source that the model uses. A typical scenario is that you developed the cube using a database with a subset of data, and now you want to test it using a similar schema with a much larger set of data.

To change a data source, complete these steps:

1. Select the data source that you are interested in changing and change the appropriate properties.
2. Right-click the data source in the Project Explorer and select **Refresh metadata**.

Figure 5-48 shows how to update a data source in the model.

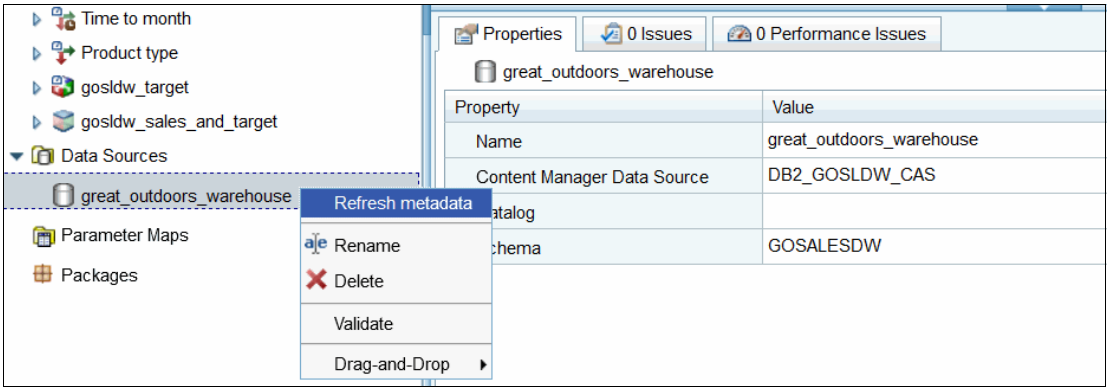


Figure 5-48 Updating a data source in your model

Cognos Cube Designer creates a new pane for the updated data source in the metadata and validates the model, verifying that each table and column reference is still valid. You will see an error in the Issues tab for each table or column reference that no longer exists in the new data source. Cognos Cube Designer also updates the data type, precision, and scale of any attribute or measure whose data type, precision and scale can be different in the new data source.

6

Advanced topics in modeling

This chapter describes how to solve common advanced modeling problems, identifies design considerations that a modeler needs to think about, shows how to troubleshoot models, and includes practice exercises.

This chapter contains the following sections:

- Model design considerations
- Time dimensions and relative time
- Slowly changing dimensions
- Role-playing dimensions
- Multiple fact and multiple fact grain scenarios
- Data quality and integrity
- Logical Modeling
- Modeling for null values
- Scenario dimensions
- Troubleshooting

6.1 Model design considerations

When you begin modeling, you need to make decisions about many aspects of the design of the model.

When designing your model, you must take into consideration the trade-offs between the reporting requirements, and the analytical possibilities and their resulting structures.

When you design a model, you also need to be aware of the nature of the project that you are undertaking. The modeling exercise itself is a small part of it. The success or failure of the project depends on the co-ordination of a multi-disciplinary team.

You will need to interact with other people to gather the requirements, establish alignment about how is responsible for which of the dynamic cubes roles, work out how the team operates including ensuring that the relevant parties actually meet and understand what the others are talking about. A simple example of that is the term cross-join. A person from a relational background tries to avoid relational cross-joins. MDX-oriented counterparts see cross-joins as a key part of their world.

A relational DBA might not know anything about online analytical processing (OLAP) or dimensions. A report author might not be familiar with OLAP constructs. A Cognos administrator might know nothing about managing an actual data source. Recommended database aggregates require database administrator (DBA) and extract, transform, and load (ETL) involvement. Table-based security filters require involvement of DBAs and security administrators. The requirements of defining what data any particular person is allowed to see and mapping that to the relevant Cognos Dynamic Cube security model can be a lengthy and painstaking process.

The process of developing your application can be hampered if you must open an IT support ticket every time you need to access information from a log file.

The project itself entails an ongoing process of feedback and refinement. This process occurs not just while the models are being developed, but also when the cubes are in production.

You will probably need to document requirements and justify capital expenditures for hardware. Knowing how to define trade-offs that allow for success within your physical limitations so that decision-makers can understand what they are choosing is important. For more information see 1.2, “Skill set requirements for IBM Cognos Dynamic Cubes” on page 7.

6.1.1 Cube design considerations

When you create a cube, you are designing a hybrid object. You are laying out both a model that generates relational queries and one that organizes their results in structures that you specify. It is the intersection of relational modeling and OLAP. You need to be comfortable in both worlds. If you understand what is being done, you will have a better grasp on your primary task of modeling to meet business needs.

Each dynamic cube is a star or snowflake schema that is centered around a single fact table. The grain of each of the facts in the fact table should be the same.

Some modeling actions, such as defining relationships, role-playing dimensions, and slowly changing dimensions, are tightly bound to relational query requirements. It is vital to understand the effect of what you have modeled and how to translate your modeling requirements into what you need to do to accomplish them.

You will not be successful with Cognos Dynamic Cubes unless your data sources are star or snowflake data warehouses. For more information see 1.3, "When to use Cognos Dynamic Cubes" on page 8.

6.1.2 Dimension design considerations

The design of a dimension necessarily requires dealing with many different types of constraints. The major ones are the reporting needs, hardware capacity, and usability.

When you, and possibly your customers, understand dimensional modeling and the concept of a hierarchical representation of your data, it is tempting to try to squeeze out as much analytical insight as possible by modeling as many hierarchies as you can discover. You might feel compelled to create a hierarchy for almost any attribute or set of related attributes.

For instance, you might create dimensions with many hierarchies, or many role-playing dimensions that either use different relationship keys or that exist to present a specific hierarchical view of a dimension.

This approach could proactively address the users needs but it could also be wasted effort and wasted resources. One problem is that there could also be physical constraints. These constraints exist not only in the cost and availability of hardware, but also due to the operating system itself.

Each hierarchy is included in the tuple defining the member. Members take up memory. If you have sufficient numbers of members in a hierarchy that itself is not absolutely necessary, you could be wasting resources. In addition, a sufficiently large number of members could exceed the number that can be stored in the OS memory and require more processing to generate them and to work with them. For more information see Chapter 17, "Hardware sizing" on page 533.

Some resource problems can be partly alleviated by the Share member cache for all cubes property as shown in Figure 6-1. When this property is set to `true`, Cognos Dynamic Cubes creates one member cache for the dimension. Each cube that has the dimension uses this cache rather than creating a new cache as new cubes are published.

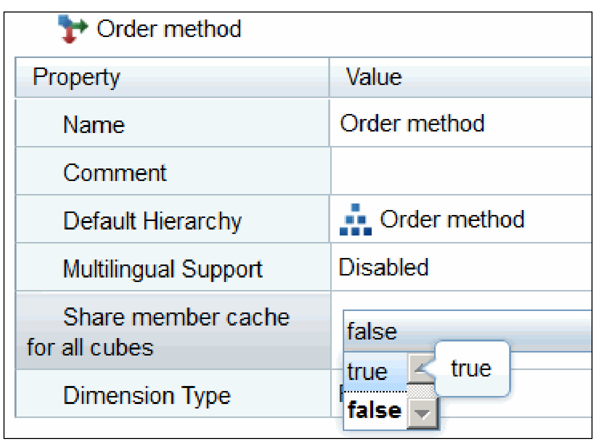


Figure 6-1 Share member cache for all cubes property

Any deviation from conformance causes a new member cache to be created, which means that you need to republish any cube that has the changed dimension to maintain the dimensional conformance.

When you include a level in more than one hierarchy a separate set of the members of the level is created, which takes up more memory. Reporting and analysis requirements drive the design. You must to be aware of those requirements. You must anticipate the questions that users will ask. A group of report consumers might need analytical information but might not have submitted their requirements to your organization or have not been able to articulate the requirement clearly due to a lack of understanding of the power of Cognos Dynamic Cubes.

You need to understand whether the requirements tend to be of managed reporting and analysis and data exploration or data discovery. Dynamic cubes are suited for the former and not the latter.

Another constraint to consider is the choice of dimension modeling style and query generation and reporting requirements. The choice is to either model many hierarchies within a dimension or to model many dimensions, each containing one hierarchy. You can create many hierarchies in a dimension.

One reason for modeling multiple hierarchies instead of multiple dimensions is to organize them together. You can take advantage of the ability of dynamic cubes to remove non-existent tuples from a report in which members from different hierarchies are nested on a report axis when convergent levels exist in those hierarchies. You can accomplish the same goal by modeling separate dimensions.

In some cases, the reason for modeling separate dimensions is to handle role-playing dimensions. Each dimension has a distinct relationship to the fact table. You need to create role-playing dimensions so that the relationship keys defining the relationship between the dimensions and the fact table are distinct for each role the dimension plays with the fact table. For more information about modeling for role-playing dimensions, see 6.4, “Role-playing dimensions” on page 180.

If you are taking advantage of dimension filters, you might need to create multiple dimensions instead of one dimension with multiple hierarchies if you must use different filters. This might be the case because you need to create comparative dimensions, for example year over year analysis.

Another design consideration is the choice between using attributes of levels to filter reports or creating attribute hierarchies to refine the member definition, and building in that filtering. Attributes take up memory just as members do. There is a performance trade-off in query response time if you choose to use the attributes to filter the query.

Having many separate dimensions representing essentially the same object could be confusing. A report consumer will not know the provenance of the report objects in a report. Report consumers and report authors might not understand why these objects exist.

6.1.3 Hierarchy design

Hierarchy design considerations are related to dimension design. Take into account the trade-off between the organizational and classification richness of hierarchies and member find ability. Take also into account the purpose of the reporting requirements. Understand the trade-offs between resource availability and reporting requirements.

There are three distinct major hierarchy types: Parent-child hierarchies, natural hierarchies, and attribute hierarchies.

Parent-child hierarchies are hierarchies where the hierarchical structure is defined in the data rather than in any set of levels. The data defines the member and the parent of the member. You create parent-child hierarchies in parent-child dimensions.

A natural hierarchy is a set of levels that are defined by a set of conceptually related database columns that contain various degrees of classification detail. These *attribute families* are sets of related database columns that define elements of a hierarchy. For example, a customer dimension table could have columns that contain information describing where the customer lives, the country, country subunits such as province or state, the city or town, and the actual address of the customer. Another example of a natural structure is a time dimension with a hierarchy of levels: Year, Quarter, Month, and Day.

One example of a natural hierarchy is Product dimension in the sample model. It has Product line, Product type, and Products. There are columns in the dimension table that define these levels of classification detail. In fact, there is sufficient detail for an additional level, Product detail, which consists of defining each product in further detail to model. The Retailers dimension is another example. It organizes the members of the dimension by geography.

A set of attributes that are related can provide a mechanism for organizing and presenting members in a fashion that corresponds to how your users see their world. For example, if you want to organize your customer dimension's hierarchy by geography, the attributes that store the geographical information can be used to generate the hierarchy.

A hierarchy level captures a grain of dimension abstraction. A series of levels in a hierarchy can provide a means for your users to classify the entities of the dimensions and ask questions that are vital.

Using those attributes enables the users to better understand their business and gain insights that they otherwise will not discover.

Hierarchies do not need to be created from sets of levels whose source attributes are strictly related in form. They can also be created from attributes that are conceptually unrelated but allow for a classification that lends itself for analysis.

The attribute hierarchy is a hierarchy that contains members that are generated from an attribute describing something about the members of the dimension. Some examples of attribute hierarchies include Product color, Product age, Customer sex, Customer marital status, Customer census class, and Retailer store type.

You define attribute hierarchies for use in reports to allow your users to refine queries by being nested with other report objects along an axis or use them as stand-alone report objects.

Depending on your reporting requirements, your design of the attribute hierarchy varies. If you want to report against more than one hierarchy in your report, you need to include the convergence level of the dimension in each attribute hierarchy in order for only those tuples that exist to be returned. You can achieve the same effect by turning on zero suppression in the report. Doing so could have a greater performance effect than designing your hierarchy to have the convergent level in it. Null suppression is performed after the query has been processed, and operates on the entire cross-join set of tuples specified by a crosstab report. The convergence level is the leftmost level whose key matches the leaf grain of the dimension table. An example of a convergence level is shown in Figure 6-3 on page 165.

For example, assume that you add an attribute hierarchy of Product color to the Product dimension. Assume that you create only one level in it with Product Color Code as the key. If you add the Product color level to a report where you have something from the existing products hierarchy, each nesting of a product color member has all of the members of the Product level that you used, even if no Product of that type was made of that particular color.

Normally, to remove the non-existent tuples from the report requires setting the zero suppression settings for that report. Alternatively, you can include the convergence level in the hierarchy. The non-existent tuples are removed automatically.

Each time that you reuse a level in a dimension, a new set of members of that level are created and added to memory, which further complicates your design decisions.

Many of the functions of attribute hierarchies can be achieved by the use of attributes, but your users might have difficulty finding them and understanding their use.

Another aspect to keep in mind is the organization of large numbers of members. There are limits to the number of child members that are displayed in the IBM Cognos studios. A wide member tree could be problematic. However, if you create levels to refine the organization of members, you could make them less difficult for your users to find.

One possible root cause of your users not being able to find members could be that the hierarchy is not organized in the way that they see the world.

Take into account that the hierarchy context of a member affects the query. Users might not understand the query that they have accidentally generated as a result.

Another aspect of a hierarchy design is the choice whether to have the hierarchy with an Allmember or with multiple root members, which are the members of the highest level in the hierarchy. Aesthetics aside, having an All member can help organize a hierarchy better. In addition, a report that has the members of the highest level of the hierarchy in it can be more durable if it uses an expression such as children, which get the children of the All member and automatically fetches new members as the hierarchy data changes.

Using the `Multiple root members` property enables you to model scenario dimensions in situations where it does not make sense to roll all member values up into a single member. Scenario dimensions are often used when budgeting and forecasting. The aggregation of the root members can be confusing. It is important to specify a default member in a scenario dimension. For more information about modeling scenario dimensions, see 6.9, “Scenario dimensions” on page 189.

Yet another aspect of hierarchical design is slow change. You need to know your data and the concept of slowly changing dimensions (SCD). You need to be able to recognize situations of slowly changing dimensions.

What you want to accomplish, what information you want to convey, and what hierarchy you are modeling affects whether you must consider slowly changing dimensions and what you need to do. For more information about modeling for slowly changing dimensions, see 6.3, “Slowly changing dimensions” on page 179.

A hierarchy does not need to have all the natural levels of a hierarchy in it. There can be gaps between levels and you can truncate the hierarchy to a level above the dimension grain.

You could have gaps in a hierarchy if you want to not present a particular level in a hierarchy. The trade-off is that there will be more child members of parent members of the level above the gap. An example of a hierarchy that has a gap in it is if you remove the product type level from the product dimension. (see Figure 6-2 on page 165) In this case, the product members are children of the members of the Product Line level rather than the Product Type level.

Another example is where a natural level has null values and you either do not want to handle null values or the method for modeling for it creates confusion because the captions of the padding member are the same as the captions of their parent. By deleting that level from the hierarchy or not modeling it at all, you do not have to worry about this issue.

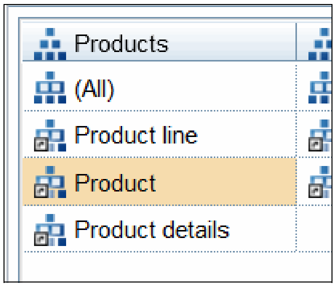


Figure 6-2 Hierarchy with a gap between natural levels.

An example of a truncated hierarchy is the Time to Month dimension in the sample model. The underlying dimension table, SLS_TIME_DIM, has columns for year, quarter, month, and day. The dimension grain is the lowest, that is, the least abstract or most specific, attribute of the dimension table. The concept of dimension grain is closely related to the concept of fact grain. They are the opposite sides of the coin. You might not want to go to the most detailed level of classification in a dimension. In the sample model, the Time to Month dimension exists to allow for modeling for the sales target cube, whereas the fact grain for time is month, without arbitrarily projecting month values to the day level members.

You can reuse levels in multiple hierarchies. In the case of attribute hierarchies, reuse them to define convergence levels in the hierarchies so that non-existent tuples are suppressed from reports with no need to enable the zero suppression report settings. For information about zero suppression report settings, see “Suppression” on page 422.

Figure 6-3 shows that the Product details level is used in multiple hierarchies as the convergence level.

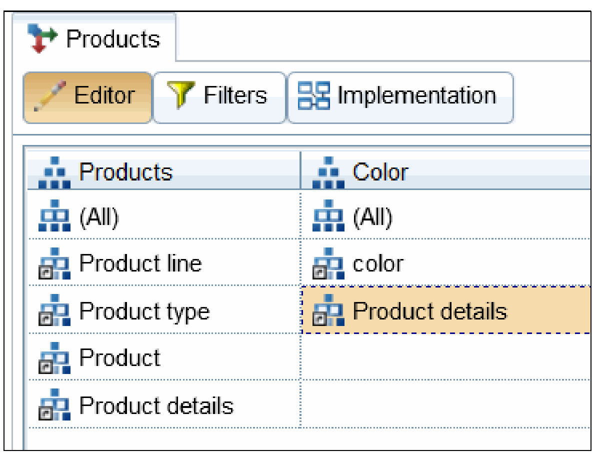


Figure 6-3 Attribute hierarchy with convergence level

6.1.4 Measure design

Designing measures is not just a matter of devising the expressions that you need. You must be able to control the timing of the aggregation of elements within your expression to control where computation takes place and you must know how to design measures to optimize performance. You should know about the trade-offs that are available to you. Your design choice is tied to aggregate awareness. Try to design measures so that the dynamic cube can use aggregate awareness.

A dynamic cube has two types of measures: Regular measures and calculated measures. Both types have the same available list of aggregation. They are Average, Calculated, Count, Count Distinct, Count Non Zero, Custom, Maximum, Median, Minimum, Standard Deviation, Sum, and Variance.

Knowing when to choose which measure type is important. Apart from knowing how to control aggregation and how the measures results will be generated, the measure types allow you to use both the underlying relational database and the dynamic cube.

Regular measures are pushed to the database to be computed. Calculated measures are computed locally, but allow for the exploitation of aggregate awareness by routing queries to aggregate tables. Apart from the computation location trade-off, the aggregation type of your measure determines the design of the measure.

You should be familiar with the following measure concepts:

- Regular Aggregate of Custom. The Custom value indicates that the value of the measure is computed by an external business process. Custom measures are a specialized form of non-distributive measure that do not roll up. Custom aggregates are used to store values for complex, pre-computed measures based on business rules that Cognos Dynamic Cubes are unable to roll up. Values must exist in the measure or aggregate tables at the precise level of a query. Otherwise, the values are shown as Null. Using a Custom regular aggregate allows you to customize measure values using advanced business logic.

  Use the Custom regular aggregate if you have a rollup that cannot be calculated with the existing aggregation types. Create the necessary aggregate tables to support the levels of queries that you are likely to require.

- Non-distributive measure. Count Distinct, Average, Standard Deviation, Variance, and Median are all non-distributive measures. Non-distributive measures must be computed from the fact table at the lowest level fact data and cannot be aggregated from one summary level to the next. If a database aggregate table that exactly matches the granularity of a SQL query exists, then the query can be routed to that aggregate. For example, assume two dimensions in a cube, Time and Products. Assume that Time has Year, Quarter, Month, and day levels. Assume that Products has Product line, Product type, and Products. Assume that the grains of the fact table are Day and Products. Assume again an aggregate table that has been aggregated to the Month level and Products level. If a query is for Month and Products, then the query can be routed to the aggregate table. If the query is for Quarters and Products, its results must be computed from the fact table.

  A dynamic cube stores the values of non-distributive measures that it computes in its data cache for later use.

  Because aggregate tables for non-distributive measures must always be computed from detail data and cannot be incrementally updated, be selective of the levels of aggregation used when choosing the levels at which to aggregate non-distributive measures.

- Regular Aggregate of Calculated. A calculated regular aggregate measure is dealt with similarly to non-distributive measures. The aggregation of any measure that is used as an element in the expression is calculated first. The result of that aggregation is used in the rest of the expression. This approach allows you to control the timing of the aggregation. Cognos Dynamic Cubes cannot decompose a calculated regular aggregate regular measure to take advantage of aggregate tables. The expression is computed from the fact table or from aggregate tables that match the precise level of aggregation of the query.

Measure design example

In this example, assume that you have a column for Sales in your fact table and you want to compute the average sales as efficiently as possible. In general, the design of any measure follows the example. Non-distributive measures, such as average, require additional consideration.

There are several approaches that you can take to implement this example.

Implementation option 1

An obvious implementation approach of this measure is to create a measure with a regular aggregate of Average. This approach has drawbacks. Because average is a non-distributive aggregation rule, the dynamic cube recognizes it as such and the query can be routed to aggregate tables subject to the rules outlined in 6.1.4, “Measure design” on page 165 in the *Non-distributive measure* bullet.

Otherwise, the measure is aggregated from the fact table at the lowest level of detail. A dynamic cube might be able to optimize the query, but only by using aggregate tables that precisely match the level of aggregation of the original query. Finally, if the underlying database is aggregate aware, the database might reroute the query to aggregate tables.

Implementation option 2

A second approach to modeling a measure with a regular aggregate of Average is to attempt to work around the non-distributive regular aggregate by creating a measure that does not use the non-distributive aggregate. In this example, the approach is to create a measure with a regular aggregate of sum, which sums the sales (for example, Sales_sum), another measure with a regular aggregate of Count (for example, Sales_count), and create a measure with the expression of Sales_sum / Sales_count and a regular aggregate of calculated.

The second approach could be considered worse than the first one. When a dynamic cube is published to the content store, it is represented in a model with two parts: A relational model and a dimensional model that contains references to query items in the relational model. In the second approach, the Sales_sum/Sales_count expression is part of the relational model, contained with a definition of a query item. A dynamic cube is unable to optimize the expression because it has no visibility to it. It knows only of its encompassing query item. Therefore, the dynamic cube is unable to route the query to any available aggregate tables because it knows nothing about the expression.

Implementation option 3

A better implementation is similar to the second one. You create the Sales_sum and Sales_count measures, which you hide by setting the Visible property of the measure to `false`. You then create a calculated (dimensional) measure with the same expression as in the second implementation, Sales_sum / Sales_count. You then set the regular aggregate of the expression to be average. This results in the Sales_sum measure and the Sales_count measure being computed by the database. The results of those two computations are passed to the dynamic cube and the expression is computed by the dynamic cube. The dynamic cube knows how the value is computed because the calculated measure expression is part of the dimensional model. The dynamic cube is able to use any aggregates that exist for the two source measures. The expression is computed in the dynamic cube rather than in the database but can be optimized by using available aggregate tables.

Conclusion

The best solution is one that takes advantage of a specific optimization within Cognos Dynamic Cubes for measures with a regular aggregate of Average. If the same fact column is modeled as three separate measures with regular aggregates of Sum, Count, and Average, Cognos Dynamic Cubes uses any available in-memory or in-database aggregates for the Sum and Count measures to compute the average value.

Because Sum and Count are distributive measures, it is possible to aggregate values for these measures from lower-level aggregate tables. This process makes it much easier and more flexible to optimize a measure with the non-distributive Average regular aggregate. If the measure with a regular aggregate of Average is the only one required by users, then the measures with the Sum and Count regular aggregates can be hidden by setting their visible property to `false`.

6.2 Time dimensions and relative time

A time dimension is defined in Cognos Cube Designer, and date and time semantics are assigned to levels of its hierarchies. These semantics allow for the introduction of relative time members encompassing current, prior, and next periods, as well as custom relative time members, described later in this section. Time is an important dimension in nearly all applications and the relative time functions of Cognos Dynamic Cubes allow users to author expressive reports with little effort.

6.2.1 Current periods

All relative time members within a time hierarchy are expressed relative to a leaf-level member of the hierarchy identified as the current period.

The Current Period property exists for each level in a time dimension. You can seed it with a set value or with an expression. Unless otherwise defined, the current period is the last member of the leaf level of the hierarchy.

The current period value does not need to be set for all of the levels in a hierarchy. If it is set for a non-leaf level, then the dynamic cube attempts to determine the current period for the other levels. For example, if you specify a current period value for a month level, the quarter and year levels can derive their values from it. The lower levels assign the current period to the last member on each level. If the current period values do not match, then no relative time can be calculated. For example, assume that you specify the month current period value to be August 2015 and the day current period value to be September 1, 2015. The day current period value does not exist in the current month. The mismatch prevents relative time members from being generated.

If you do not set the current period value for a level, the last member in the level is deemed to be the current period.

The source of the current period property can be a static value, which you enter in the expression editor. This is a simple approach, but it requires you to update the expression each time the current period changes, thus also requiring that you republish the cube.

Another, more flexible approach is to use an expression to extract or generate a current period value from the current database time stamp. The value for the current period must match the business key value for the period that you want as the current period.

A more durable approach is to create a single-row table that contains one or more columns that contain the key values that correspond to the levels of the time dimension for which you intend to define the current period. These columns can be added to a level, typically the highest, within a time hierarchy as hidden attributes that can then be used within the current period expressions of any level within the hierarchy. The single-row table must be joined to the time dimension table, possibly with something as simple as a column in each table that contains the value of `1`.

During ETL, the values necessary for the current period refresh the single-row table. Therefore, when the cube is started the current period values are obtained from the single-row table. This approach avoids using a system or static value to define the current period. You do not need to republish the cube to alter the definition of the current period member.

Example for current period expression from a database value

This section provides an example of configuring the current period expression to use a database value assigned during ETL. This approach allows you to control exactly what the current period is as a part of database updates and maintenance without requiring any model updates or republishing the cube. This approach joins a single-row table to the time dimension with a hidden attribute used as the current period expression.

Figure 6-4 shows an example of a single-row table containing a single column value with the current day key of 20141021 for October 21, 2014.

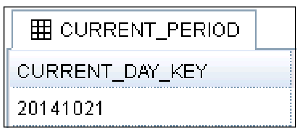


Figure 6-4 Example of a single-row table with current day key

Perform the following steps to configure the current period expression to use a database value assigned during ETL:

1. In Project Explorer, double-click the **Day** level of the `Time` dimension to open the level editor.
2. From the Source pane, select the CURRENT_DAY_KEY column of the CURRENT_PERIOD table to drag it into the level editor list of attributes.
3. Select the new **Current Day Key** attribute in the list.
4. In the properties, set the **Visible** property to `false` to make the attribute hidden.

Figure 6-5 shows the **Current Day Key** attribute in the **Day** level and the **Visible** property set to `false`.

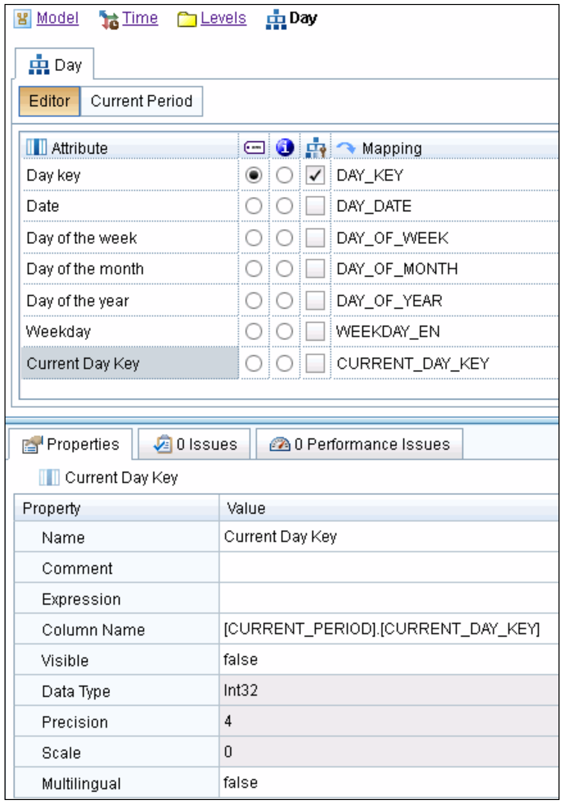


Figure 6-5 Current Day Key attribute added to Day level with Visible property set to false

5. Select the Current Period tab to start the Current Period expression editor.
6. Select the **Current Day Key** attribute from the Project Explorer and drag it into the Current Period expression editor.

   Figure 6-6 shows the **Current Day Key** attribute as the Current Period expression.

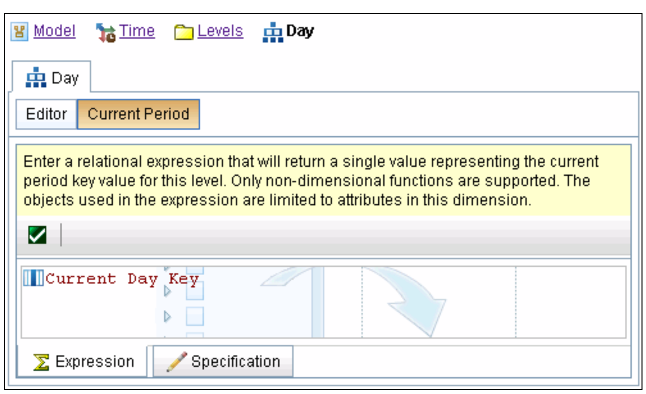


Figure 6-6 Current Day Key attribute in the Current Period expression editor

7. In Project Explorer, double-click the **Time** dimension to open the dimension editor.
8. Select the Implementation tab to see the Dimension Metadata Diagram.
9. Click **Create Join** in the Implementation tab toolbar.

10. In the Create a new join window, set the **Cardinality** between the tables to `One to One`.

11. Click **Add to join expression** to add a join relationship. Define the join as CURRENT_PERIOD.CURRENT_DAY_KEY = GO_TIME_DIM.DAY_KEY, as shown in Figure 6-7.

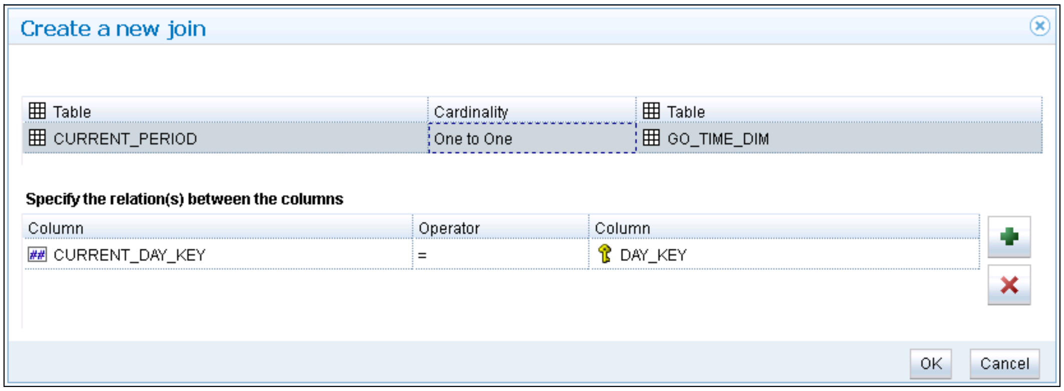


Figure 6-7 Join definition between Time dimension table and current period table

Figure 6-8 shows the Dimension Metadata Diagram with the new join defined.

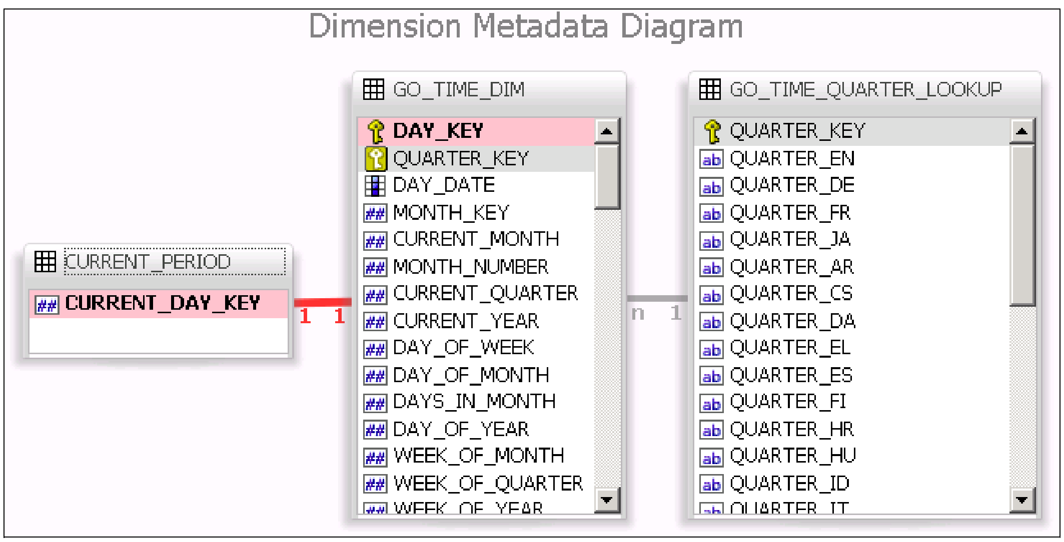


Figure 6-8 Dimension Metadata Diagram showing the Time dimension table joins

6.2.2 Creating a time dimension

A time dimension needs several elements defined:

- The *Dimension type* value in the Property window must be set to `Time`
- Each hierarchy must have the **Add relative time members** value in the Property viewer set to `true`.
- Each level in a hierarchy must have the appropriate **Level type** property value. The property value must be in order in the hierarchy. There are several wildcard properties, such as season, which are not required to be in order. For more information about the Level type property, see 6.2.4, “Level types” on page 173.
- Define the **Current periods**. For more information about the current period property, see 6.2.1, “Current periods” on page 168.

- Define the relative time autogeneration options. For more information about relative time members, see 6.2.5, “Relative time members” on page 174.
- Define custom relative time members. For more information about custom relative time members, see 6.2.6, “Custom relative time members” on page 175.

6.2.3 Exercise for creating a time dimension

Use the following steps to create a time dimension where the current period value is static value, set for all levels of a hierarchy, such that the current period corresponds to July 1, 2014.

> **Note:** As described in 6.2.1, “Current periods” on page 168, the current period value does not have to be set for all levels in a hierarchy. If it is set for a non-leaf level, then the dynamic cube attempts to determine the current period for the other levels. See 6.2.1, “Current periods” on page 168 for other approaches for setting the current period value.

1. Select the **Time (ship date)** dimension in the Project Explorer.
2. Set the **Dimension type** value in the Property panel to `Time`.
3. Select the Time (ship date) hierarchy.
4. Set the **Add Relative Time Members** value in the property viewer to `true`.
5. Select the GO_TIME_DIM table in Data Source Explorer.
6. Right-click **View Data** and pin the tab.
7. View the data in the CURRENT_YEAR column. CURRENT_YEAR is the business key of the Year level of the Time dimension. The values in CURRENT_YEAR generate members in Cognos Dynamic Cubes.
8. Select the **Year (ship date)** level in the **Time (ship date)** dimension.
9. Set **Level Type** to **Years**.
10. Click **Current Period** in the Property window.
11. Enter `2014` in the Current Period expression editor. You have now set the current period value for the year level.
12. View the QUARTER_KEY column in the pinned data tab for GO_TIME_DIM.

    The data is in the form of `20141`, `20142`, `20143`, and `20144`.
13. Select the **Quarter (ship date)** level in the **Time (ship date)** dimension.
14. Set **Level Type** to **Quarters**.
15. Click **Current Period** in the Property window. Enter `20143` in the Current Period expression editor using the same data form of the business key to correspond to Q3 of 2014.
16. View the MONTH_KEY column in the pinned data tab for GO_TIME_DIM.

    The data is in the form of `201401`, `201402`, `201403`, and so on.
17. Select the **Month (ship date)** level in the **Time (ship date)** dimension.
18. Set **Level Type** to **Months**.
19. Click **Current Period** in the Property window. Enter `201407` in the Current Period expression editor using the same data form of the business key to correspond to July 2014.
20. View the DAY_KEY column in the pinned data tab for GO_TIME_DIM.

The data is in the form of `20140101`, `20140102`, `20140103`, and so on.

Select the **Day (ship date)** level in the **Time (ship date)** dimension.

21. Set **Level Type** to **Days**.
22. Click **Current Period** in the Property window. Enter `20140701` in the Current Period expression editor using the same data form of the business key to correspond to July 1, 2014.

Now, if you refresh the Members folder, you will see the relative time members visible and corresponding to the current period of July 1, 2014. Figure 6-9 shows the relative time members sub tree, excluding the reference members.

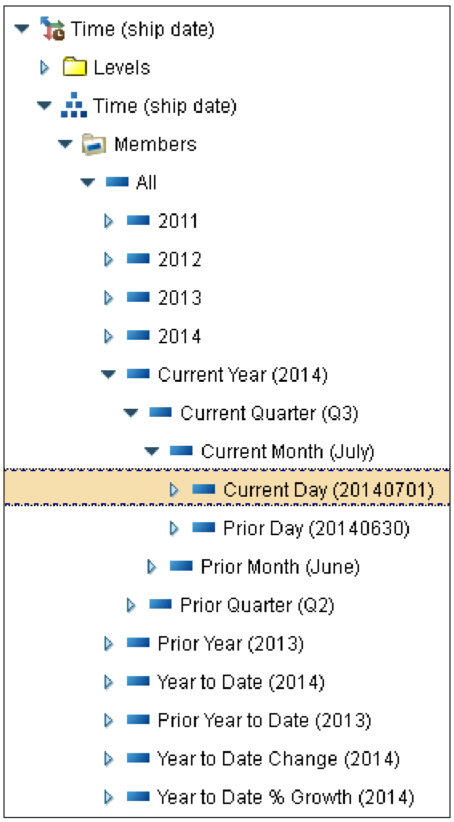


Figure 6-9 Relative time members sub tree with reference members excluded

6.2.4 Level types

All of the values in the Level type property are related to defining a time level except for Regular. The Level type property serves as the source of part of the relative time member's caption.

A level in a time hierarchy that has a Level type property value of Regular is identified and an error is issued. A hierarchy that has a Level type property value that is not in the correct order causes an error to be issued. A level type property does not affect member generation except to provide part of the name of some relative time members.

Periods, Seasons, and Holidays can be placed in any position in the hierarchy.

6.2.5 Relative time members

When you set the **Add Relative Time Members** property of a hierarchy to `true`, relative time members are generated automatically. You can control the types of relative time members that are generated.

The autogeneration of relative time members by default generates the current and prior periods members as well as a sub tree of the entire hierarchy that corresponds to the current and prior members.

Cognos Dynamic Cubes are capable of generating the following built-in relative time members where *period* can be any one of the level types assigned to a level in a time hierarchy:

- Current period
- Prior period
- Next period (not by default)
- Period to date
- Prior period to date
- Next period to date (not by default)
- Period to date change
- Period to date % growth
- Next period to date change (not by default)
- Next period to date % growth (not by default)

If the **Add Relative Time Members** property is set to `true`, six members are generated in the member tree. For example, if the level type is Years, the following members are generated:

- Current Year
- Prior Year
- Year to Date
- Prior Year to Date
- Year to Date Change
- Year to Date % Growth

Their names are derived from the level type property setting of the levels in the hierarchy.

You can control the generation of the relative time members in the Relative Time tab of the hierarchy editor. The default setting is to autogenerate prior periods and prior periods to date, and to autogenerate the reference relative time members sub trees. The next period and next periods to date members are not autogenerated by default.

If all of the settings are set off, only current, to date, to date change, and to date % growth are generated, as shown in Figure 6-10.

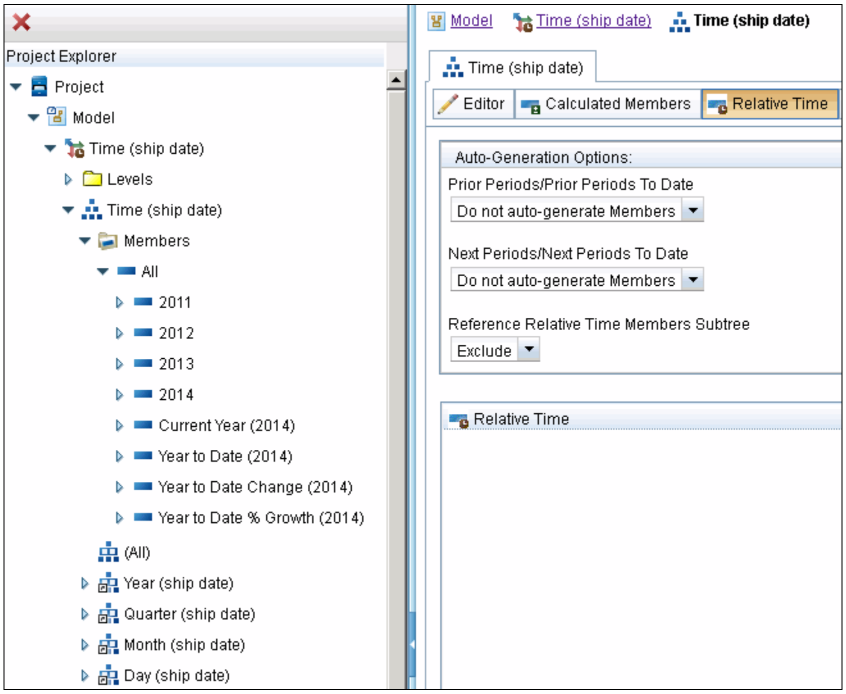


Figure 6-10 Relative time sub tree with auto-generation options set off

6.2.6 Custom relative time members

Custom relative time members are custom single period members, custom period to date members, and custom N-period running total members. These custom members are defined using a combination of properties described in this section.

Target Period property

Target Period specifies the period, or periods, for which you want to create a relative time member. The effect of your selection depends on what type of relative time member you are creating.

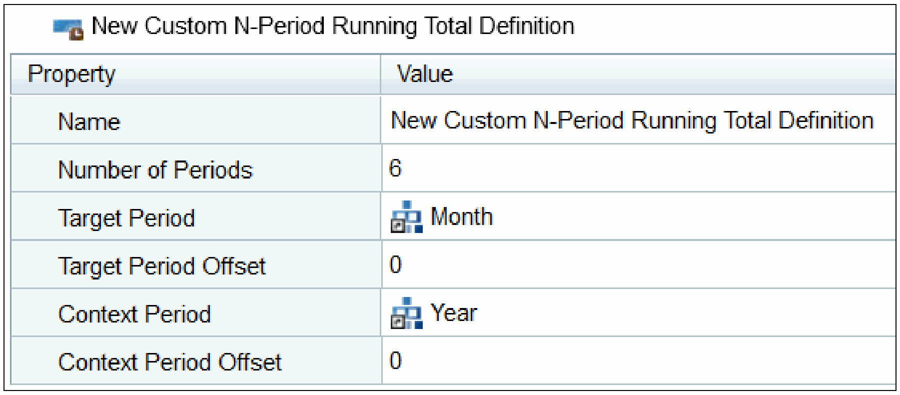
If you are modeling a Custom Single Period Definition, Target period specifies the type of period for which you want to construct a relative time category.

If you are modeling a Custom Period To Date Definition or a Custom N-Period Running Total Definition, Target period specifies the granularity of periods that you set to compute the total.

Target Period Offset property

The Target Period Offset property sets the position of the selected target period relative to the current period.

The effect of your selection depends on what custom relative time member you are creating. In general, it is the position away from the custom relative time period starting point. For example, assume that the custom relative time member is an N-Period Running total definition with the Target Period of Month, the Target Period Offset is 0, the Context Period is Year, the Context Period Offset is 0, and the Number of Periods is 6, as shown in Figure 6-11.



New Custom N-Period Running Total Definition

Property	Value
Name	New Custom N-Period Running Total Definition
Number of Periods	6
Target Period	Month
Target Period Offset	0
Context Period	Year
Context Period Offset	0

Figure 6-11 Custom N-period Running total definition

If the current month is March 2014, then this definition produces a custom relative time member with six children in the IBM Cognos studios metadata tree, each being a month from October 2013 to March 2014 as shown in Figure 6-12.

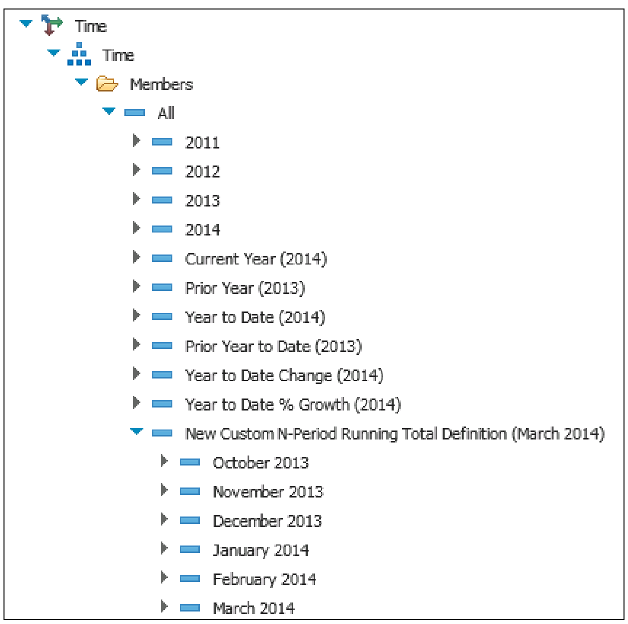


Figure 6-12 N-Period running total custom relative time member in IBM Cognos studio metadata tree

Figure 6-13 shows the custom N-period running total definition with a Target Period Offset value of 2 instead of 0. The Number Of Periods is 6, the Target Period is Month, the Context Period is Year, and the Context Period Offset is 0. The target period offset value moves the target period forward by 2 months, from March 2014 to May 2014.

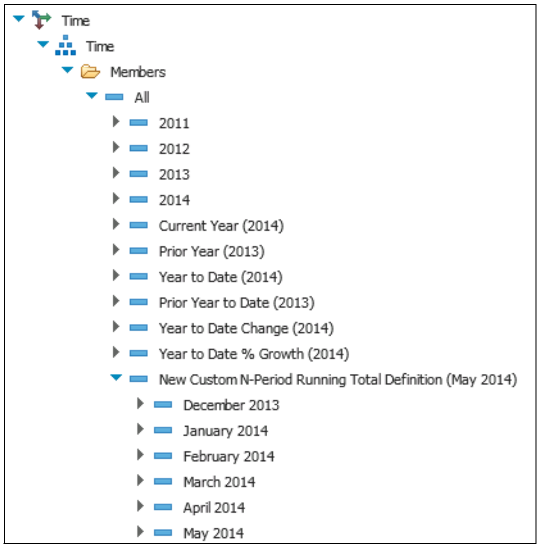


Figure 6-13 Custom N-period running total Target period offset illustrated

Context period

The Context Period property specifies a time period within which the target period occurs, establishing a context for the target period.

Context period offset

The Context Period Offset property sets the position of the context period relative to the current period, where zero (0) corresponds to the current period. A positive value changes the context forward. A negative value changes the context backward.

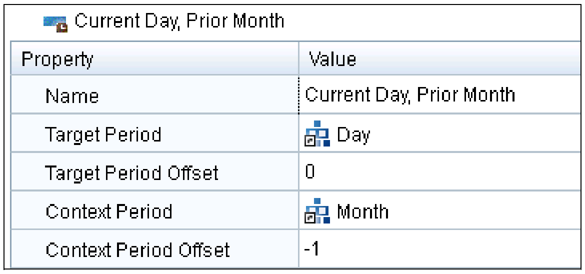
Figure 6-14 shows a custom single period relative time member definition. The **Target Period** property value `Day` indicates that the custom relative time member is a day. The **Target Period Offset** property value `0` indicates that the target is the current day. The **Context Period** property value `Year` indicates that the context is year. The **Context Period Offset** value is `-1`, which indicates that the context is the prior year. For example, if the current day is March 15, 2014, then the custom single period defined in Figure 6-14, `Current Day, Prior Year`, correspond to March 15, 2013.

Current Day, Prior Year

Property	Value
Name	Current Day, Prior Year
Target Period	Day
Target Period Offset	0
Context Period	Year
Context Period Offset	-1

Figure 6-14 Custom single period definition for the current day, last year

In another example, if the Context Period property value is set to `Month`, as in Figure 6-15, then the custom single period corresponds to February 15, 2014.



Current Day, Prior Month

Property	Value
Name	Current Day, Prior Month
Target Period	Day
Target Period Offset	0
Context Period	Month
Context Period Offset	-1

Figure 6-15 Custom single period definition for the current day, last month

When the parallel period has fewer members than the current period, the last member of the parallel period is returned. For example, assume that the current day is a leap day, February 29, the context period is year, and the context period offset is -1. In this case, the custom single period relative time member uses the values of February 28 of the prior year.

Figure 6-16 shows the example custom single period relative time members with their corresponding time member reference in parentheses in the metadata tree in IBM Cognos studios. Notice that because the **Target Period** is `Day`, the custom single period relative time members are listed at the same level as **Current Day**.

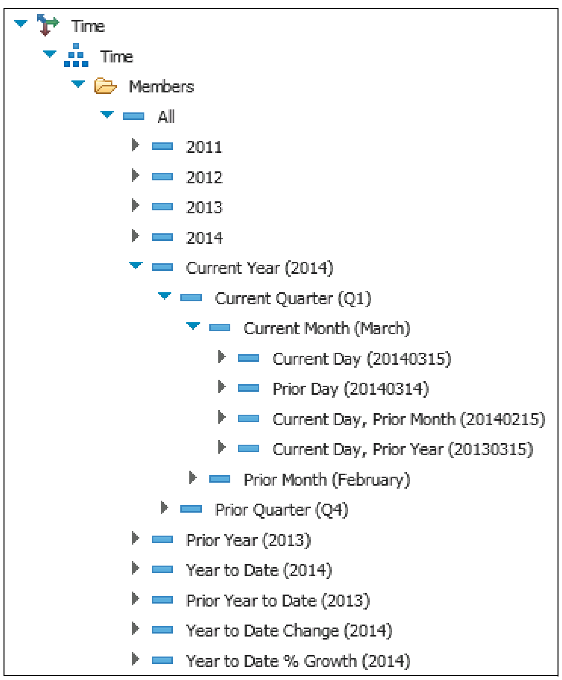


Figure 6-16 Custom single period members in the IBM Cognos studios metadata tree

For more information about custom relative time members, see the following articles:

- *New options to control the auto-generation of predefined relative time members in IBM Cognos Cube Designer 10.2.1.1 Fix Pack 3* can be found at:

  http://www-01.ibm.com/support/docview.wss?uid=swg27041029

- *New custom relative time members in IBM Cognos Cube Designer 10.2.1.1 Fix Pack 3* can be found at:

  http://www-01.ibm.com/support/docview.wss?uid=swg27040594

- *Dynamic Cubes may resolve relative time members differently to IBM Cognos Transformer* can be found at:

  http://www-01.ibm.com/support/docview.wss?uid=swg27041006

6.3 Slowly changing dimensions

Slowly changing dimensions (SCD) are a type of dimension in which the data describing the entities in a dimension changes over time. This section describes the SCD types and use cases that Cognos Dynamic Cubes supports.

Cognos Dynamic Cubes supports modeling both type 1 and type 2 slowly changing dimensions. In addition, it is possible to model time-stamped SCDs.

6.3.1 Modeling slowly changing dimensions

The following are the core tasks for modeling slowly changing dimensions:

- Identify any slowly changing dimensions in your intended application.
- Establish the slow change strategy.
- Determine what keys, if any, uniquely identify members in the context of the SCD.
- Determine what fact keys are sufficient.

An example of a slowly changing dimension that is easy to understand is an employee dimension. Employees are hired, promoted, given raises, change positions, change workplace locations, resign, retire, and are fired. The effect of these changes can alter the structure of a tree that describes the hierarchical relationship between members of the organization. For example, a change in position might result in an employee becoming a manager and acquiring subordinates who then appear under the employee in the member tree. The change can result in an employee exchanging one set of subordinates for another set.

A change in the value of an attribute of a member can also be significant. For example, a customer might get married. This change might affect how you analyze the customer. Depending upon the needs of your organization, you might have no need to keep track of data that was associated with the person before their marriage. Conversely, it might be important to identify the data that is associated with a person based on marital status.

How the data changes are handled classify the types of slowly changing dimensions. Although the types and classifications of slowly changing dimension types vary, the definition of type 1 and type 2 SCDs seem to be universally recognized:

- Type I SCDs are characterized by the overwriting of the attribute state by the new data. This type, obviously, has drawbacks. The most important drawback is the loss of the historical view of the data.
- Type II SCDs are characterized by a new record being created in the table for each change in the attributes of an entity that are tracked by the table. Slowly changing dimensions of this type need some way to identify each record in the dimension table to distinguish one member instance from another. The natural key of an entity will be repeated, but a surrogate key is needed to uniquely identify each record.

6.3.2 Modeling a regular slowly changing dimension

When you are modeling a regular slowly changing dimension, the business keys of those higher levels in the hierarchy that uniquely identify members must be included in a level. This approach has some performance implications, but, without it, the members will not be uniquely identified. In certain cases, it is possible for the cube to start and generate members, but it is an incorrect set of members.

The leaf level should have a unique key that exists in the database table. More information for handling slowly changing dimensions in a data warehouse is contained in the standard reference sources for data warehouses, and ETL.

The surrogate key must be joined to the fact table. In this way, each record can track the data for each instance of an entity that has changed.

To model a slowly changing parent-child dimension, both the parent and child need surrogate keys to uniquely identify a member. With proper qualification, a member could have multiple parents (which is not supported by Cognos Dynamic Cubes). Therefore, a surrogate key must exist not only for the key that identifies the member, but also for its position in the hierarchy.

In the EMP_EMPLOYEE_DIM sample table, for example, there are slow changes for entities in the manager levels 4, 5, and 6 and the Employee level. Each of those levels must have extra keys added to uniquely identify the members.

6.3.3 Modeling time-stamped slowly changing dimensions

A time-stamped slowly changing dimension is an SCD that is a snapshot at a point in time. To model it, create a filter expression. Ensure that the keys of the dimension after the filter match the keys of the fact table for the fact records that you want to use in the SCD. You need to be aware of the effect of filters on dimensions built using snowflakes.

6.4 Role-playing dimensions

The goal for modeling role-playing dimensions is to generate the relational queries in a predictable fashion. A role-playing dimension is a dimension that has more than one possible logical relationship between the dimension table and a fact table. Each represents an aspect of the relationship between the dimension and the fact table and how the aspects, or roles, give context to the fact. You must model role-playing dimensions where there are more than one possible relationship between the dimension table and the fact table or if the dimension table's dimension grain is lower than the grain of the fact table. For more information about multiple fact grains, see 6.5, "Multiple fact and multiple fact grain scenarios" on page 181.

Model role-playing requires you to separate each of the relationships so that only one relationship exists between an entity and the fact table. You must create a duplicate of the dimension for each additional relationship that you want to have. Each of these entities is referred to as a role-playing dimension. They exist as aliases to the dimension table. Each role-playing dimension uses one of the keys to the fact table.

For example, a Sales fact table can have a relationship to the time dimension based on both the sales date and the product shipping date. You must plan the query so that, if you want to know the quantity that was shipped in a particular month, the intended keys (those that identify the shipping dates) are used in the query.

With a role-playing dimension, you know that the query will be generated in the way that you want it to. A role-playing dimension enables a query to be generated in a deterministic way. If you use an object from the Sales date dimension in a report, you can be assured that the query will use the sales date key. If you use an object from the Product shipping date time dimension, you can be assured that the query will use the shipping date key. If you use a dimension grain-modeled role-playing dimension, then you know that query results will not be projected below the fact grain, which could mislead and confuse report consumers.

The sample Cognos Cube Designer model has these examples of role-playing dimensions:

- Time
- Time (close date)
- Time (ship date)
- Time to month
- Products
- Product type

`Time to month` and `Product type` are in the `gosldw_target` cube. The others are in the `gosldw_sales` cube. `Time to month` and `Product type` exist so that queries using those dimensions match the grain of the fact table used in `gosldw_target`, SLS_SALES_TARG_FACT. The fact grain for the time dimension in SLS_SALES_TARG_FACT is month. The fact grain in SLS_SALES_TARG_FACT for the products dimension is product type. The dimension grain for time (in the GO_TIME_DIM table) is day, which is a more detailed level of grain than month. The dimension grain for products (in the SLS_PRODUCTS_DIM table) is product key, which captures the grain of each product by the details of color and size, and is a more detailed level of grain than product type.

6.5 Multiple fact and multiple fact grain scenarios

This section describes modeling scenarios that include more than one fact table, and scenarios when the fact and dimension have differing level of granularity.

6.5.1 Multiple fact tables

If you want to allow the package users to query against multiple cubes, use virtual cubes.

With virtual cubes, you can build a cube that shares dimensions between cubes and build reports that use facts from multiple cubes. An example is to have a virtual cube that includes a cube with actual values and a cube with target or budget values.

To the virtual cube, you can build in calculated measures that reference measures from both source cubes. An example is a measure that calculates the variance from plan by subtracting the plan values from the actual values. If the remainder is a positive number, then the variance exceeds the plan. Another example is a measure that takes that variance and compares it to the plan value to produce a percent of plan measure.

By having built-in calculated measures in the virtual cube model, report authors can reference these measures in multiple reports instead of re-creating these expressions in each report.

Including calculated measures in the model provides a single source for expressions and reduces the risk of an expression being incorrectly created in one or more reports. Enforcing commonality can reduce misunderstanding and misinterpretation of information. In addition, because it is built in, the performance of the calculation can be faster than a calculation made in a report.

Virtual cubes are covered in greater detail in Chapter 8, “Virtual cubes” on page 223.

6.5.2 When the fact grain and dimension grain differ

A common modeling problem is multi-granularity, which occurs when levels of dimensional detail for facts differ. In the data source, the level of information in a dimension can be more precise than the fact data of some fact tables in which the dimension takes part.

For example, a time dimension can have dimension information for the levels of year, quarter, month, and day. Day is the leaf grain or dimension grain. For a Sales fact table, the facts might exist at the day level, or a yet more precise level of detail. For a fact table with planned sales values, the fact grain or level is probably at a higher level of detail such as month.

This difference in fact grain can make it difficult to correctly plan queries if a report user included in a report a dimension level that is below the fact grain.

The core multi-fact grain modeling problem is double counting. You must recognize and avoid double counting. You must know when you have to model for multi-fact grains and how to decide which approach to use to model for it.

The term double counting can confuse people into thinking that they must be looking for values that are exactly double the expected value. Double counting is more accurately thought of as multiple counting. It is the by-product of attempts of an SQL query to project below the fact grain. The result is that the value for the fact grain is returned for each record of the dimensions that are projected below the fact grain.

For example, if a query attempts to retrieve results for the days level for a fact table that has data captured at the month grain, then for each day member the value for the month is returned. Therefore, the value for the month is aggregated by the number of days in the month.

A dimension table can have several columns in it that capture related data but at differing levels of detail or classification:

- A time dimension, which has years, quarters, months, and days.
- A product dimension, which has product line, product type, and products.
- A customer dimension has columns that describe the geographic detail for customers such as the countries where customers live, national subunits such as states or provinces, cities, and actual street addresses. The customer dimension could have related columns that describe other types of attributes of customers such as their type and subtype.

These differing levels of detail usually correspond to levels in a hierarchy. The lowest level of detail is the grain of the dimension table. It might also be called the leaf grain. In multi-fact situations, you might need to deal with facts that exist at the levels other than the leaf level of the dimension.

A fact grain is the level of dimensional classification where the fact data has been captured.

When you create a relationship between a dimension and the measure dimension in a cube, you must be able to specify the grain of the fact for a dimension. The fact grain is either at or above the dimension grain. When the fact grain and the dimension grain match, a query is not in danger of generating double counting. When the fact grain is above the dimension grain, a query might be in danger of generating double counting.

To avoid problems with double counting, specify the grain in the relationship between the measure dimension and the dimension. If the fact grain is above the dimension grain, click to clear the **Join is at the lowest level of detail for the dimension** check box. This action generates a `group by` determinant in the relational model with the key of the dimension end of the relationship as the key. This setting guides the query to know the context of the dimension and fact grains, generate the appropriate SQL, and avoid double counting. Figure 6-17 shows the relationship editor.

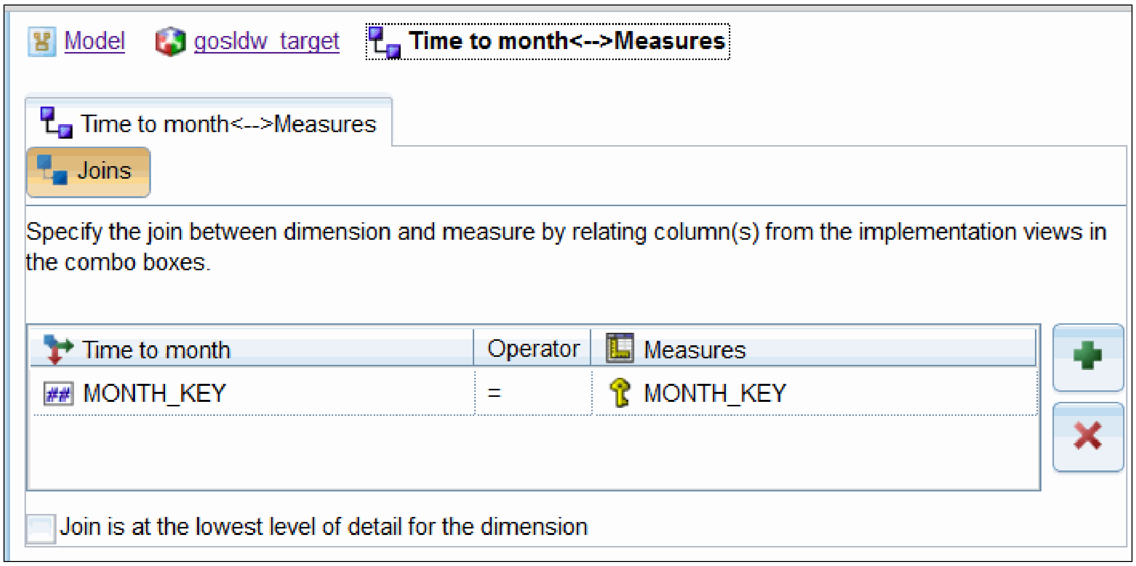


Figure 6-17 Relationship between Time to month and the measure dimension of gosldw_target

The basic approach allows you to use a dimension for all cubes of all fact grains. Figure 6-18 shows Sales target value for the Time dimension grain of month.

Sales target	2010					
	Q1			Q2		
	January	February	March	April	May	June
Camping Equipment	22,627,100	26,702,500	23,039,600	22,970,700	23,830,200	26,286,600
Golf Equipment	10,303,100	11,910,500	10,994,600	10,750,100	11,107,400	11,746,800
Mountaineering Equipment						
Outdoor Protection	2,479,300	2,845,500	2,489,000	2,465,400	2,561,100	2,813,900
Personal Accessories	22,218,600	26,337,000	33,218,500	23,768,500	30,027,200	32,747,600

Figure 6-18 Sales Target values for the Time dimension grain of Month

Differing granularity and multiple fact tables

When merging cubes into virtual cubes, you might encounter a scenario where the corresponding fact tables of each cube are at differing levels of granularity. The method for handling this multiple fact granularity scenario is to create role-playing dimensions for each instance of a dimension that have varying levels of granularity to different fact tables. In addition, clear the **Join is at the lowest level detail of the dimension** check box. These dimensions have their hierarchies truncated to the level matching the grains of the fact tables in the cubes in which they belong. Because the dimension grain matches the fact grain, there is no possibility of projecting data below its grain.

As an example, in the sample model, you must model a time dimension down to the day level for the Sales cube, and model a time dimension down to the month level for the sales target cube.

For the Sales cube, the relationship is formed between the appropriate keys in the fact table (ORDER_DAY_KEY for the measure dimension's key to `Time`, SHIP_DATE_KEY for the measure dimension's key to `Time (ship date)`, CLOSE_DATE_KEY for the measure dimension's key to `Time (close date)`) and the dimension (DAY_KEY).

For the sales target cube, the relationship is formed between the month level and the fact table keys (MONTH_KEY).

If you want to create reports that use data from both cubes, you can create a virtual cube. The virtual cube can have the two time dimensions merged. The levels of both source cubes are merged. The virtual cube might have levels that are below the grain of measures of one of the source cubes.

Queries that are made in the virtual cube that use dimension levels below the grain of a measure return null values, ensuring that the consumers of the cube do not have double counting.

The sample model demonstrates the merging of the two time dimensions in the virtual cube `gosldw_sales_and_target`. Figure 6-19 shows the `gosldw_sales_and_target` virtual cube in the sample model. For more information about virtual cubes, see Chapter 8, "Virtual cubes" on page 223.

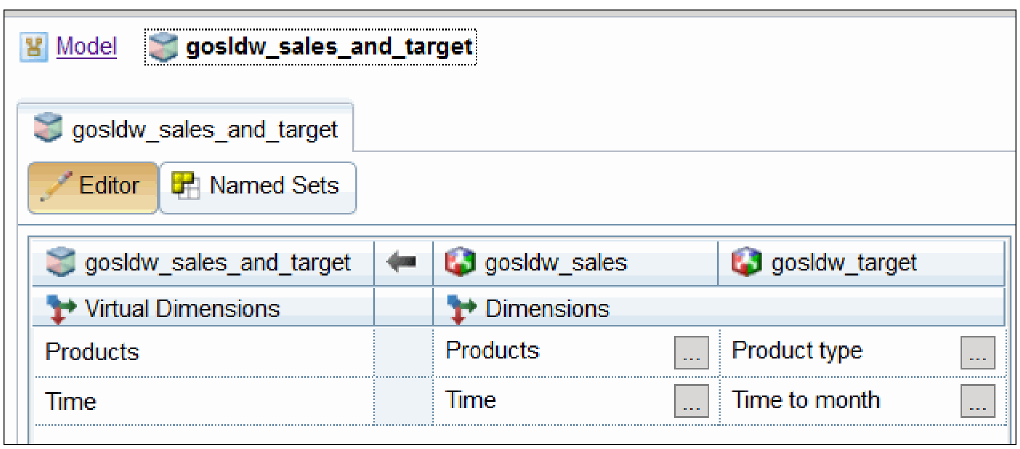


Figure 6-19 Merging of dimensions for multi-fact grains

As in the Cognos Cube Designer sample you can have the following cubes:

- A sales fact cube where the fact grain for the time dimension is at the day level.
- A planned sales cube where the fact grain for the time dimension is at the month level.

A virtual cube that uses both of these cubes as its source enables you to make a query with day level objects (either the level itself or a member of that level, depending on the Cognos studio that you are using) against the sales facts, and get results and get the expected null values for the planned sales facts. If you use a time dimension grain that is common to both fact tables, you get non-null results for measures from both fact tables.

6.6 Data quality and integrity

Data quality is a major consideration for ETL processing and of the greater data management needs of the organization. It is the assurance of the validity of what is tracked in the data warehouse.

The scope of ETL data quality is largely outside of the bounds of this book but you must be aware of this issue, and know how to detect and address data quality symptoms.

Data can exist in the fact table that does not have matching dimension keys. There can be data in the dimension tables that do not have matching fact data. This last case in itself is not going to be a problem, but you should know that this case can exist. ETL could create duplicate keys or keys whose data type does not match the column of the table.

Cognos Cube Designer has functionality to alleviate some aspects of data quality issues. The default filter behavior, which is governed by the **Exclude Facts Without Corresponding Dimension Keys** property (see Figure 6-20), is to add a filter to the fact table to exclude any keys that are not in the dimension table. If a key exists in the dimension table and not in the fact table, then you will get a null result for that key. You can adjust the **Exclude Facts Without Corresponding Dimension Keys** property of the filter to not generate the additional fact table filter. You can also set a filter in the measure dimension.

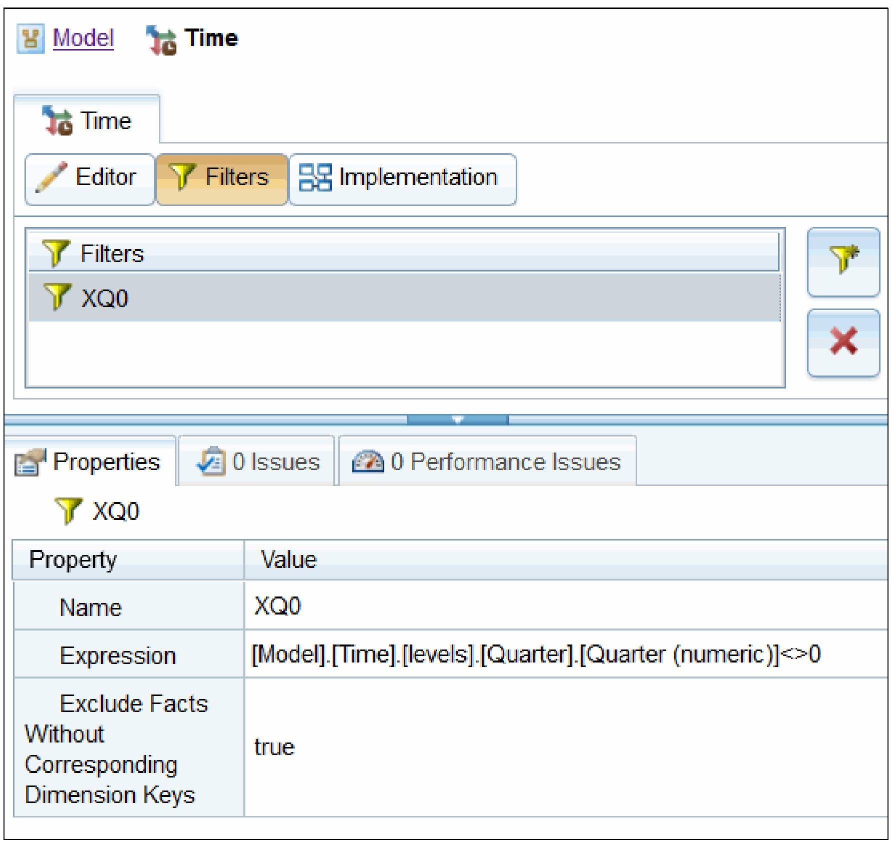


Figure 6-20 Properties of a filter

Part of the control over the data quality must take place when the data is acquired, possibly from the merging of disparate systems.

Another aspect is the control over the data quality during ETL. You must know methods for identifying possible data quality issues.

Most databases support referential integrity. Often it is turned off or is made declarative and the integrity is only enforced during the ETL processing. Erroneous modifications that are made to the data during or outside of the ETL process can create cases where a fact table has no matching dimension records. This situation might affect query results and cause discrepancies between subtly different but essentially similar queries.

Each data point in a dynamic cube is defined by a member from each dimension in the cube. If a value is required for some data point, then the SQL generated by Cognos Dynamic Cubes does not specify a filter on the table associated with a particular dimension if the member of that dimension is the All member. This approach to SQL generation allows for smaller and faster running queries.

When a dimension is in scope, the join between the fact and dimension table is specified in the SQL query and the dimension is filtered by an explicit set of dimension key values. When the member of a dimension is the All member, dynamic cubes do not specify a join filter for that dimension. All records are included, even records with invalid or missing dimension key values. This difference causes a discrepancy between values, depending upon which dimensions are involved in a query.

You should validate your records before implementing Cognos Dynamic Cubes.

Example 6-1 shows an example of a query that you should run for each dimension in a dynamic cube.

Example 6-1 SQL query to determine referential integrity errors between fact and dimension tables

```
select distinct FACT.Key
from FactTable FACT
where not exists
(select *
 from DimensionTable DIM
 where DIM.Key = FACT.Key)
```

The results of this query determine whether there are any fact records with invalid dimension key values. Any returned data is the set of invalid dimension key values. If no data is returned, there are no referential integrity errors.

The SQL query can also be used as a subquery to obtain the full set of records from the fact table.

If your fact table contains records with invalid or unknown dimension key values, a common practice is to create a row in the dimension table to represent these dimension keys. New fact rows with invalid or unknown dimension key values can be assigned this dimension key value until the fact records and the dimension table can be updated with the correct information. With this practice, records with problematic dimension key values are visible, regardless of which dimensions are involved in a report or analysis.

You should also validate snowflake dimensions.

You might have a situation where tables in a snowflake dimension are joined on a column for which the outer table did not contain values for rows in the inner table. In this case, the inner dimension table joins to the fact table, but the outer dimension table does not join to the inner dimension table.

Example 6-2 shows an example of an SQL query that you should run to ensure that snow flake dimensions do not have this type of referential integrity error.

Example 6-2 SQL query to determine referential integrity errors between snow flake dimension tables

```
select distinct INNER.Key
from D2_inner INNER
where not exists
(select *
 from D1_outer OUTER
 where OUTER.Key = INNER.Key)
```

In this example, the dimension is built from two tables: `D1_outer` and `D2_inner`. `D2_inner` is joined to the fact table. Key is the column on which the two dimension tables are joined.

Any returned data is the set of invalid dimension key values. If no data is returned, there are no referential integrity errors.

6.7 Logical Modeling

You can model the structure or your application before you access metadata, and then plug in the metadata later. This approach is called logical modeling. It allows you to model an application that can be reused with different metadata. You can model before having access to the metadata.

The normal modeling workflow is to import the metadata from the data sources and then to build your model from that metadata.

The logical model workflow inverts the normal workflow. You first define the dimensions and cubes that address your reporting requirements. You then map your data source metadata to the logical model as shown in Figure 6-21.

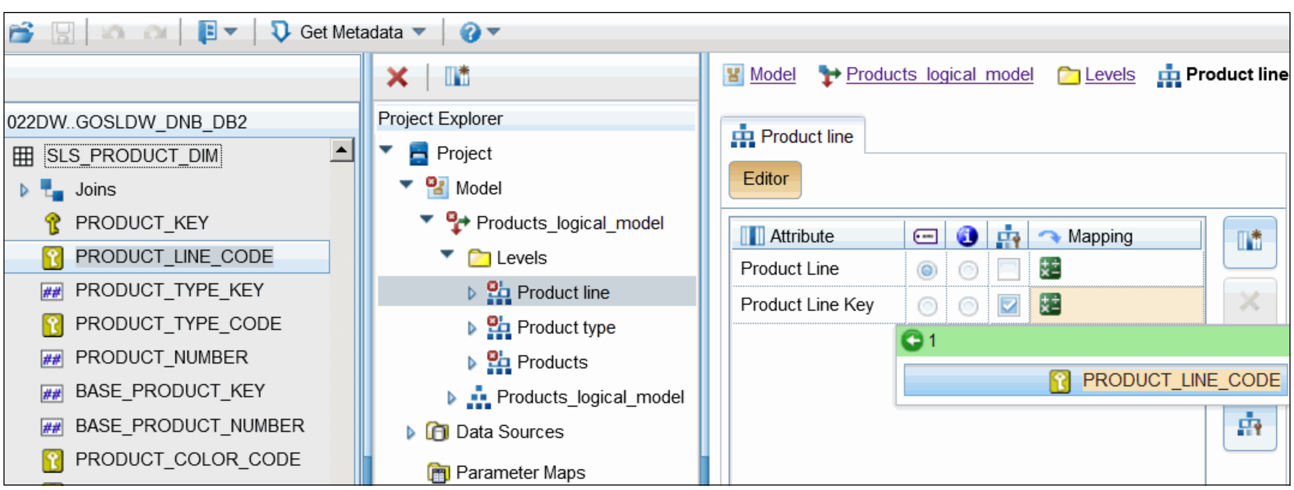


Figure 6-21 Mapping an attribute in a logical model to a database column

By following this approach, you can model a logical model without immediate reference to any particular set of physical metadata, thereby freeing you from any particular metadata. Your logical model could be used with different metadata, such as a snowflake or a dimension table structure.

After you map metadata to your logical model, you can customize the model to refine it and address the requirements of modeling to suit the metadata such as filtering a lookup table.

An example of an attribute being mapped to a metadata column is shown in Figure 6-22. In the level Product line, the attribute Product Line Key has the column SLS_PRODUCT_DIM.PRODUCT_LINE_CODE mapped to it.

The PRODUCT_LINE_CODE is now in the mapping cell as shown in Figure 6-21 on page 187.

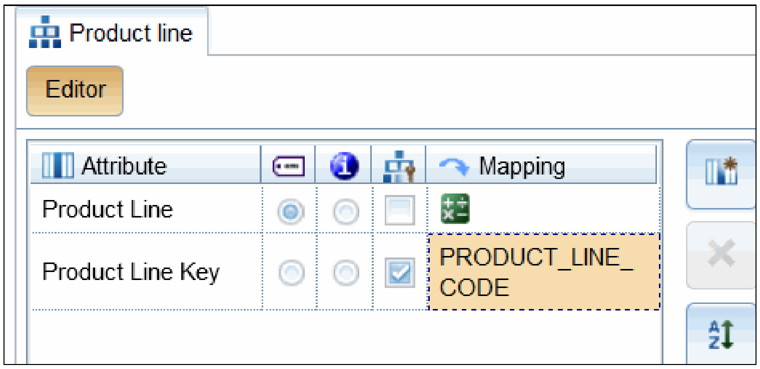


Figure 6-22 Column mapped to an attribute.

6.8 Modeling for null values

You need to be aware if there are null valuesin the data you are working with. Null valuesaffect query results. If you have experience working with SQL, you might have encountered this situation.

Figure 6-23 shows the result of a query where the results of an expression differ.

Retailers

No null checking	Null handled	Rtl Address1	Rtl Address2	Rtl City
2211 Connecticut Ave. Suite 210 Washington	2211 Connecticut Ave. Suite 210 Washington	2211 Connecticut Ave.	Suite 210	Washington
	138 Murphy Road Wilmington	138 Murphy Road		Wilmington
	10003 South Orange Blossom Trail Orlando	10003 South Orange Blossom Trail		Orlando
	1903 Garber Drive Tallahassee	1903 Garber Drive		Tallahassee
	3684 Bull Street Savannah	3684 Bull Street		Savannah
	1520 Ala Moana Blvd. Honolulu	1520 Ala Moana Blvd.		Honolulu
	511 W. 37th Street Boise	511 W. 37th Street		Boise
2845 Fullerton Avenue Suite 320 Chicago	2845 Fullerton Avenue Suite 320 Chicago	2845 Fullerton Avenue	Suite 320	Chicago
4430 Stevenson Drive P.O. Box 3571 Springfield	4430 Stevenson Drive P.O. Box 3571 Springfield	4430 Stevenson Drive	P.O. Box 3571	Springfield
	1837 Manderly Drive Bloomington	1837 Manderly Drive		Bloomington
3823 Douglas Avenue Unit 3 Des Moines	3823 Douglas Avenue Unit 3 Des Moines	3823 Douglas Avenue	Unit 3	Des Moines

Figure 6-23 Query result differences handling for null values

The attribute `No null checking` has the following expression:

```
Rtl Address1 ||' '|| Rtl Address2 ||' '|| Rtl City
```

The attribute `Null handled` has the following expression:

```
Rtl Address1 ||' '||

if (Rtl Address2 is null)
        Then
        ('')
        Else
        (Rtl Address2)

||' '|| Rtl City
```

Because RTL_ADDRESS2 has null values for some records, the results for any instance of the expression returns nulls even though the other elements of the expression return explicit non-null values.

If the value for the member caption is NULL, then the key value for the member is substituted in the member tree in the Cognos studios.

6.9 Scenario dimensions

Scenario dimensions are dimensions that group data that is related conceptually, but the postulated state or situation or the nature of the data do not make the aggregation of the data possible in a meaningful way. The most common types of scenario dimension uses are budgeting, planning, and forecasting.

It usually does not make sense to aggregate, for example, actual and budget values, so a scenario dimension usually does not have an ALL member. In this case, to model set the `Multiple root members` property of the hierarchy to `true`. The members of the top level of the hierarchy are displayed as the highest members of the hierarchy in the member tree as shown in Figure 6-24.

For example, you can have a member for actuals and another for budget. Each root member in essence becomes the root member of its descendant members.

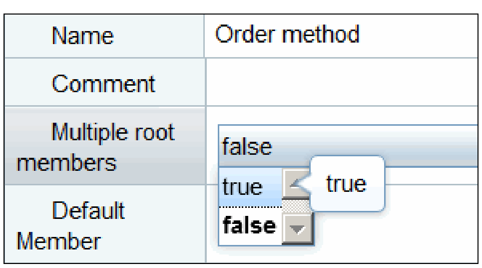


Figure 6-24 Multiple root members property of a hierarchy.

It is important to define the default member for a hierarchy, especially if there is no `All member`. If you do not define the default member, the first member is designated as the default member. This member, and the scenario that it represents, is the scenario that is displayed when a report is run. For example, if no member of the hierarchy is included in the report, the data reflects the values that are associated with the default member. It is possible that the choice of any scenario member as the default member is confusing to users. They might not understand where the data is coming from.

A technique to alert the report authors and consumers to choose a scenario is to create a member that returns null values until the users make a choice to include a specific scenario that they want to use. This technique is explained in 6.9.1, “Modeling a scenario dimension with a forced null default member” on page 190.

A scenario dimension can have any number of levels. If the hierarchy has only one level, the forced null default member technique illustrated in 6.9.1, “Modeling a scenario dimension with a forced null default member” on page 190 will not be valid.

Another technique to alert report authors and consumers to choose a scenario is to incorporate scenario dimension objects into report prompts. This technique prompts the report consumer for the scenario they want to see. The report prompt must be created for each report.

6.9.1 Modeling a scenario dimension with a forced null default member

Perform the following steps to model a scenario dimension with a forced null default member:

1. Set the hierarchy multiple root members property to `true`.

   If you do not set the multiple root members property to `true` first, the calculated member will have the `All` member of the hierarchy set as its parent member in the parent member property of the calculated member.

   > **Tip:** If you accidentally create the calculated member first, you can clear the parent member property value by selecting the calculated member in the `Calculated Members` folder, and selecting the **Parent Member** property value in the properties and pressing the **Delete** key.

2. Create a calculated member.
3. Give an appropriate name to the calculated member so that users can identify it and its purpose.
4. Enter NULL into the expression editor of the calculated member.
5. Select the hierarchy.
6. Expand the member folder.

7. Select the calculated member and drag it onto the **Default Member** value field as shown in Figure 6-25.

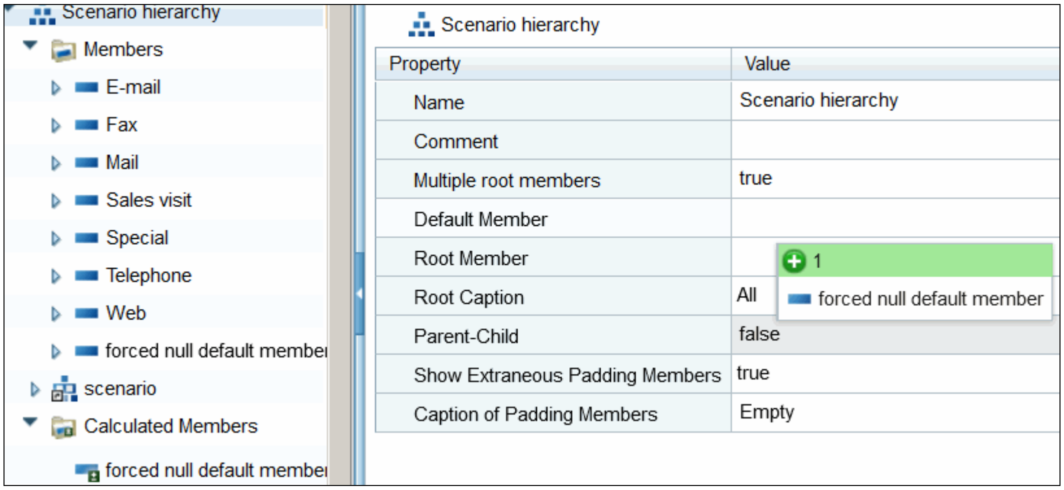


Figure 6-25 Setting the default member value by dragging a member from the member tree.

When users run a report, they will not see data unless some object from the scenario dimension is in the report, for example in an axis or the context filter.

Tip: You can delete a default member value by selecting it and pressing **Delete**.

6.9.2 Modeling a secured scenario dimension

A similar technique can be used to model secured hierarchies in scenario dimensions.

If the member security is applied to a hierarchy and the default member is not accessible, then the default member is the first accessible member found. The search is performed in a breadth-first manner.

Use this technique to model a secured scenario dimension:

1. Create a calculated member.
2. Create the expression of the calculated member:

```
AGGREGATE (currentmeasure WITHIN SET members ({name of the top level of the
scenario dimension hierarchy}))
```

3. Make this calculated member is the default member for the hierarchy.

6.10 Troubleshooting

This section provides information for troubleshooting common issues that you might encounter.

6.10.1 Double counting

Double counting is the industry term to describe the incorrect aggregation of data. The usual cause of the problem is a mismatch of the dimensional modeling to the fact grain of the data.

For more information, see 6.5, “Multiple fact and multiple fact grain scenarios” on page 181.

6.10.2 Importing metadata

Cognos Cube Designer models consist of a model file plus one or more metadata files. For each schema that is imported into the model and used, a metadata file is created in the data directory. The data directory is in the directory where Cognos Framework Manager and Cognos Cube Designer are installed.

When you open a model, Cognos Cube Designer attempts to load the metadata that is specified in the data sources that are used in the model. If it cannot load, it displays the following message when you try to expand folders in the metadata tree:

```
BMT-MD-6606 Cannot find the data source connection for this data source.
```

If you expand the data sources in the Project Explorer view and select a data source, for example `great_outdoors_warehouse`, you should see something similar to Figure 6-26 in the Properties tab.

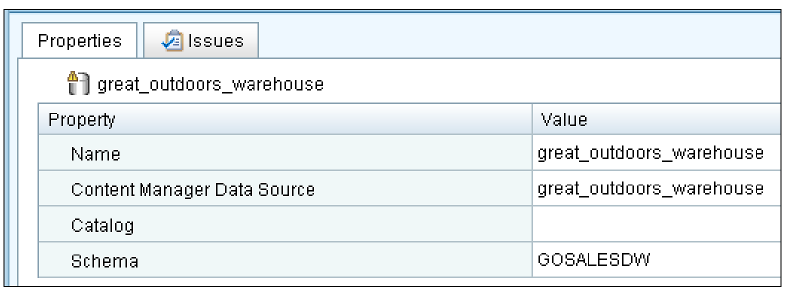


Figure 6-26 Data source connection

The four properties of a data source connection are the Name, the Content Manager Data Source, the Catalog, and the Schema. The Catalog and Schema might be familiar to you. They are properties derived from the databases that you are working on. Depending on your database, you might have a Catalog. If not, this property has no value. Content Manager Data Source property is the name of the data source in Content Manager. Name is the name of the data source given to it by Cognos Cube Designer when the metadata was imported. It is derived from the Content Manager Data Source name.

You can change the data source Name property to any name you want if no other data source in the model has that name. You can change the Content Manager Data Source value to be any value you want that matches a valid data source in Content Manager. You might want to change the value if you want to switch your model from a test environment to a production environment.

After you have changed the Content Manager Data Source, Catalog, and Schema values to match what you have, you need to fetch the metadata of that data source connection by choosing the refresh metadata menu item from the context menu for the data source. Figure 6-27 shows the refresh metadata menu item.

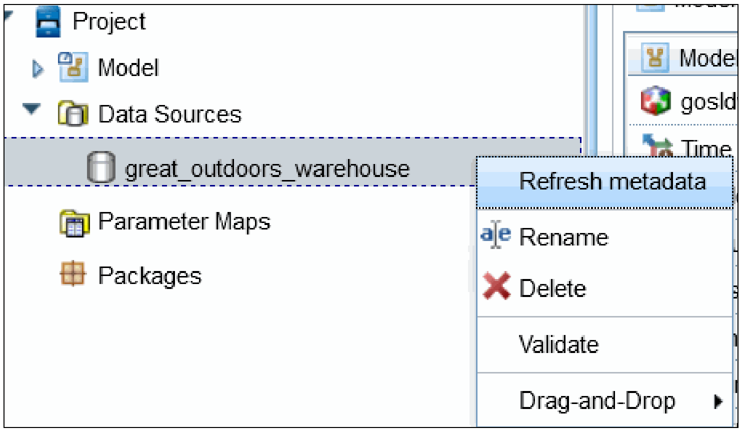


Figure 6-27 Refresh metadata menu

For more information, see Chapter 13, "Dynamic cube lifecycle" on page 445.

There is a data source for each unique set of schema and catalogs of data source connections that you are using in the model. Cognos Cube Designer appends a number to each additional data source if the name exists as a data source. You can change these names to anything you want. Figure 6-28 shows a model that has two data sources from the same data source in Cognos Content Manager. Each data source that is shown in Cube Designer is derived from a single schema.

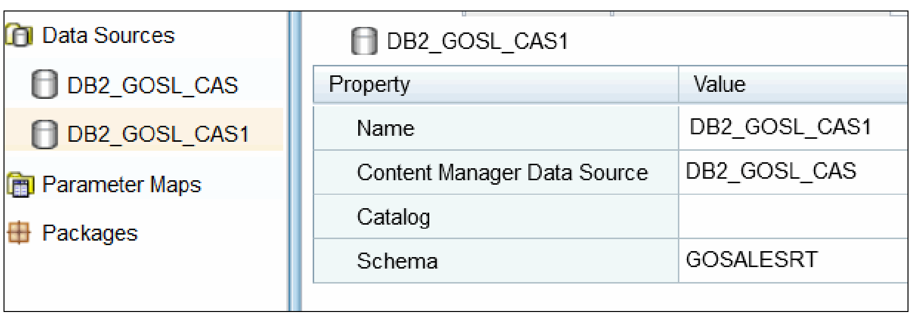


Figure 6-28 Two data sources in a model.

In the Issues tab you see a message similar to the following message:

```
The data source great_outdoors_warehouse does not have physical metadata
associated with it. The data source needs to be refreshed.
```

You must change the values of the data source properties to match your connection. Then select the data source in the model and select **Refresh metadata**.

In your data directory, you see a created relational metadata file. The name has the following structure:

relmd_{*whatever data source connection name you have defined*}_{catalog}_{schema}

The data directory exists in the location where you installed Cognos Cube Designer.

If you have physical metadata schemas in the model but not all are used in the model, you will see the following message in the Issues tab:

```
The data source { } does not have physical metadata associated with it. The data
source needs to be refreshed.
```

You might also see messages in the Issues tab that identify objects that do not have valid references. The method for handling that case is the same as the general metadata retrieval case.

6.10.3 Publishing errors

You might see the following message after you publish a cube with all the publish options set to `On`. This message indicates that the cube startup failed to respond before a timeout.

```
BMT-MD-6587 The Query Service has not acknowledged that the cube configuration has
been received. Please verify the cube configuration and status in the IBM Cognos
Administration portal.
```

This is usually due to a transient timeout problem.

To address the problem, manually start the cube in IBM Cognos Administration as shown in Figure 6-29. For more information, see Chapter 7, “Administering dynamic cubes” on page 199.

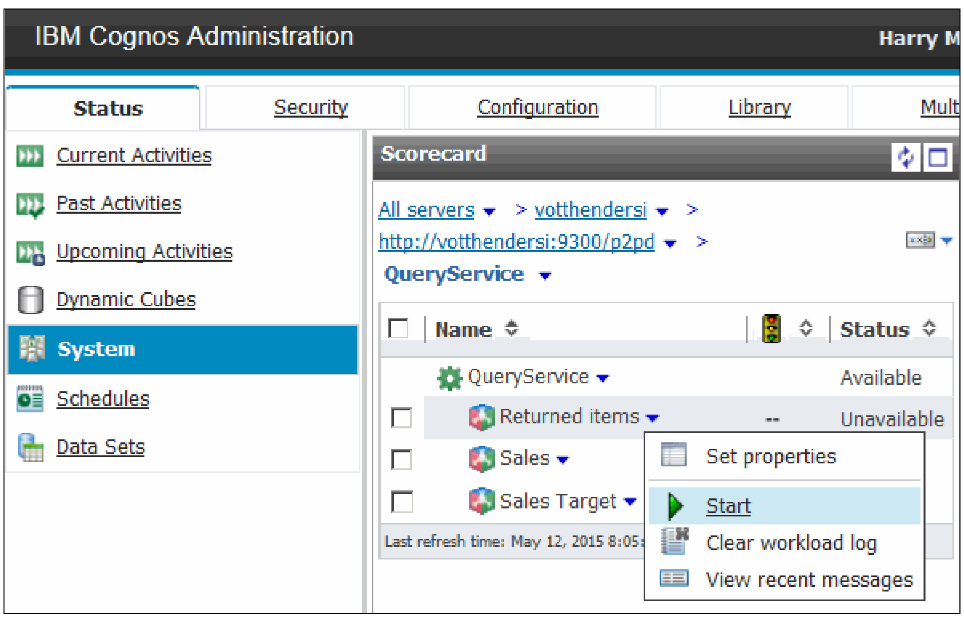


Figure 6-29 Manually starting a cube in IBM Cognos Administration

6.10.4 Refreshing metadata that changed in the database

One of the databases that you are using for your model might contain changed metadata. Changes can include added or deleted tables and columns, and changed table and column properties such as column data types.

Right-click the data source and select **Refresh metadata**.

Figure 6-30 shows the **Refresh** context menu action when you right-click the data source, and the **Refresh** toolbar button for refreshing metadata.

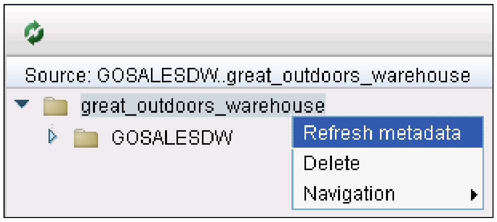


Figure 6-30 Refresh metadata in the data source view

Confirm that the changed metadata displays in the metadata viewer. Validate the model. If something that is used in the model was removed, correct the object references so that they point to the replacement objects.

6.10.5 Members

You must refresh the members of a hierarchy any time you make a change to a dimension or a hierarchy to view the effect of the change. To refresh members, right-click the members folder and select **Refresh** as shown in Figure 6-31.

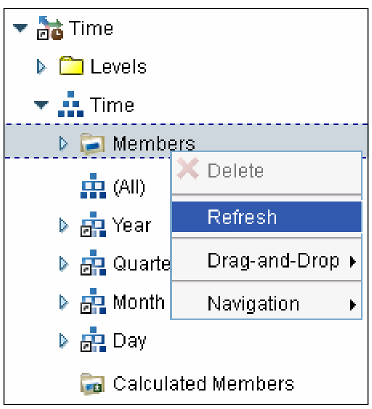


Figure 6-31 Refresh menu action for Members folder

You can also right-click the dimension and select **Refresh members** as shown in Figure 6-32.

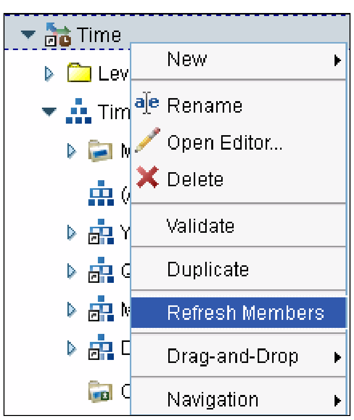


Figure 6-32 Refresh Members menu action for a dimension

6.10.6 Missing members

If a null value exists in a source column for an attribute that is being used in the level key, some members cannot be generated. If the null value happens in a member caption, the member caption displays NULL as its caption, even if other elements of the expression that are used to define the member caption are not null.

To correct this problem, change the attribute expression to handle null values. See 6.8, "Modeling for null values" on page 188 for more information.

6.10.7 Ellipses in the project viewer tree

Sometimes the Project Viewer tree might display ellipses for some parts of the project. To force it to generate correctly, resize the Project Viewer tree.

6.10.8 No Protocol message while running Cognos Cube Designer

If you start Cognos Cube Designer from the .EXE file directly from the Framework Manager `install` directory, you might encounter problems. Error messages such as `No Protocol` are displayed. You must run Cognos Cube Designer from the shortcut that is created during the installation. The shortcut has the `Start in` value set to run in the `BIN` directory. Always start Cognos Cube Designer from the shortcut in the **Start** menu or from the **Tools** menu in Framework Manager.

6.10.9 Logging in Cognos Cube Designer

There are two methods for logging in Cognos Cube Designer:

- Debug console

  The debug console is a separate window that is started when Cube Designer is run. The debug console logging is designed to address problems where a stack trace is necessary to identify what component has encountered a problem and where. One example would be a null-pointer exception.

- Indication Processing Facility (IPF) logging

  IPF logging is a logging mechanism that is used in IBM Cognos BI to record application activity and debugging information into a log file.

In addition, in some cases log files are generated after importing the Framework Manager model and the IBM InfoSphere Warehouse Cubing Services model into Cognos Cube Designer.

You can define the logging that you want by editing the appropriate configuration file.

Enabling debug console logging

To enable debug console logging in Cognos Cube Designer, complete the following steps:

1. Edit the `FMDesigner.ini` file.

   The file is located in the location where you installed Cognos Cube Designer in the directory `..\bin64\fmdesigner` if you are using the 64-bit version of Cognos Cube Designer or in the directory `..\bin\fmdesigner` if you are using the 32-bit version.

2. Enter the following text at the top of the file in this order:

```
-console
-debug
-log
```

3. Save `FMDesigner.ini`.

4. Restart Cognos Cube Designer.

Enabling IPF logging

To enable IPF logging in Cognos Cube Designer, complete the following steps:

1. Open **ipfCubeDesignerclientconfig.xml.sample** in the directory `..\configuration`.
2. Examine the categories.

   For each category that you want to enable, ensure that the category is not commented out. For example:

   ```
   <category ... >
   ...
   </category>
   ```

   For each category that you want to disable, comment it out. For example:

   ```
   <!--category ... >
   ...
   </category-->
   ```

   Table 6-1shows the logging categories defined in the `ipfCubeDesignerclientconfig.xml` sample file.

Table 6-1 IPF logging categories

Category	Description
Trace.fmeng	Enables all categories of logging. Overridden by explicitly disabling individual categories.
Trace.fmeng.memory	Memory statistics for the whole process (memory used, available memory, and free memory).
Trace.fmeng.platform	Platform-related messages, such as session management. All exceptions that are generated are logged to this category.
Trace.fmeng.metadata	Messages related to fetching metadata.
Trace.fmeng.import.cubingServices	Messages related to importing an IBM InfoSphere Warehousing Cubing Services cube. These are the same messages that appear in the text file that is generated after importing a cube.
Trace.fmeng.import.frameworkManager	Messages related to importing objects from the classic Framework Manager model.
Trace.fmeng.error	Messages related to errors. A call stack is generated.
Trace.fmeng.publish	Messages related to publishing models, starting cubes and view data, viewing members in the member folder, and so on.

3. Save the file with a new name: `ipfclientconfig.xml`.
4. Restart Cognos Cube Designer.

The IPF log results are stored in the `fmeng_trace.log` file in the `..\logs` directory that Cognos Cube Designer is installed in.

7

Administering dynamic cubes

You must configure and start dynamic cubes before you can run queries against them. Dynamic cubes must be administered over the time when they are in use. Common administration tasks include assigning the cube to the query service instance, starting it, monitoring its health, and refreshing its contents.

This chapter contains the following sections:

- Adding and removing cubes
- Starting, stopping, and monitoring cubes
- Managing the cache
- Setting up routing rules
- Setting up redundant cube instances
- Scheduling the refresh of the cache
- Administering cubes from command line

7.1 Adding and removing cubes

Dynamic cubes must be assigned to the query service before they can be loaded into memory and made available for queries.

7.1.1 Adding a cube to the server group

IBM Cognos instances are grouped in *server groups* to support redundancy and load balancing. Unless a server group is defined and a server instance is added to it, the server remains a member of the default server group.

For ease of use, cube management operations in IBM Cognos Administration are also geared towards server groups, allowing you to add cubes to all instances of query service of all servers in the group at once. You can also start, stop, and refresh all cube instances with a single administration command.

Depending on the preference of the administrator, the dynamic cubes information can be displayed with focus on server groups or on data stores. To switch between the two views in the IBM Cognos BI portal, use these steps:

1. Go to IBM Cognos Administration.
2. Select the Status tab.
3. Click **Dynamic Cubes**.
4. Select the view from the `Data Stores - (All)` list as shown in Figure 7-1.

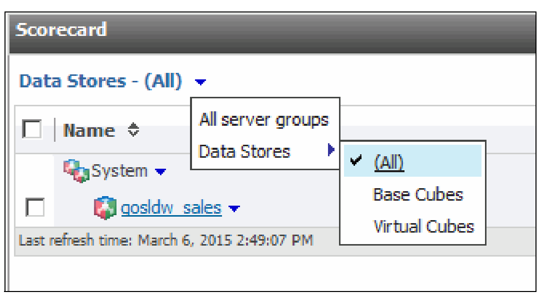


Figure 7-1 Selecting the view for dynamic cubes

To demonstrate cube administration functions, this chapter uses the `gosldw_sales` cube that was defined in Chapter 5, "Basic modeling" on page 105. Before the cube becomes available, it must be published from IBM Cognos Cube Designer as described in 5.8, "Deploying dynamic cubes for reporting and analysis" on page 154.

If you have defined security for the cube, it can be configured as described in 9.12, "Applying security views " on page 307.

To add the cube to the server group, complete these steps:

1. In the IBM Cognos BI portal, open **IBM Cognos Administration**. Select the **Status** tab and click **Dynamic Cubes** (see Figure 7-2).

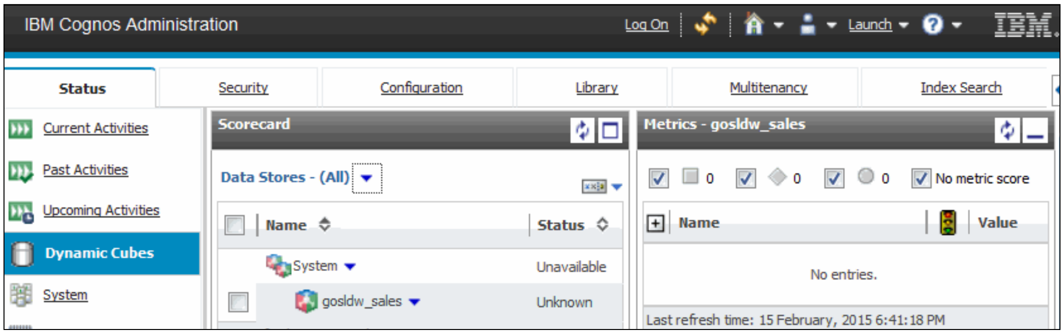


Figure 7-2 Dynamic Cubes administration

2. From the Scorecard pane, select **Add data store to server group** from the drop-down list next to the cube you want to add as shown in Figure 7-3.

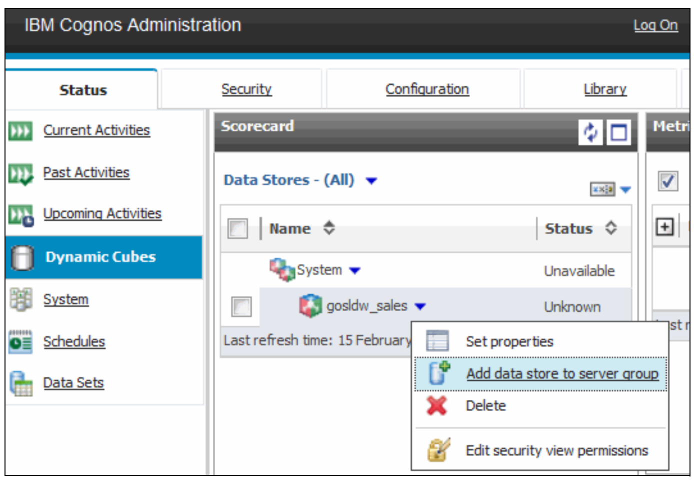


Figure 7-3 Adding a cube to the server group

3. The Add data store to server group window is displayed with a menu for you to select the server group to add the cube to. Select the server group and click **OK** as shown in Figure 7-4. Then click **OK** in the *View the results* window.

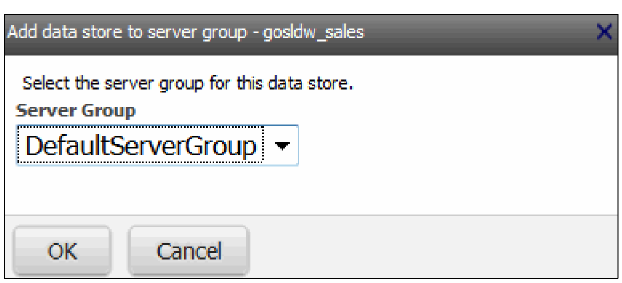


Figure 7-4 Server group selection dialog

4. The cube name becomes a link as shown in Figure 7-5. It is now added to all servers in the server group. All instances of the cube can be managed together from the menu next to the cube name. Click the cube name to see instances of this cube in different server groups and on different servers, and manage them individually.

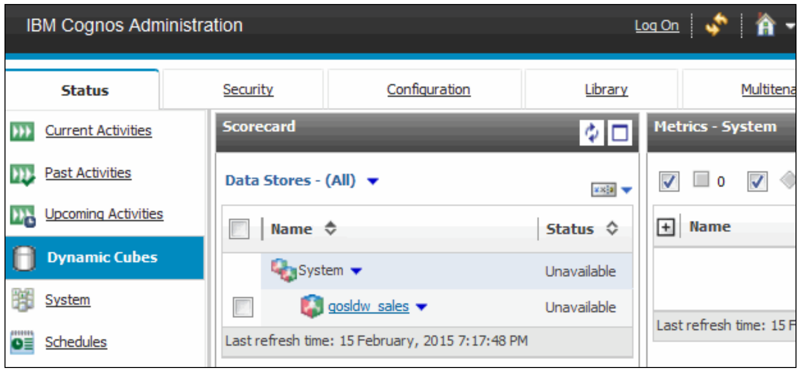


Figure 7-5 Cube added to server group

> **Note:** To see the view update in a timely manner, click the **Refresh** icon in the upper right corner of the view. It might take several seconds for the cube addition to take effect in the query service.

When the cube is added to the query service, in most cases it can work with the default configuration settings. The cube configuration can be changed on the individual cube instance configured for a query service by following these steps:

1. In IBM Cognos Administration, select the Status tab and click **Dynamic Cubes**.
2. Drill down to the cube instance to get to the query service instance.
3. Select **Set properties** from the drop-down list next to the cube name as shown in Figure 7-6.

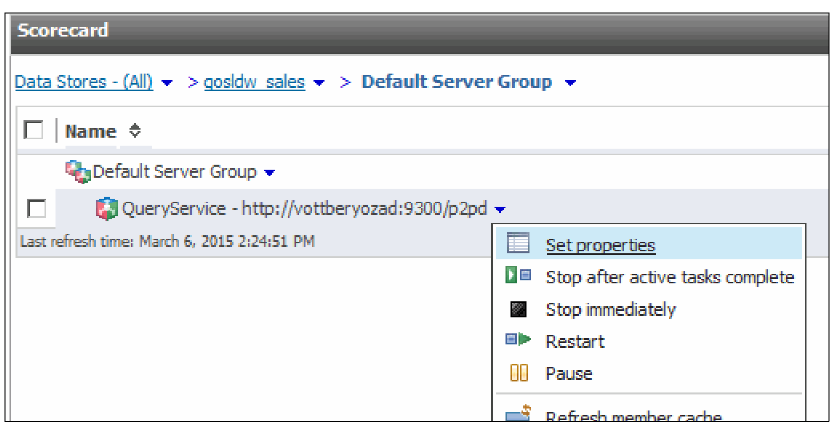


Figure 7-6 Set properties command for the cube

4. A cube configuration properties window opens where you can change the settings as shown in Figure 7-7. Change the settings as needed and click **OK** to finish.

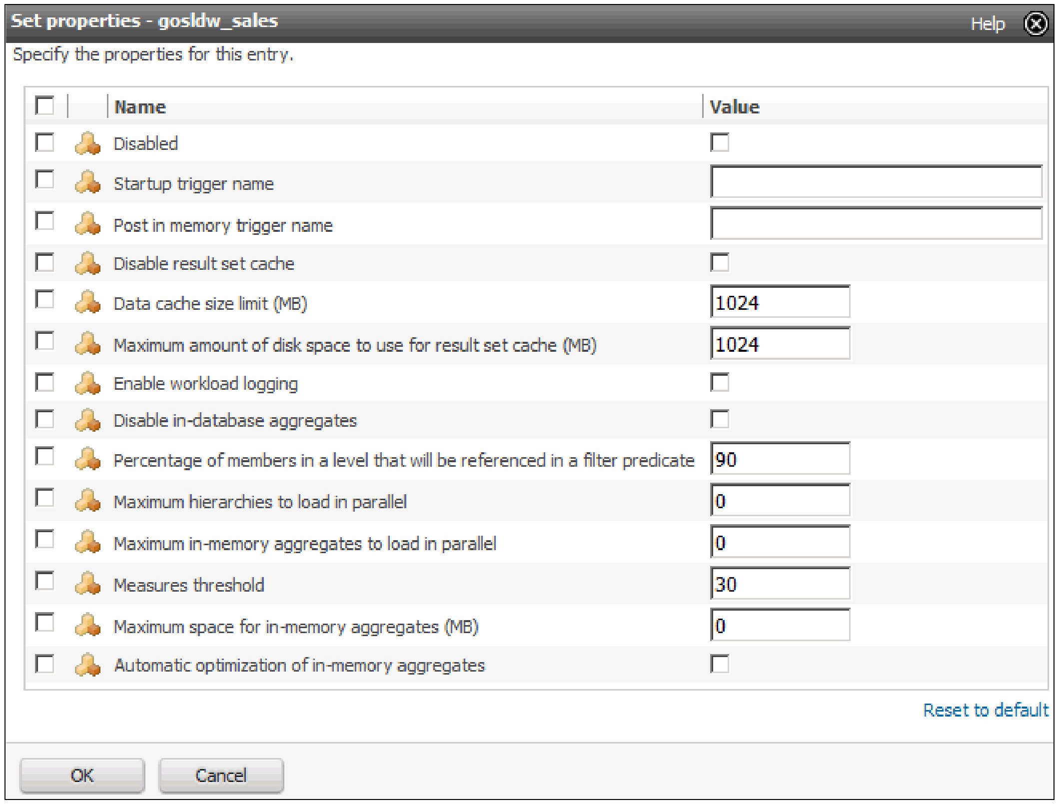


Figure 7-7 Cube configuration properties

The following configuration settings are available for the cube. Some of them do not apply to virtual cubes and will not be displayed for such cubes.

- **Disabled:** Controls whether the cube should be enabled or disabled.
- **Startup trigger name:** Name of the trigger to run when the cube starts.
- **Post in memory trigger name:** Name of the trigger to run when the in-memory aggregates finish loading.
- **Disable result set cache:** Controls whether to enable result set cache.
- **Data cache size limit (MB):** The maximum size of the cache that contains cell values.
- **Maximum amount of disk space to use for result set cache (MB):** Result set cache size limit.
- **Enable workload logging:** Controls whether to log workload information for the aggregate advisor.
- **Disable in-database aggregates:** Controls whether to enable in-database aggregates.
- **Percentage of members in a level that will be referenced in a filter predicate:** Limits the percentage of members to be referenced explicitly in the filter expression beyond which they will be replaced with a level reference. Value of zero means no limit.
- **Maximum hierarchies to load in parallel:** Limits the number of threads to be allocated for loading hierarchies. If the value is 0 (zero), the number of threads calculated is twice the number of CPU cores.

- **Maximum in-memory aggregates to load in parallel:** Limits the number of threads to be allocated for loading in-memory aggregates. If the value is 0 (zero), the number threads is twice the number of CPU cores.
- **Measures threshold:** Percentage of members that are referenced in the query beyond which all measures will be retrieved.
- **Maximum space for in-memory aggregates (MB):** Limits the space available for storing in-memory aggregates.
- **Automatic optimization of in-memory aggregates:** Controls whether in-memory aggregates should be optimized automatically.

7.1.2 Removing a cube from the query service

> **Important:** Before removing the cube from query service, stop the cube first.

Removing a cube from the query service is similar to the steps for adding a cube:

1. In IBM Cognos BI portal, go to IBM Cognos Administration. On the Status tab, click **Dynamic Cubes**.
2. From the list associated with the cube, select **Remove data store from server group** to delete the cube as shown in Figure 7-8.

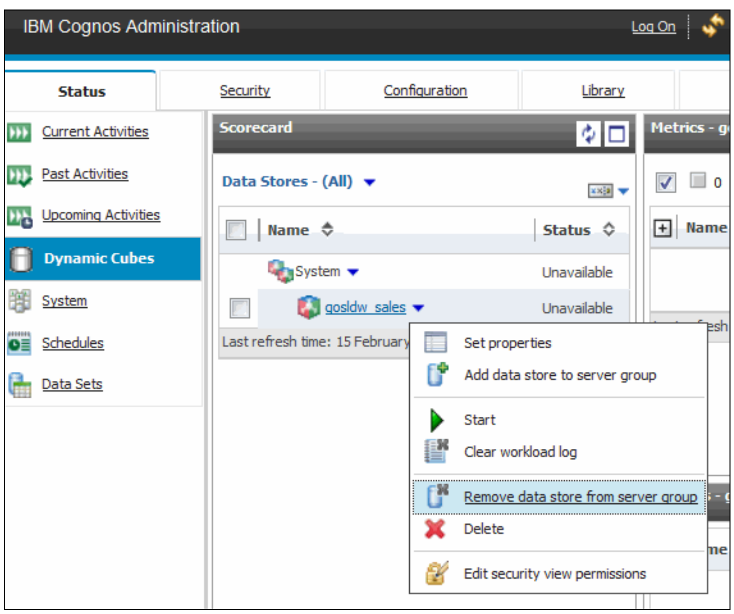


Figure 7-8 Removing data store from server group

3. In the next windows, you are prompted to select the server group to remove the cube from, and to confirm the operation.
4. When the cube is removed, the link associated with the cube name disappears and the cube is no longer associated with servers in the server group.

7.2 Starting, stopping, and monitoring cubes

A number of administration commands are available for managing the cubes:

- **Start:** Described in 7.2.1, “Starting a cube” on page 205.
- **Stop after active tasks complete:** Described in 7.2.2, “Stopping a cube” on page 208.
- **Stop immediately:** Described in 7.2.2, “Stopping a cube” on page 208.
- **Restart:** Run the command **Stop** after active tasks complete followed by a **Start** command.
- **Pause:** Described in 12.3, “Applying near real-time updates” on page 436.
- **Refresh member cache:** Described in 7.3, “Managing the cache” on page 211.
- **Refresh data cache:** Described in 7.3, “Managing the cache” on page 211.
- **Refresh security settings:** Described in 9.11, “Publishing and starting the cube” on page 307.
- **Clear workload log:** Described in 11.3.4, “Running the Aggregate Advisor wizard” on page 350.
- **Incrementally update data:** Described in 12.3, “Applying near real-time updates” on page 436.

7.2.1 Starting a cube

Before a cube can be used, it must be started. Use the following steps to start a cube:

1. From the drop-down list next to the cube name select **Start** (see Figure 7-9). A window dialog opens to confirm the operation. Click **OK**.

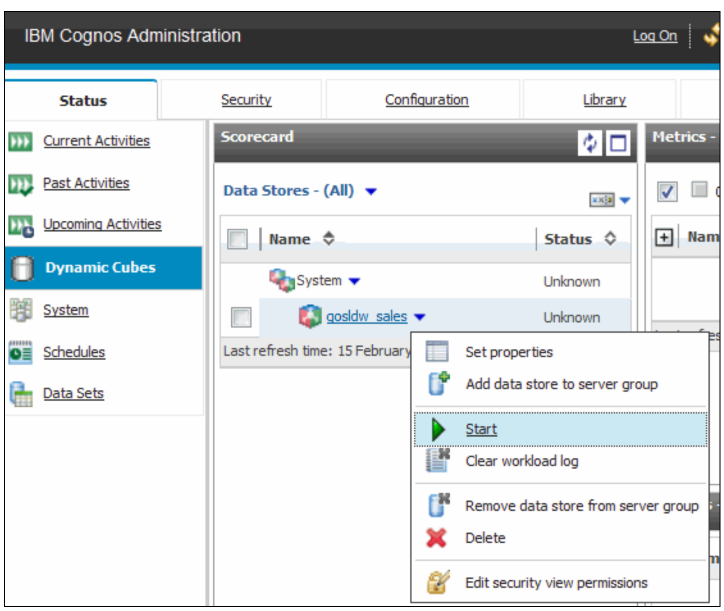


Figure 7-9 Starting the cube

2. When the command is running, a window opens as shown in Figure 7-10.

> **Note:** This window indicates that the command was successfully submitted, but does not mean that it has finished successfully.

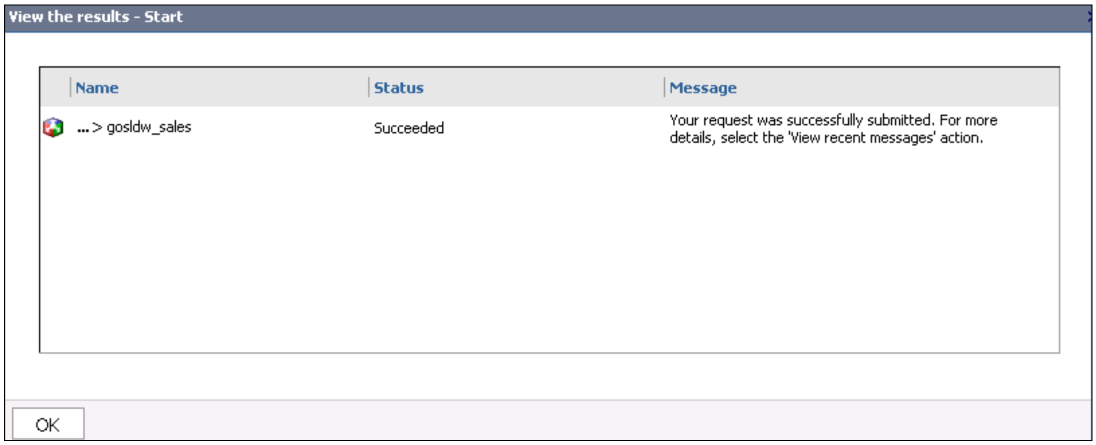


Figure 7-10 Cube started notification window

3. When the cube starts successfully, its status changes to **Available** as shown in Figure 7-11.

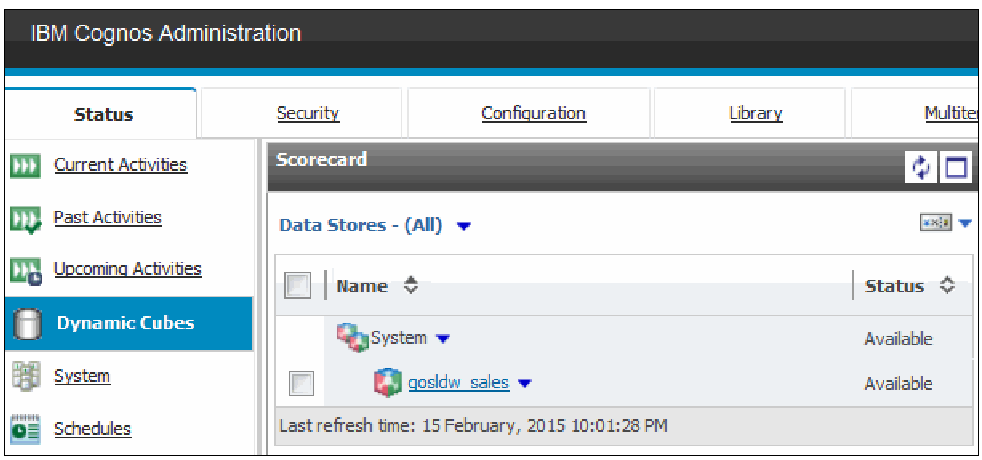


Figure 7-11 Cube state turns to Available after the cube starts successfully

4. If the cube start is not successful, or additional information is needed about the command state, use the **View recent messages** command to review details about the state of the cube. Click the cube name to open the server group view as shown in Figure 7-12.

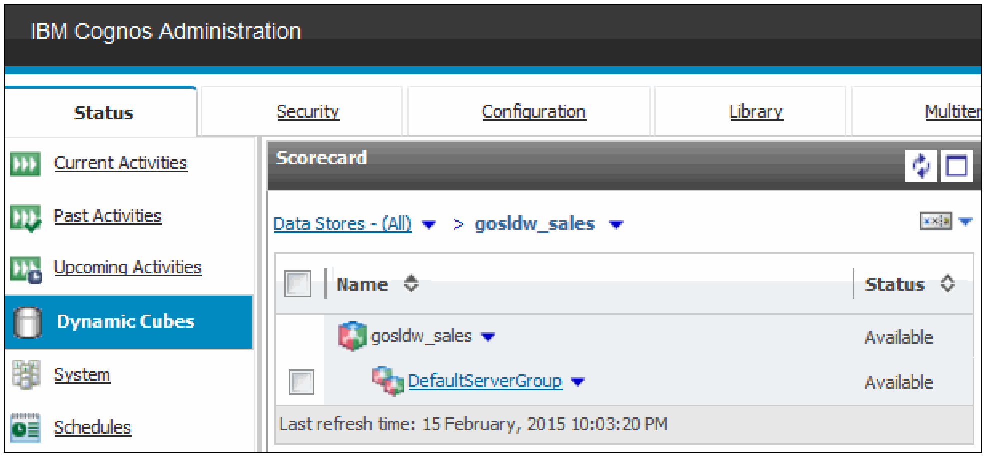


Figure 7-12 Displaying the list of server groups the cube belongs to

5. Click the server group name to display a list of cube instances under that server group. Click the drop-down menu next to the cube instance name that you want and select **View recent messages** as shown in Figure 7-13.

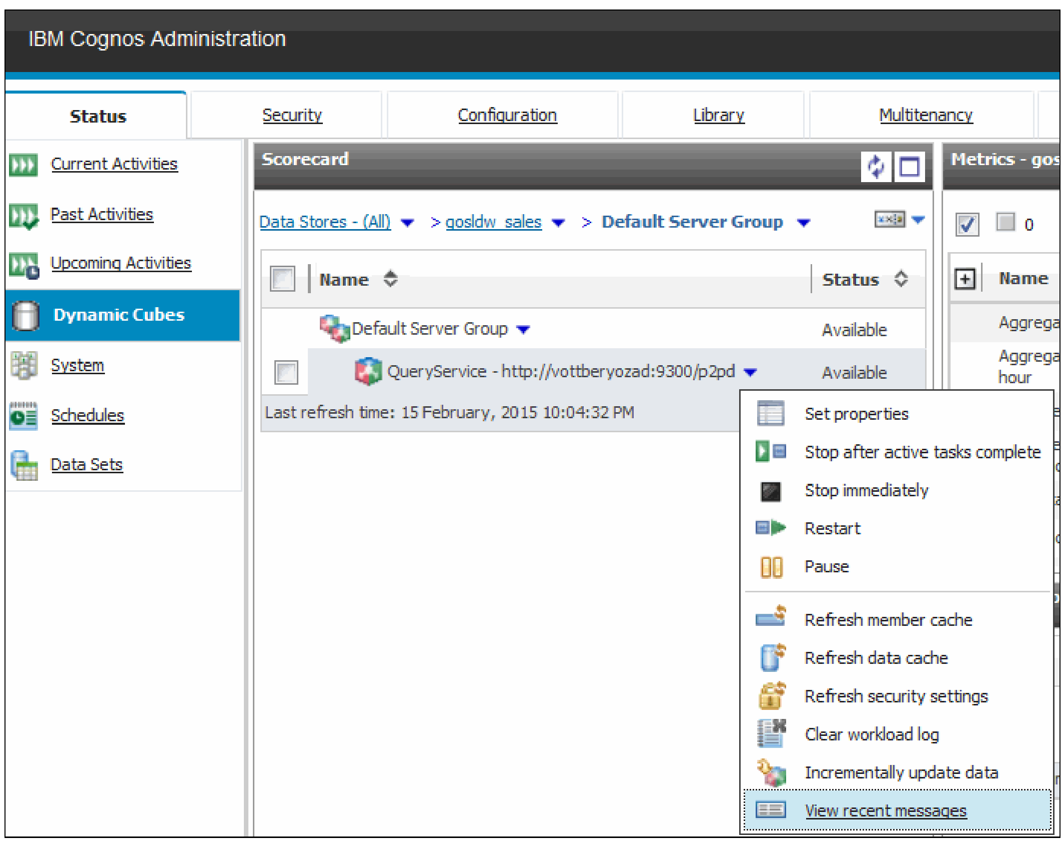


Figure 7-13 Selecting "View recent messages" command

6. A log of recent messages is displayed showing the result of the cube start command, and any warning or error messages as shown in Figure 7-14.

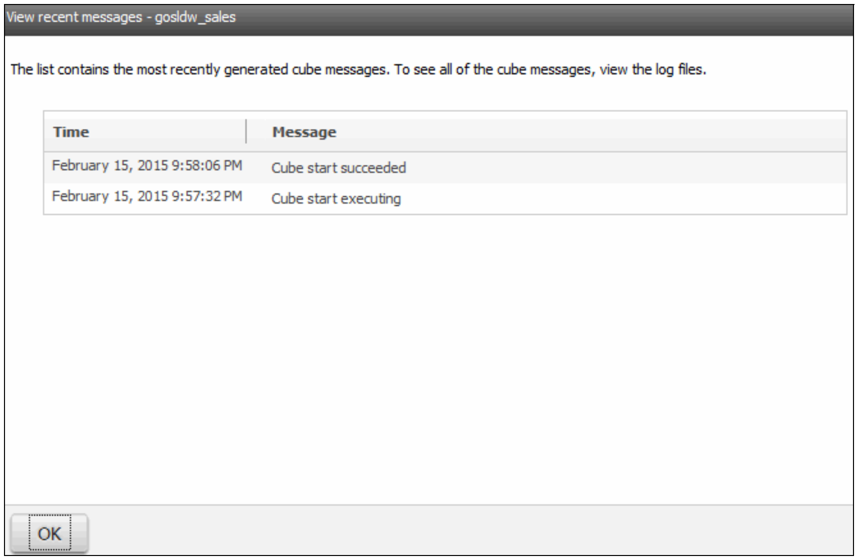


Figure 7-14 Recent messages log for the cube start command

7.2.2 Stopping a cube

To stop a cube, select one of the following commands from the drop-down list next to the cube name:

- **Stop after active tasks complete**: The cube stops after the queries that are currently running finish.
- **Stop immediately**: The cube stops immediately, without waiting for the active queries and commands to complete. Some user queries can fail as a result. This command is not supported for virtual cubes.

7.2.3 Monitoring cube state through metrics

When managing dynamic cubes, a good practice is to monitor metrics that are displayed for each cube in the Metrics window. Metrics are available for individual cube instances. To display metrics, navigate to the cube instance view as you did for the View recent messages command in step 5 on page 207.

Figure 7-15 shows the cube metrics view.

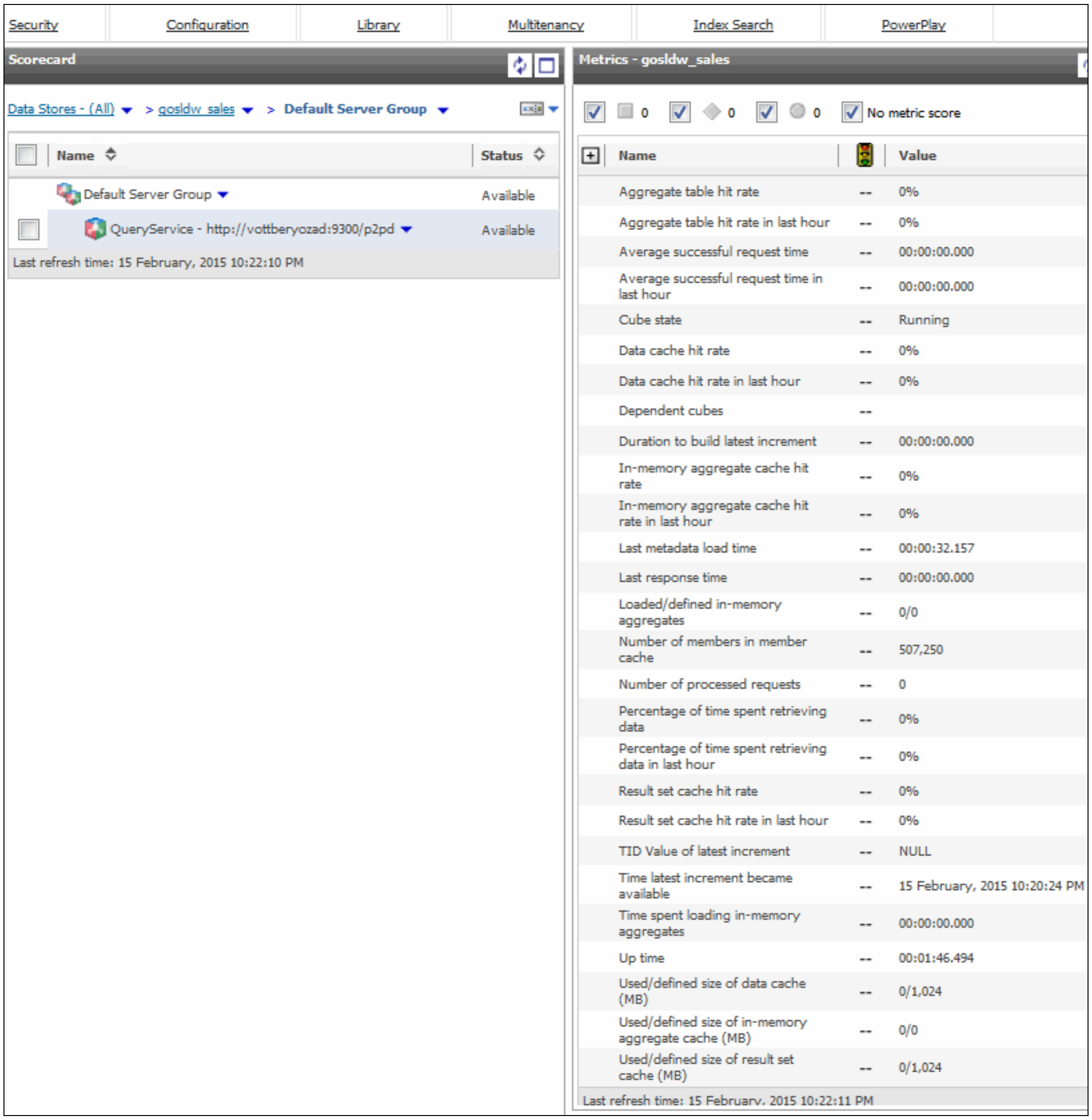


Figure 7-15 Cube metrics view

In addition to overall values, some metrics also have the value collected in the last hour, which helps you track both historic averages and recent changes to the system.

The following metrics are available:

- Aggregate table hit rate

  A higher cache-hit rate indicates better utilization of the aggregate table and better query performance.

- Average successful request time

  This metric indicates the average time for a report to run successfully. It is useful when monitoring the performance of the server and database. Slow request times might indicate performance issues that must be investigated.

- Cube state

  This metric indicates the current cube state: Disabled, Stopped, Starting, Running, Stopping, Pausing, and Paused.

- Data cache hit rate

  A higher cache-hit rate indicates better utilization of the data cache and better query performance.

- Dependent cubes

  Describes current dependency relationships between source and virtual cubes.

- Duration to build latest increment, TID Value of latest increment, and Time that latest increment became available

  These metrics support monitoring of increment loading, the time it took to load the latest increment, the TID of the last increment ID, and the time the latest increment became available.

- In-memory aggregate cache hit rate

  A higher cache-hit rate indicates better utilization of in-memory aggregates and better query performance.

- Last metadata load time

  This metric shows how long the previous load took for the metadata to be loaded or refreshed. Disproportionately long times might indicate modeling or configuration problems.

- Last response time

  This metric shows how long the most recent request took to run any request against this cube. Useful for detecting performance degradation problems.

- Loaded/defined in-memory aggregates

  Reflects the total number of defined in-memory aggregates for this cube, and how many of them have already been loaded. Because the loading of aggregates sometimes takes a long time, it can continue after the cube starts without blocking cube operations.

- Number of members in member cache

  Total number of members in all hierarchies in the cube. This metric serves as a measure of cube size and can help to estimate hardware requirements for the cube.

- Number of processed requests

  A running count of requests that were issued for this cube. It helps gauge how busy the cube is.

- Percentage of time spent retrieving data

  This metric indicates that the percentage of time data is retrieved from the database. Higher percentage means worse performance and fewer cache hits. However, this situation should be weighed relative to the `Average successful request time`. If queries are fast enough to satisfy service level agreements (SLAs), then a higher value of this metric might not be important.

- Result set cache hit rate

  A higher hit rate means better result cache use and indicates that most of the report results can be reused, improving query times. A low hit rate of this cache is typically not an area of concern because result set reuse was introduced as an optimization in situations where reports are rerun by users with the same security privilege.

- Time spent loading in-memory aggregates

  In-memory aggregates continue loading after the cube start, and this metric tracks the total time that it took to load them.

- Up time

  This metric indicates the length of time the cube is running.

- Used/defined cache sizes

  Cache usage and defined size values are displayed for data cache, in-memory aggregate cache, and result set cache.

7.3 Managing the cache

Dynamic Cubes support two types of caches that can be managed by the administrator:

- Member cache

  This cache contains cube members that are loaded from the relational data source. The member cache can be refreshed when appropriate, such as when the source data is changed, to update the cube with the latest metadata.

- Data cache

  This cache contains data values that correspond to the current set of cache metadata. This cache can be refreshed when the data values in the source relational data source are changed. In general, data values change more frequently than cube metadata.

Use the following steps to manage the cache:

1. Refresh the member or data cache by selecting **Refresh member cache** or **Refresh data cache** from the cube drop-down list as shown in Figure 7-16.

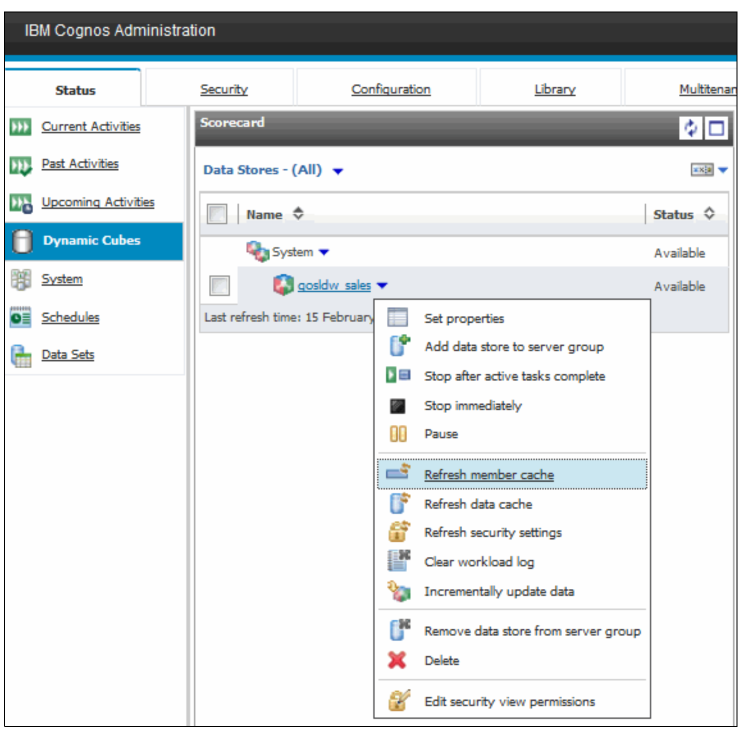


Figure 7-16 Refreshing cube cache

2. A window is displayed confirming that the command was run. Click **OK**.
3. Select **View recent messages** on a cube instance (as described in step 5 on page 207) to check the progress of the refresh. The window indicates either a success message or a failure message, as shown in Figure 7-17.

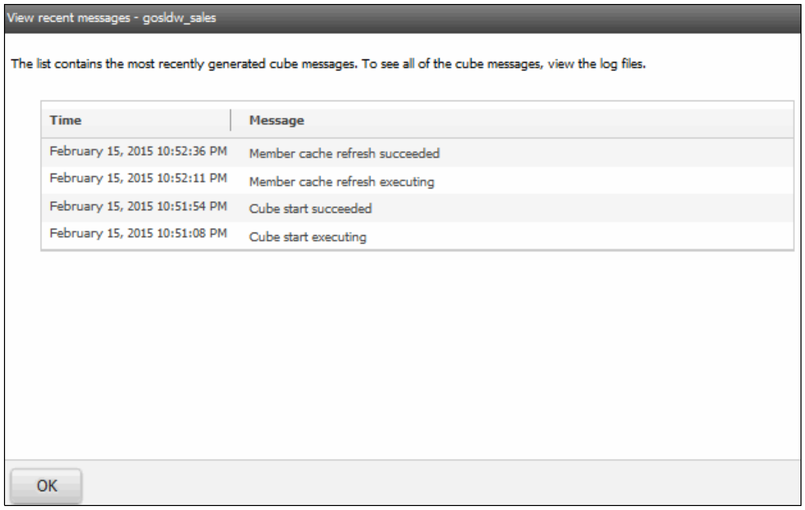


Figure 7-17 Member cache refresh messages

> **Note:** Because data cache is closely associated with the member cache, refreshing the member cache automatically causes the data cache to be refreshed.

7.4 Setting up routing rules

If you are setting up your IBM Cognos BI deployment to address applications that are sourced from dynamic cubes and other sources in a mixed environment, you must create server groups and dispatcher routing rules to ensure that queries are routed to the correct report server. The query service that has a dynamic cube enabled is where the metadata and other cached data items are located. If the dynamic cube application queries are sent to the wrong report server, an error message is returned rather than the requested data.

For more information about setting up server groups and routing rules, see the *Advanced Dispatcher Routing* topic in the *IBM Cognos Business Intelligence Version 10.2.2 Administration and Security Guide* that can be downloaded from:

http://www-01.ibm.com/support/docview.wss?uid=swg27042003

7.5 Setting up redundant cube instances

To enable load balancing and optimize query performance in high-load environments, you can set up multiple instances of the dispatcher with identical cube configurations.

When the query is submitted by the user, it is routed to the next available dispatcher, making sure that all dispatchers are load-balanced. As your requirements change, you can add or remove dispatchers from the server group to match your performance needs.

Because queries are routed to different dispatchers based on load, you must ensure that all instances of the dynamic cube on all dispatchers are operating correctly. If a query is routed to a dispatcher where a cube is not available, the query fails. In such cases, the queries can appear to fail randomly, complicating problem diagnostics.

To avoid disrupting query execution, adding and removing cube instances must be done in a controlled way. Before making the instance of the cube in a particular dispatcher available to the server group, ensure that this dispatcher is *not a member of the server group*. This way, no queries can reach this dispatcher during addition or removal, which eliminates the risk of disrupting query execution.

After the cube instance is configured and is running, use the following steps to add the dispatcher instance to the server group:

1. In IBM Cognos Administration, select the Status tab and click **System** (see Figure 7-18).

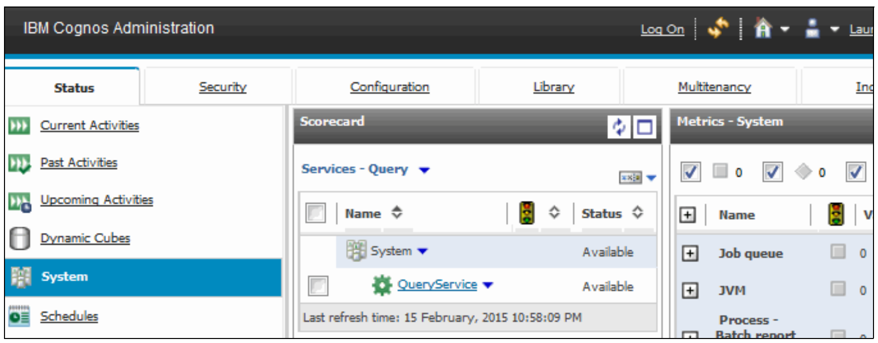


Figure 7-18 System view

2. In the Scorecard pane, click the name of the server that holds the dispatcher instance that you are configuring. Click the drop-down menu next to the dispatcher address and select **Set properties** (see Figure 7-19).

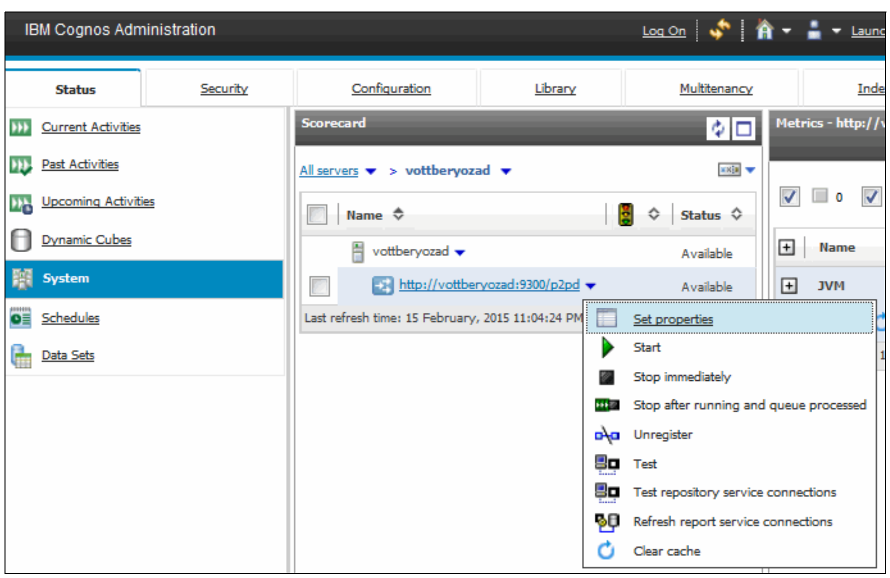


Figure 7-19 Dispatcher Set properties command

3. In the Properties window, select the Settings tab, and scroll down to the `Server group` field (Figure 7-20).

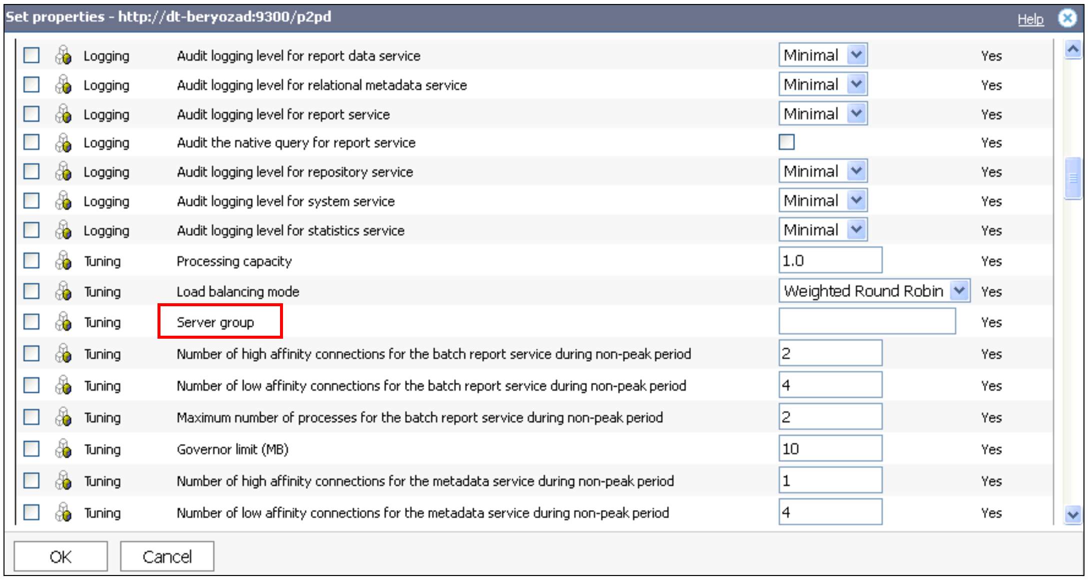


Figure 7-20 Dispatcher properties

4. Enter the name of the server group you want this dispatcher to be a part of, and click **OK**. The dispatcher will now be able to process queries that are routed to the server group.

Note: Propagation of information about dispatcher membership in server groups to all affected dispatchers is not instantaneous. You might have to wait up to a minute for the changes to take effect.

The process of removing a dispatcher from the server group is the reverse of the procedure that is described.

Clearing the server group name field in the dispatcher configuration prevents queries for the cubes in that server group from being routed to it. The cube or the dispatcher itself can then be administered or taken down for maintenance.

For more information about server groups and routing tables, see the *Advanced Dispatcher Routing* topic in the *IBM Cognos Business Intelligence Version 10.2.2 Administration and Security Guide* that can be downloaded from:

http://www-01.ibm.com/support/docview.wss?uid=swg27042003

7.6 Scheduling the refresh of the cache

Administration commands often need to be run at predefined times and intervals. For example, the data source from which the dynamic cube is built might be continuously updated, so the cube member and data caches must be updated also.

As an example, schedule a refresh of a cube metadata to run once a day:

1. In IBM Cognos Administration, select the Configuration tab, and click **New Query service administration task** to open a menu. From this menu, you can manage various tasks. Select **Dynamic cube** to schedule a dynamic cube command (see Figure 7-21).

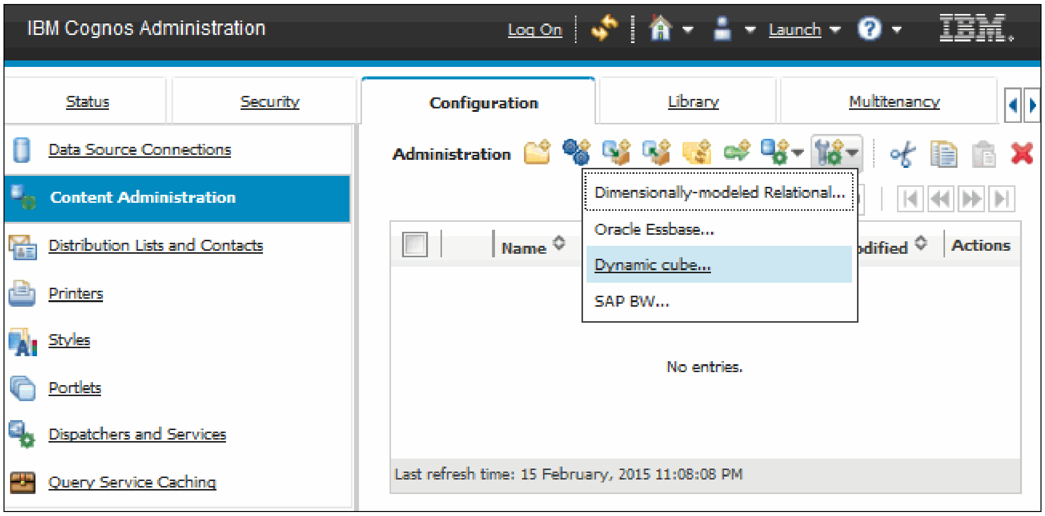


Figure 7-21 Dynamic cube task-scheduling command

2. In the dialog that opens, enter the name for the cache update task that you are creating, for example, `Cache refresh task` (see Figure 7-22).

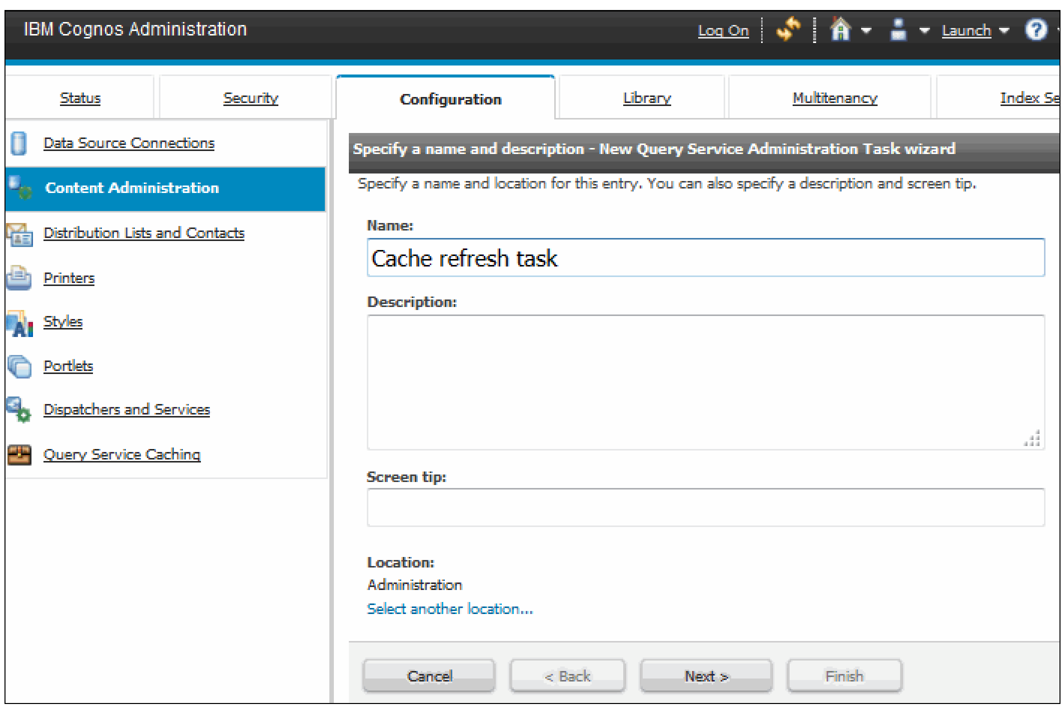


Figure 7-22 Task name dialog

3. Click **Next**.

4. In the next window, select **Refresh member cache** as the command in the **Operation** field, and select the check box next to the name of the cube that you want to refresh (see Figure 7-23).

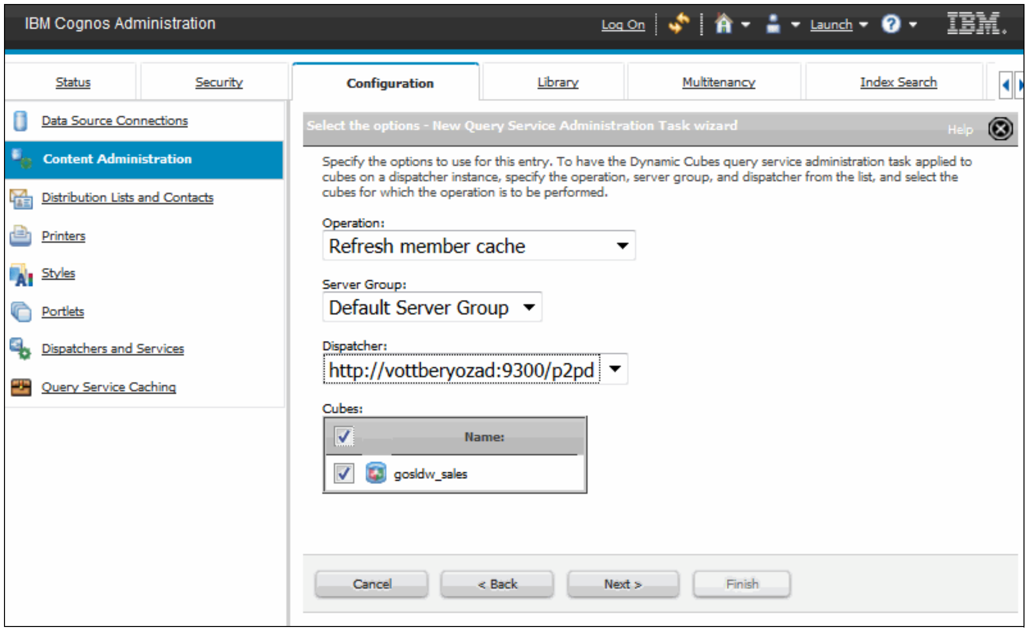


Figure 7-23 Task type and cube selection

5. Click **Next**.
6. In the next window, select **Save and schedule** and click **Finish** to create the task (see Figure 7-24).

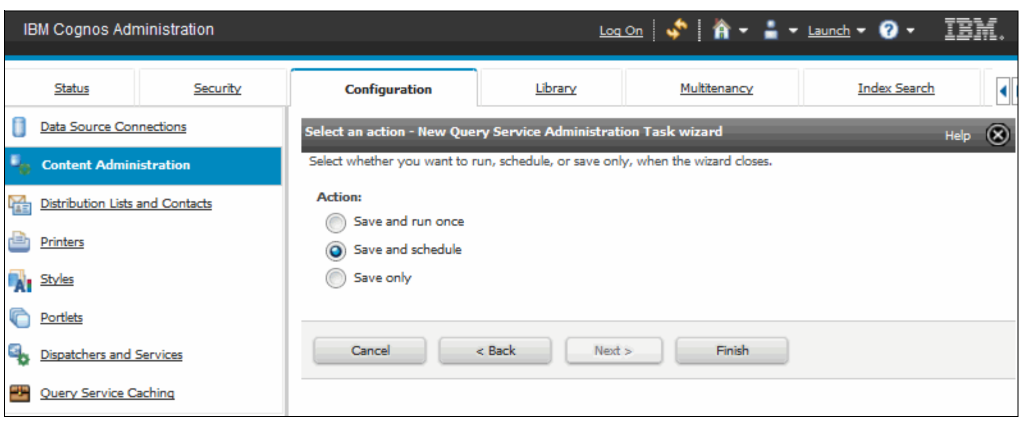


Figure 7-24 Saving the task

7. In the new window that opens, define the schedule to specify when to run this task (see Figure 7-25). Select the By Day tab, select **Every 1 day(s)**, and then select **Daily Frequency**. Enter a time period during which you want this task to run. Change the **Start** field to specify a time in the future, for example, the beginning time of the task.

> **Note:** Because administrative commands can increase load on the dynamic cube, a common practice is to schedule maintenance tasks during a time when the cube is not busy, such as late at night.

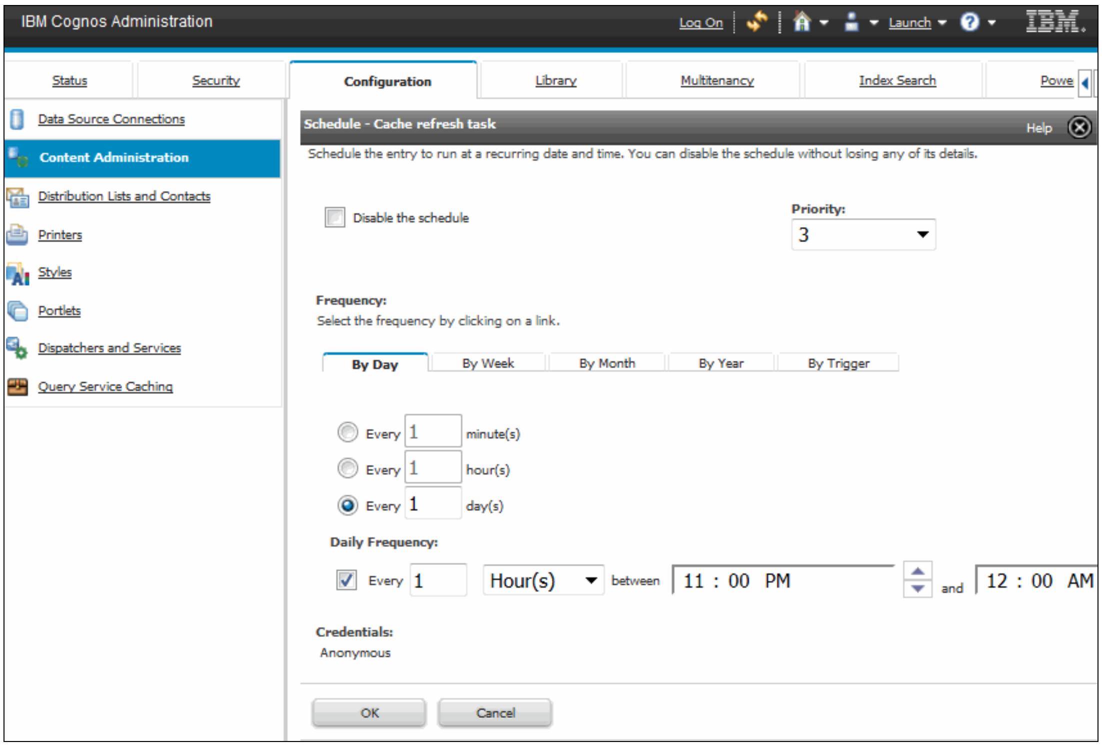


Figure 7-25 Schedule definition

8. Click **OK**.
9. The next window displays the currently scheduled tasks (see Figure 7-26).

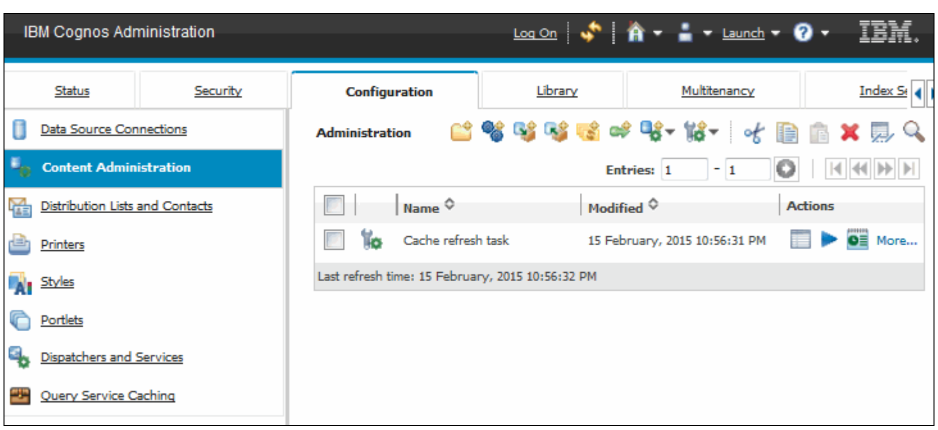


Figure 7-26 Content Administration window displaying the currently scheduled tasks

10. Wait for the time of the scheduled task to pass, and click **More** next to the task. A window that lists available task actions opens (see Figure 7-27).

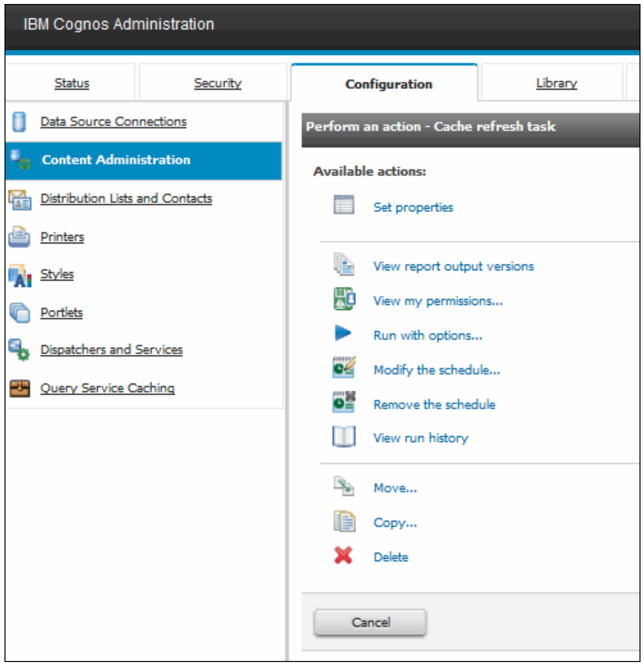


Figure 7-27 Task actions window

11. From the list of available commands, select **View run history**. The record of when the task was last run opens and shows the status (see Figure 7-28).

Figure 7-28 Task execution history

12. Click **Close**.

7.7 Administering cubes from command line

In IBM Cognos 10.2.1, Fix Pack 3 added several capabilities including the DCAdmin tool. For more information, see *Enhancements in Cognos BI 10.2.1 Fix Pack 3* at:

http://www-01.ibm.com/support/docview.wss?uid=swg21660073

DCAdmin is a command-line tool that queries and manages dynamic cube instances. It can be used in cases where automation outside of the IBM Cognos system is required, and it is a simple substitute for developing an IBM Cognos software development kit (SDK) application.

This tool can be found in the product `bin` folder in 32-bit install and `bin64` in 64-bit install. It is called `dcadmin.bat` (on Windows) and `dcadmin.sh` (in UNIX).

The tool supports a number of cube administration commands as shown in Example 7-1. The syntax information can be obtained by running the tool without parameters.

By default the assumption is that the IBM Cognos instance runs on the server where the command is run. If that is not the case, the server address and port can be changed by using command-line options `-s` and `-p`.

Example 7-1 DCAdmin tool command syntax

```
C:\cognos10\bin>dcadmin.bat
Usage:

 dcadmin[.bat|.sh] [-p port] [-s server] [-x output_file] [-l "namespace,userid,password"]
[-arg argName argValue] command [cube0 cube1 ...]

 -p port - Specify the port to use. Default is 9300.
 -s server - Specify the server name to use. Default is "localhost."
 -x output_file - Specify the name of the output file to write the structured
      command result to
 -v - Enable verbose output
 -l "namespace,userid,password" - Specify the BI login parameters
      For example: -l "LDAP,admin,secret123"
 -arg argName argValue - Specifies command arguments.
        Currently supported arguments:
          transactionID - Optional argument to command incrementallyLoadCubes

Note that any parameter containing a comma or a space must be included in double quotes,
e.g. "param1,param2"

Available commands:
  getCubeState
  getCubeMetrics
  startCubes
  forceStartCubes (internally invokes SDK call "startCubes" with parameter
"startROLAPCubesAndSourceCubes")
  stopCubes
  forceStopCubes (internally invokes SDK call "stopCubes" with parameter
"stopROLAPCubesImmediately")
  restartCubes
  pauseCubes
  incrementallyLoadCubes
  refreshCubeDataCache
  refreshCubeMemberCache
  refreshCubeSecurity
  clearCubeWorkloadLog

Script returns exit code 0 when command was successful, and code 1 in case of an error
```

The following example demonstrates how to use the DCAdmin tool to issue an administration command from command line. In this example, the command issued is Refresh member cache (**refreshCubeMemberCache**) for cube `gosldw_sales` for Windows. You use similar steps on UNIX.

Perform the following steps:

1. Open the application command-line interface and change the current folder to the `bin` folder of the IBM Cognos instance.
2. Ensure that the `gosldw_sales` cube is running:

   `dcadmin.bat getCubeState`

   Example 7-2 shows the command output.

   Example 7-2 Output of command to check cube states

   ```
   C:\cognos10\bin>dcadmin.bat getCubeState
   Running command 'getCubeState' on cube(s) []
   Finished running command 'getCubeState' on cube(s) []

   Cube states:
          gosldw_sales : running
   ```

3. If the cube is not running, start it:

   `dcadmin.bat startCubes gosldw_sales`

4. Refresh the cube member cache:

   `dcadmin.bat refreshCubeMemberCache gosldw_sales`

   Example 7-3 shows the command output.

   Example 7-3 Output of command to refresh cube member cache

   ```
   C:\cognos10\bin>dcadmin.bat refreshCubeMemberCache gosldw_sales
   Running command 'refreshCubeMemberCache' on cube(s) [gosldw_sales]
   Finished running command 'refreshCubeMemberCache' on cube(s) [gosldw_sales]

   gosldw_sales : SUCCESS : Execution of the command "member cache refresh" on the
   cube "gosldw_sales" succeeded.
   ```

As shown in Example 7-3, the command is successful and the member cache of the cube `gosldw_sales` is now refreshed.

> **Tip:** In cases where greater control over dynamic cube administration functionality is required, consider implementing an application that takes advantage of IBM Cognos SDK.

Note: The instructions in this chapter follow the preferred way to administer dynamic cubes. There is an alternative way to achieve the same results (which was also available in earlier versions of the product):

- To add and remove cubes from the query service, and modify cube configuration properties, perform the following steps:
  a. In the IBM Cognos BI portal, go to IBM Cognos Administration. On the Status tab, click **System**.
  b. From the Scorecard view, select **All servers** → **Services** → **Query**.
  c. Select **Set properties** from the drop-down menu next to QueryService. To edit cube assignments and properties, click **Edit** in the Value column for dynamic cube configurations**.**
- To run cube administration commands and monitor cube metrics, complete the following steps:
  a. In IBM Cognos BI portal, go to IBM Cognos Administration. On the Status tab, click **System**.
  b. From the Scorecard view, select **All servers** → **Services** → **Query**.
  c. Click **QueryService** to show the current list of configured cubes.
  d. Click the cube name to monitor its metrics in the Metrics pane.
  e. Click the drop-down menu next to the cube name to display the list of available administration commands for that cube.
- To edit security view permissions, perform the following steps:
  a. In the IBM Cognos BI portal, go to IBM Cognos Administration. Select the Configuration tab and click **Data Source Connections**.
  b. Find the cube name in the list of data sources and click it to go to the cube model.
  c. Click **model** to show the list of security views.
  d. Click the **Set properties** icon to the right of the security view name to give access to that view to groups and users.

8

Virtual cubes

Virtual cubes enable reporting applications to use multiple fact tables.Virtual cubes can be used to improve the response time of the cube server queries by using efficient data partitioning for optimum cache utilization. Virtual cubes also enable merging sales numbers with currency exchange rates to provide a global view of the business.

This chapter explains how to create virtual cubes, how they work, and how to manage them.

This chapter contains the following sections:

- Overview of virtual cubes
- Creating virtual cubes
- Common use cases for virtual cubes
- Managing virtual cubes
- Virtual cube considerations

8.1 Overview of virtual cubes

A virtual cube is defined by merging two existing cubes. You can use virtual cubes for these purposes:

- Join two cubes with the same structure to facilitate data management.
- Join two cubes that share only some dimensions to make calculations on the shared data.
- Create multi-fact applications.
- Merge two base cubes, two virtual cubes, or one virtual cube and one base cube. The resulting virtual cube can, in turn, be merged with other source cubes into another virtual cube.

You can include dynamic cubes from the current project, dynamic cubes or virtual cubes deployed as data sources to the content store, or a combination of both. This approach allows you to select a cube that was developed in a different project.

When a report is created and run using a virtual cube, the queries are sent to the source cubes and the result sets merged.

Note: The two cubes that are merged can belong to separate data sources.

For example, Figure 8-1 shows cubes that are combined to form virtual cubes. Cube A and Cube B are merged to form Virtual Cube AB. Virtual Cube AB and Cube C are then merged to form Virtual Cube ABC. Cube A, Cube B, and Cube C are base cubes. Cube A and Cube B are the source cubes of Virtual Cube AB. Virtual Cube AB and Cube C are the source cubes of Virtual Cube ABC.

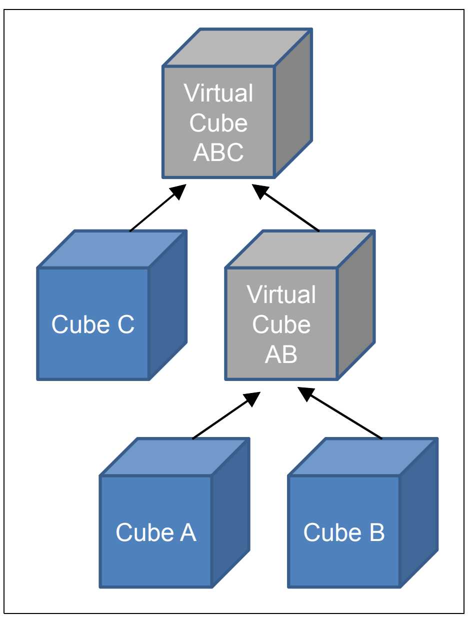


Figure 8-1 Sample virtual cube

When you create a virtual cube, it attempts to auto-merge dimensions with the same name from the source cubes.

During the creation of the virtual cube, within a merged dimension, the source hierarchies auto-merge if they have the same name.

In a merged hierarchy, the levels are merged based on the order of their appearance in the hierarchy, regardless of their name.

Figure 8-2 illustrates a virtual cube in which dimensions that have the same name are auto-merged.

New Virtual Cube		gosldw_sales	gosldw_target
Virtual Dimensions		Dimensions	
Branch		Branch ...	...
Employee by region		Employee by region ...	Employee by region ...
Order method		Order method ...	...
Product brand		Product brand ...	...
Products		Products ...	...
Promotions		Promotions ...	...
Retailers		Retailers ...	Retailers ...
Time		Time ...	...
Time (close date)		Time (close date) ...	...
Time (ship date)		Time (ship date) ...	...
Employee by manager		Employee by manager ...	...
Sales order		Sales order ...	...
Time to month		...	Time to month ...
Product type		...	Product type ...

Figure 8-2 Mix of auto-merged and unmerged dimensions in a virtual cube

At run time, the virtual cube matches the keys of members in the two source cubes and generates one virtual member for each pair. Any unpaired member is also included, but it is only available for measures from its source cube.

You can manually merge objects if the auto-merge does not produce the wanted results. For example, if the objects are named differently but represent conceptually related objects, manually merge the objects. The sample model provides an example that demonstrates this situation.

The sample dimensions Time and Time to month are related in the same way Products and Product type are related. Because they are named differently, they need to be manually merged. The `Time` dimension was created by manually merging `Time` from `gosldw_sales` and `Time to month` from `gosldw_target`. `Products` was created by manually merging `Products` from `gosldw_sales` and `Product type` from `gosldw_target`.

Figure 8-3 (which follows from Figure 8-2 on page 225) illustrates the result of manually merging dimensions into virtual dimensions.

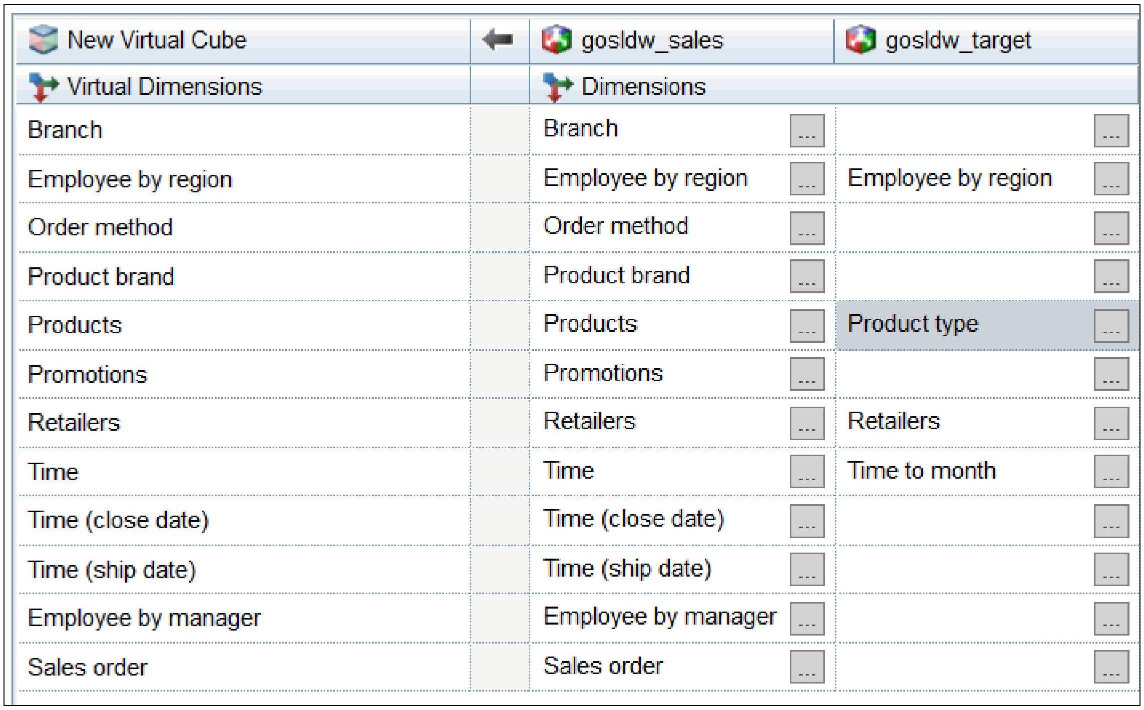


Figure 8-3 Virtual cube after manual merge of dimensions

You can create calculated measures in a virtual cube. You can create expressions that refer to source measures from different source cubes. For example, you could create a measure that calculates the variance of the sales facts from the planned sales values.

The security of the virtual cube is the merge of the security of the source objects. The set of visible members is a union of the members of the two cubes. A member that is denied access in one cube but is accessible in the other cube is visible in the virtual cube. This behavioris different from how the security filters and views are merged within a base cube. The member can only access the values of the source cube from which it is exposed.

A value that is accessible in either of the source cubes is accessible in the virtual cube.

8.2 Creating virtual cubes

This section describes how to create virtual cubes.

You can create virtual cubes from source cubes that exist in the same model as the virtual cube, from cubes that exist in the Cognos content store, or both.

8.2.1 Source object merging rules

Merging of the objects of the source cubes is done based on object names. For example, dimensions with the same name in both cubes are merged to become virtual dimensions. The dimensions of a cube that do not have the corresponding dimensions with the same name in the other cube are not merged together, but instead they are added to the virtual cube by themselves. All measures from the source cubes are included in the virtual cube.

Hierarchies and measures are merged automatically if their names are identical. When the names differ, two separate objects are added to the virtual cube.

Levels are merged based on their order in the hierarchies.

Members are merged if their key values are the same. For more information, see “Defining virtual members” on page 230.

A dimension does not have to be completely conformed to auto-merge successfully. In the sample model, there are two time dimensions, `Time to month` and `Time`. There are two time dimensions because they are modeled to deal with the different fact grains of the fact tables used in the two sample cubes. If `Time to month` existed in another namespace, it could also be named `Time`. In that case, it would be auto-merged with the other `Time` dimension when a virtual cube is created.

Figure 8-4 illustrates the result of the manual merge of `Time to month` and `Time.` Because the hierarchies of the dimensions are identically named, they are merged.

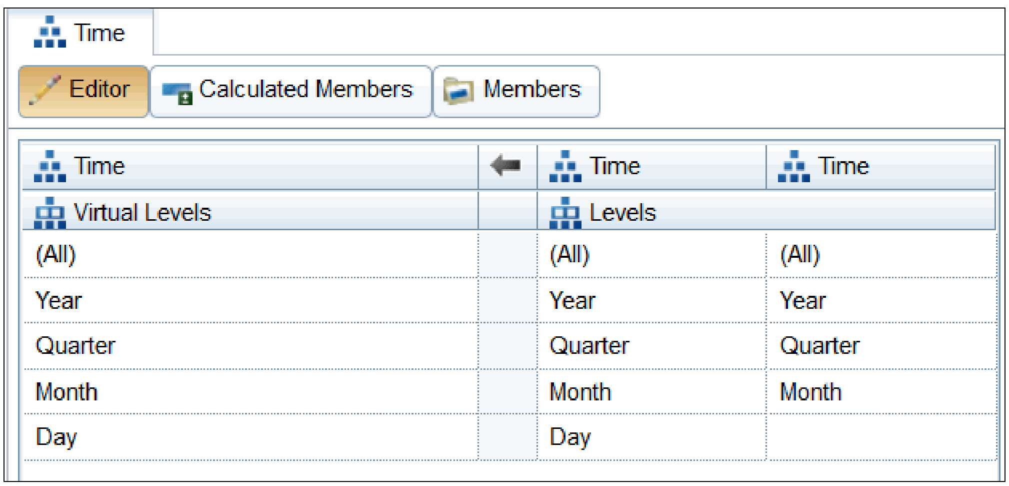


Figure 8-4 Merged non-conformed dimensions.

You can manually merge objects into a virtual object. For information, see 8.2.2, “Manually merging source cube objects” on page 227.

8.2.2 Manually merging source cube objects

It is possible to manually merge objects in a virtual cube that could not be merged automatically. The following sections describe how to manually define these objects.

Defining a virtual dimension

Figure 8-2 on page 225 shows the source cube `gosldw_sales`, which has a dimension named `Time`, and the source cube `gosldw_target`, which has a dimension named `Time to month.` Because these dimensions have different names, they were not merged automatically. The result is that two virtual dimensions, `Time` and `Time to month`, are added to the virtual cube. If both source dimensions contain the same structure and data, you could manually merge them into one virtual dimension named `Time`.

Notice that, unlike `Time`, which has levels for `Year`, `Quarter`, `Month`, and `day`, `Time to month` only has levels for `Year`, `Quarter`, and `Month`. This is because the fact grain for the sales target fact table is month. Although not strictly necessary, to avoid projection of values below the fact grain, model role-playing dimensions for each dimension that participates in a fact table at a grain higher than the dimension grain.

When a query is made against the `day` level, only values from the `gosldw_sales` cube are retrieved. A calculated measure created in a virtual cube that has an expression that references measures from both cubes returns null values if the `day` level was in the query.

For more information about modeling multiple-fact, multiple-grain queries, see 6.5, “Multiple fact and multiple fact grain scenarios” on page 181.

Use the following steps to merge `Time` and `Time to month` dimensions:

1. From the Project Explorer tree, right-click the virtual cube and select **Open Editor**. Figure 8-2 on page 225 shows the Virtual Cube editor.
2. In the Virtual Cube editor, select **Time to month** in the Virtual Dimensions column and click **Delete Virtual Dimension**.

> **Note:** You cannot use a source dimension or hierarchy more than once in a virtual cube. You can use a source measure in multiple calculated measures.

3. Select **Time** and click **Select Source Object** in the cell intersecting the Time and the `gosldw_target` columns.
4. In the Select Source Object window, select **Time to month** and click **OK**.

   `Time to month` now displays in the cell.

Defining a virtual hierarchy

Use the following steps to create a virtual hierarchy:

1. To open the Hierarchy Editor, in the Virtual Dimensions column, double-click the row that has the `Time` dimension.
2. Because both `Time` and `Time to month` dimensions have a hierarchy named `Time` in the source cubes, a virtual hierarchy named `Time` is automatically created, merging both source hierarchies, as shown in Figure 8-5. If these dimensions had hierarchies with different names, two separate hierarchies would be created.

   Similar to the dimensions, you can manually merge two hierarchies that have the same structure and data.

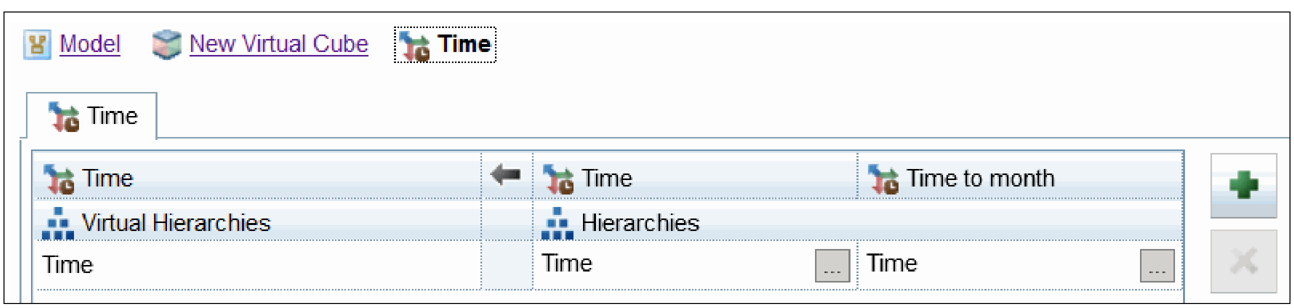


Figure 8-5 Virtual Hierarchy Editor

Viewing virtual levels

When you create a virtual cube, Cognos Cube Designer adds levels from the source cubes to the virtual cube.

If the levels in the source cubes are not identical, then the level names from the first source cube are used as the names of the virtual levels. If one source cube contains more hierarchy levels than the other source cube, the extra levels are added as the lowest levels of the virtual hierarchy. To see the virtual levels in Project Explorer, double-click the hierarchy for which you want to see the levels. Figure 8-6 shows how levels in the Time hierarchy were merged.

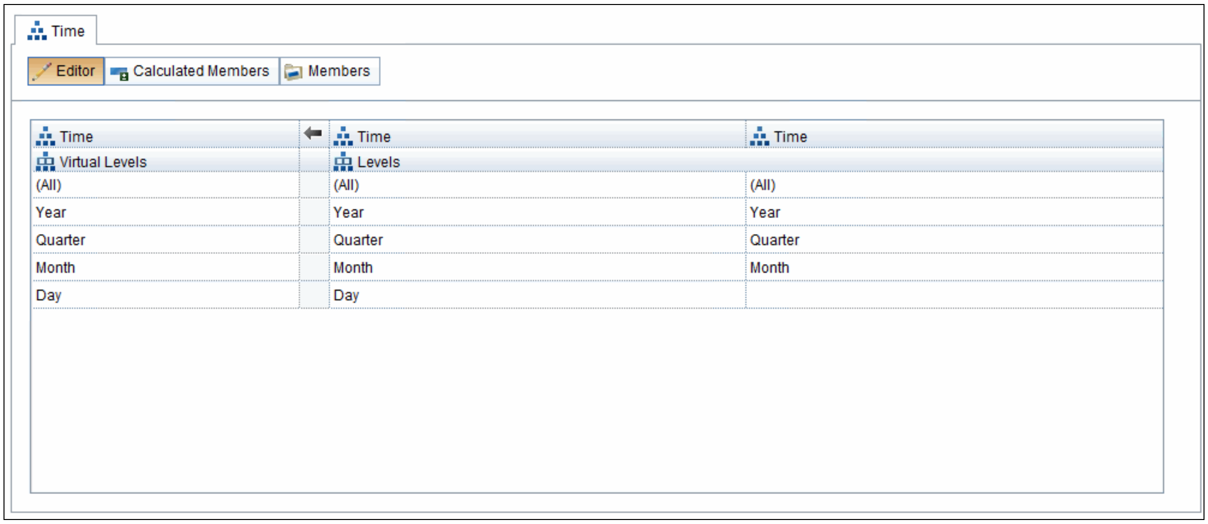


Figure 8-6 Viewing virtual levels

The levels are merged by the order in which they appear in the hierarchy. If the levels in the source dimensions to not match, you will be mixing objects together that you might not want to mix.

For example, assume that in one cube there is a Time dimension that has been modeled with the levels of `Year`, `Month`, and `Week`. Assume that in another cube there is a Time dimension modeled with the levels of `Year`, `quarter`, `month`, and `day`. Assume that the two cubes are merged into a virtual cube. You will end up with levels that mix members from conceptually different levels. In this example, you will have a level with both month and quarter members and a level with both week and month members. Figure 8-7 illustrates this scenario.

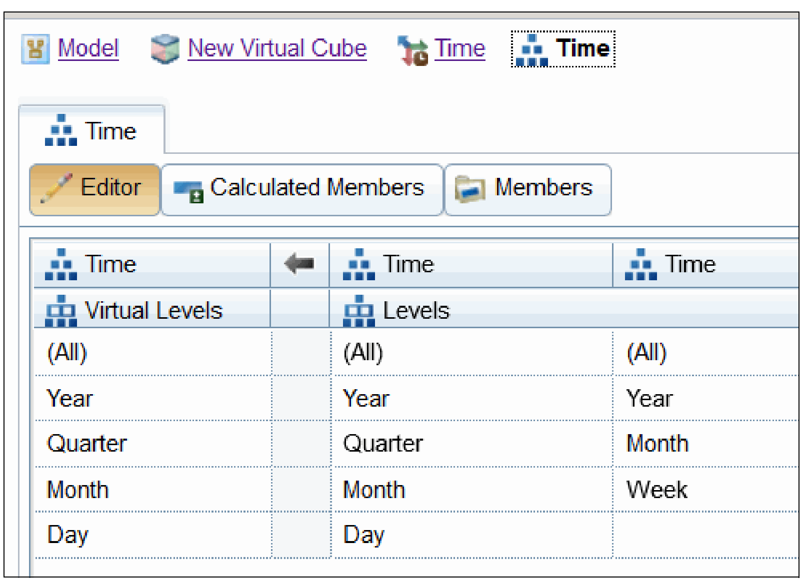


Figure 8-7 Virtual hierarchy with mismatched levels

Defining virtual members

If the keys defining the members in the two source levels are the same, then the two source members are automatically merged into one virtual member. You can use the merged virtual member in queries against both source cubes.

Any members that do not have matching level keys are added to the virtual cube as new virtual members. They can only be used in queries against their source cube.

If a virtual member is not merged correctly, or could not be automatically merged, you can manually merge two source members. When you manually merge a member, it is removed from the list of members that are automatically merged.

For example, assume that you have a member with a business key of `1066` in one cube and a business key of `1415` in another cube. You want to merge them because they represent the same entity.

Members can be merged manually in the Virtual Member editor, which is in the Members tab of the virtual hierarchy editor.

Use the following steps to manually merge two members:

1. Click **Add Virtual Member** to create a new virtual member.
2. Click the editor for the first source cube (box with ellipsis icon).
3. Expand the **Member Explorer** tree.
4. Click the member that you want to merge (as shown in Figure 8-8 on page 231) and click **OK**.
5. Repeat steps 2 through 4 for the second source cube.

At run time, the two members are merged into a virtual member.

Figure 8-8 shows the Virtual Member editor.

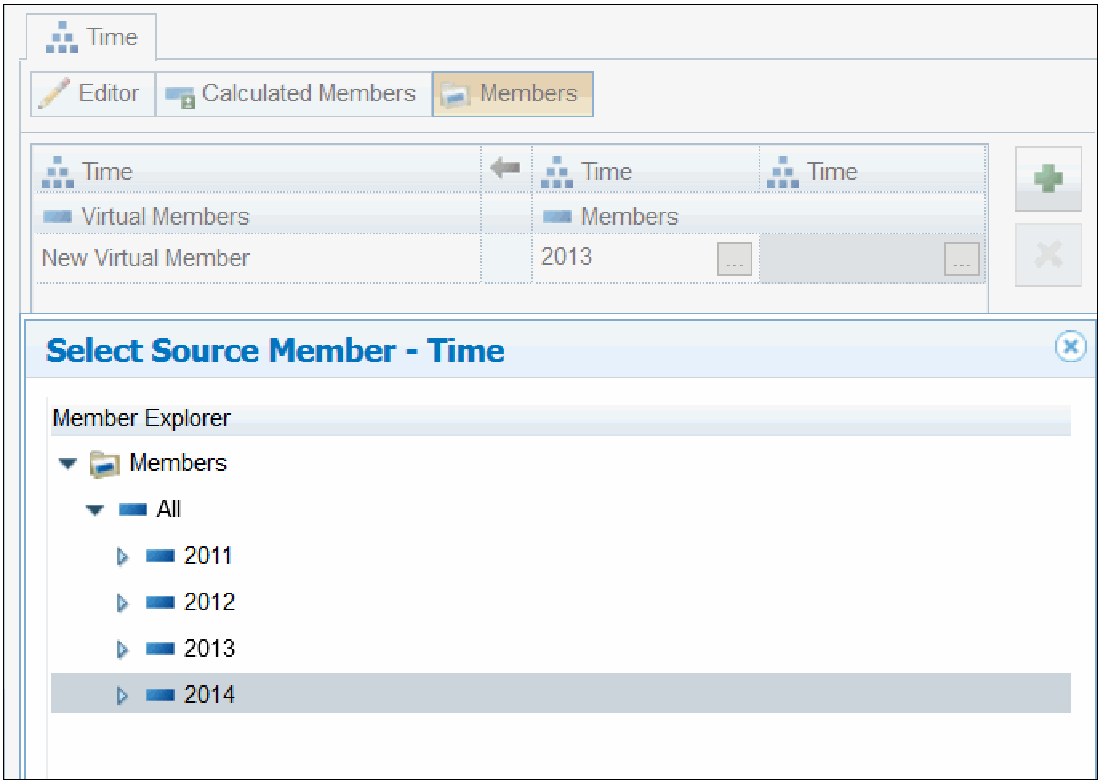


Figure 8-8 Virtual Member editor

Defining virtual measures

Complete the following steps to see the measures included in the virtual cube:

1. Open Cognos Cube Designer and go to the Project Explorer tree.
2. Locate the virtual cube in the project.
3. Double-click the **Measure** folder of the virtual cube or click the **Measure** icon in the cube editor.

When a query is run against a virtual cube, it is routed to the source cubes. The query produces two intermediary results that are aggregated according to the merge operator of the virtual cube.

You can change the virtual cube merge operator in the virtual cube Properties tab. The following merge operators are available:

- Sum
- Subtract
- Multiply
- Divide
- Maximum
- Minimum
- None

Measures that do not have identical names or measures that exist in only one of the source cubes are added to a virtual cube as new virtual measures.

If a virtual measure is not merged correctly, or cannot be automatically merged, you can manually merge two source measures. You can also delete redundant virtual measures.

When merging measures in a virtual cube, it is not possible to map a source measure to more than one virtual measure.

8.2.3 Virtual calculated measures

Virtual calculated measures can be used to build calculated measures that reference measures from both source cubes. An example is a measure that calculates the variance from plan by subtracting the plan values from the actual values. If the remainder is a positive number, then the variance exceeds the plan, which can be beneficial or not, depending on what is being tracked.

Another example is a measure that takes that variance virtual calculated measure and compares it to the plan value to produce a percent of the plan measure.

Building virtual calculated measures into the virtual cube has several advantages:

- Report authors do not need to create these expressions.

  Not having to create the expressions saves the time of the authors and the ad hoc user who might also need the expressions. Also, the report authors do not need to re-create the expression for each report in which they want to use the measure.
- The definition of the measure is consistent and controlled.

  Because the calculation exists in only one place, the risk of an expression being incorrectly created in any report is removed. Enforcing commonality can reduce misunderstanding and misinterpretation of information. The modeler must be sure that the calculation expression is exactly what is required.
- Performance of queries can be increased by having the measure already in the data cache.

  Because the calculation expression is built into the cube and can be primed, the performance of the calculation can be faster than a calculation made in a report.

The expression of the virtual calculated measure does not need to strictly conform with the merge operator. For example, virtual calculated measures calculating variance and percentage variance of sales from the sales plan would use the subtraction and divide operators, but would have the default sum merge operator.

As in regular cubes, you can hide measures in a virtual cube by setting the `visible` property of the measure to `false`.

8.2.4 Deploying virtual cubes

After you finish modeling, you can publish the virtual cube to Cognos Connection in your IBM Cognos BI Server environment so that report authors and others can begin their reporting and analysis. To publish the virtual cube, in Project Explorer right-click the cube that you want to deploy and select **Publish**.

8.2.5 Exercise: Modeling a virtual cube

This exercise creates a virtual cube from the two cubes in the sample model, merges two dimensions with other dimensions, and creates a virtual calculated measure that uses measures from both source cubes as part of its expression.

1. Open IBM Cognos Cube Designer and open the sample model `Great_outdoors_dynamiccube`.
2. Select the **Model** namespace in the Project Explorer tree.
3. Right-click the namespace and select **New** → **Virtual Cube** as shown in Figure 8-9.

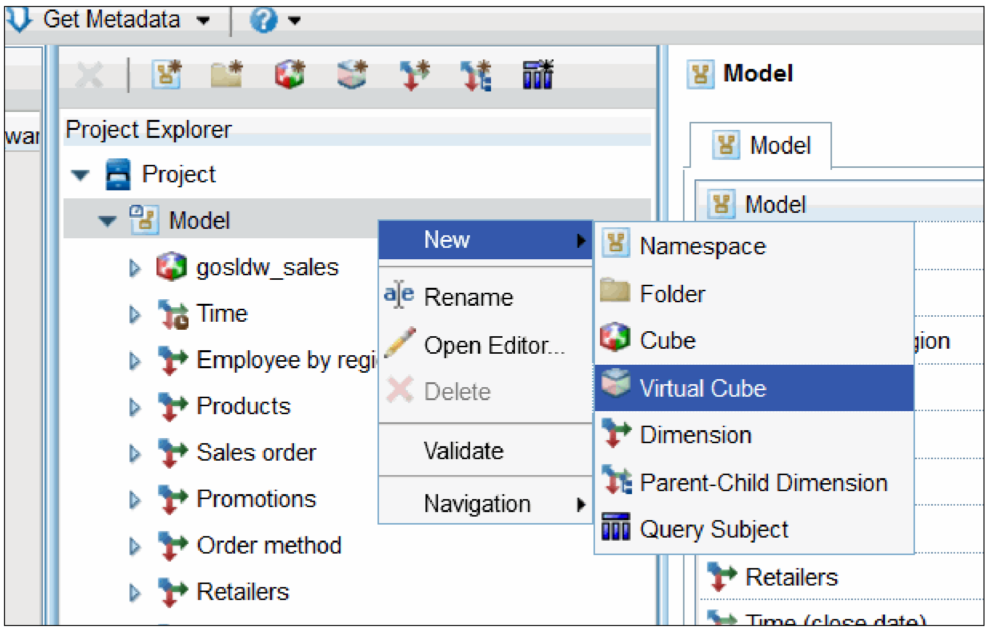


Figure 8-9 Launch New Cube Wizard

The Select Base Cubes window opens (Figure 8-10). This window can also be started by clicking the **New Virtual Cube** icon in the Project Explorer window.

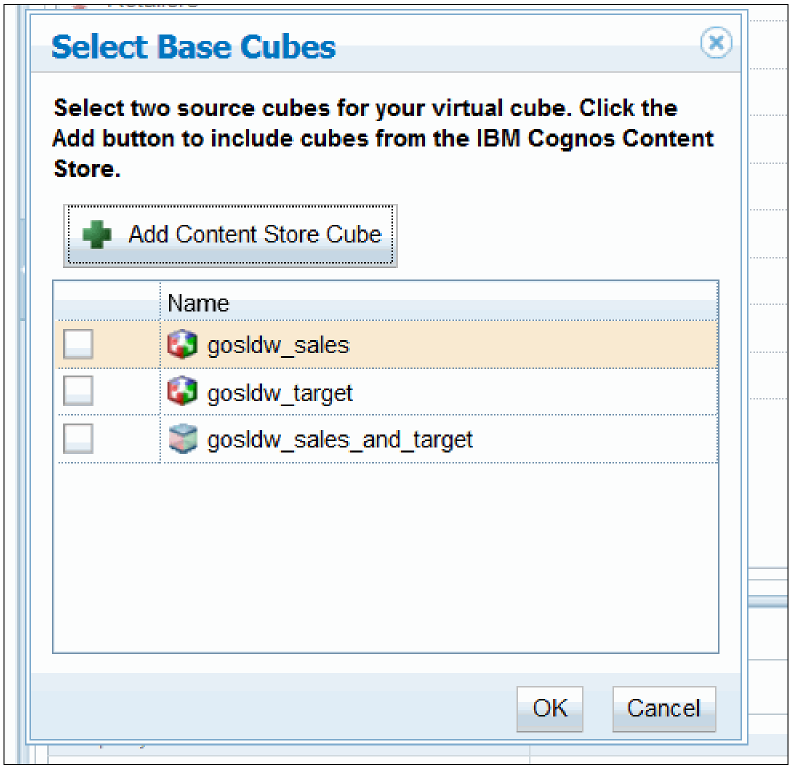


Figure 8-10 Select Base Cubes

4. In the Select Base Cubes dialog, select `gosldw_sales` and `gosldw_target`.

   You can select a maximum of two source cubes to merge into a virtual cube. You can include dynamic cubes from the current project, and dynamic cubes or virtual cubes deployed as data sources to the content store:

   – To include a dynamic cube from the project, select the cube from the list.

   – To include a dynamic cube or virtual cube from the content store, click **Add Content Store Cube**, select the required data source, and then click **OK**.

Note: The order of the cubes in the Select Base Cubes dialog determines the order of the cubes as source cubes in the virtual cube. To control the order of source cubes, reorder the source cubes in the Project Explorer or add one source cube only to the virtual cube when the virtual cube is created and add the second source cube afterward. For more information, see 8.5.1, "Modeling considerations for virtual cubes" on page 249.

5. Click **OK**.

   The new virtual cube is created. It will be at the bottom of the model section of the Project Explorer tree. The default name of any new virtual cube is `New Virtual Cube`. You can rename the virtual cube.

6. To open the definitions of the new cube, double-click **New Virtual Cube**, or from the Project Explorer tree, right-click the virtual cube and select **Open Editor**.

Figure 8-11 shows the Virtual Cube editor.

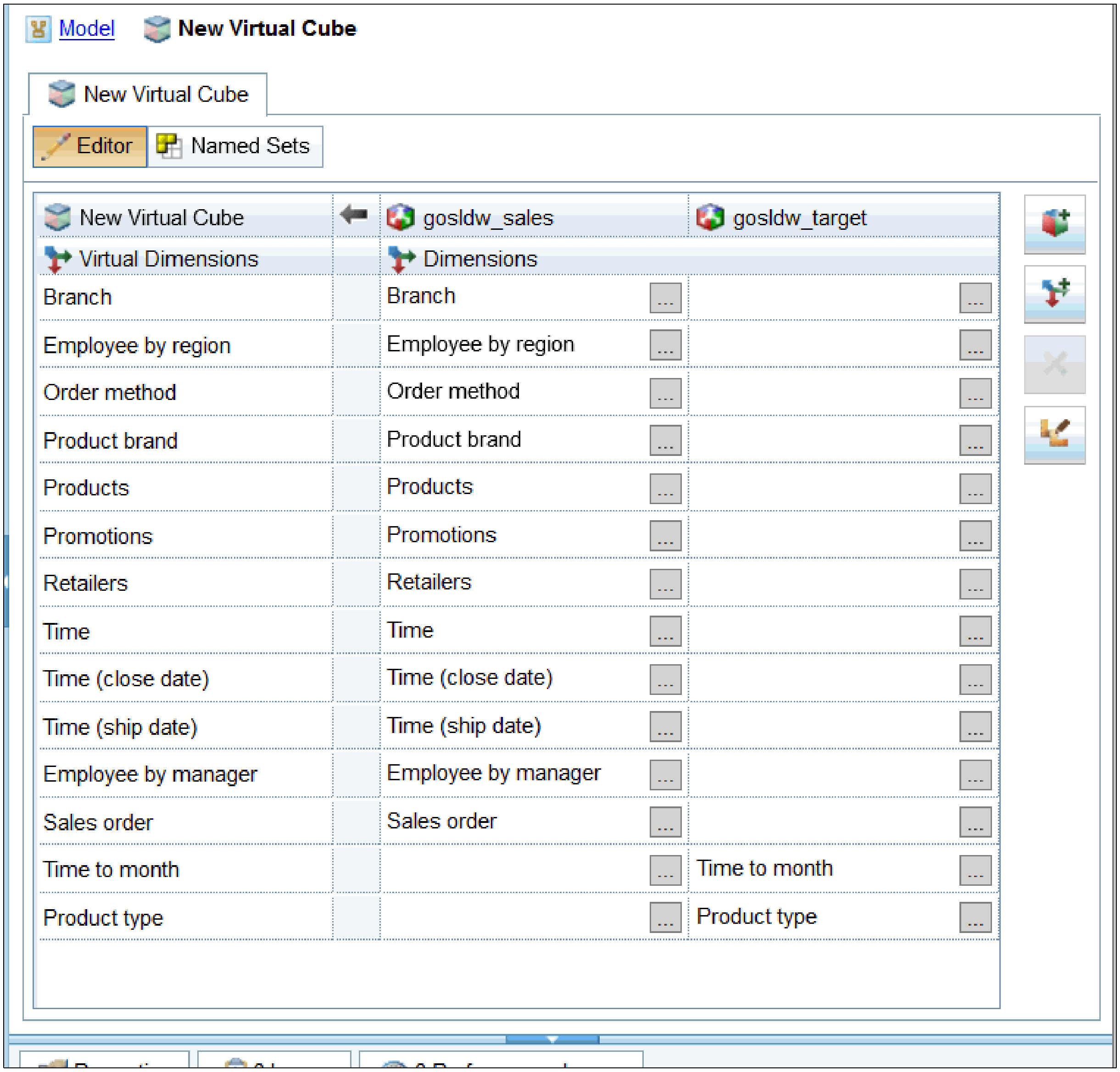


Figure 8-11 Virtual Cube editor

Figure 8-11 shows the three columns in the New Virtual Cube editor: The virtual cube column, the first source cube column (name of the cube), and the second source cube column (name of the cube).

In each column are the names of the dimensions in the virtual cube. As you can see in the figure, there is a dimension for each dimension that exists in the source cubes.

`Time to month` and `Product type` can be merged with `Time` and `Products`, respectively.

7. Select **Time to month** and click **Delete Virtual Dimension**.
8. Select **Product type** and click **Delete Virtual Dimension**.
9. Select **Time** and click **Select Source Object** in the cell intersecting the `Time` and the `gosldw_target` columns.
10. In the Select Source Object window, select **Time to month** and click **OK**.

    `Time to month` now displays in the cell.

11. Select **Product** and click **Select Source Object** in the cell intersecting `Product` and the `gosldw_target` columns.
12. In the Select Source object window, select **Product type** and click **OK**.

    `Product type` will now display in the cell.
13. Click **Measures**.
14. Select the Calculated Measures tab.
15. Click **New Calculated Measures**.
16. Select the new calculated measure and rename it **Revenue variance from plan**.
17. Select **Revenue variance from plan** and open the expression editor from the property pane.
18. Expand the virtual cube in the Project Explorer and expand the measures dimension.
19. Select **Revenue** and drag it into the expression editor.
20. Type the subtraction operator (-) after `Revenue` in the expression editor.
21. Select **Sales target** and drag it into the expression editor.

8.3 Common use cases for virtual cubes

This section describes several scenarios of when virtual cubes can be useful for your application.

8.3.1 Time partitioning

In this scenario, all sales information was originally stored in one cube named `AllSales`. Fact data is added nightly, which requires rebuilding the data cache and refreshing the summary tables. For large cubes, this task requires a significant amount of time. To improve performance, you can partition the data for the `AllSales` cube based on time to form the basis for two cubes: `HistoricSales` and `CurrentQuarterSales`. you can also define a virtual cube, named `VirtualSales`, to join the two cubes. The `HistoricSales` cube is used to record the historical sales information that does not change frequently. The smaller `CurrentQuarterSales` cube is used to record the daily sales information for the current quarter. This technique is referred to as the *delta cube*.

8.3.2 Currency conversion

In this scenario, all sales information is stored in one cube named `RegionalSalesAnalysis`, and there is a business need to convert some of the sales figures that are in US dollars into other currencies. If both the sales and exchange rate information are in the `RegionalSalesAnalysis` cube, then the cube will contain a large amount of redundant data and it will be difficult to maintain. To address this problem, you can create a cube named `ConversionCube` to store the exchange rates so that the `RegionalSalesAnalysis` cube remains unchanged when the exchange rates are updated. Next, you can define a virtual cube named `SalesConversion` to handle the currency conversion for the sales data. The `RegionalSalesAnalysis` and `ConversionCube` cubes will share some common dimensions such as `Time`, but the two cubes will generally have some non-conformed dimensions as well.

The `SalesConversion` virtual cube needs to define a calculated member that handles the actual currency conversion. Because the actual currency conversion might not occur daily but instead only at preset days or at intervals such as monthly or weekly, use the `ClosingPeriod` and `OpeningPeriod` functions in this scenario.

The dimensions of a simple currency conversion base cube are as follows:

- Currency dimension: Typically a flat dimension where each currency is assigned a caption and an identifier, ideally numeric.
- Time dimension: Ideally conformed with the time dimension of the cubes with which the currency conversion cube will be merged.

The single measure in the fact table for this base cube is `Conversion To Local`. The assumption is that any cube with which the currency conversion cube is merged contains values in a single, agreed upon base currency. The `Conversion To Local` measure is thus the rate used to convert base currency values into a value in another currency.

The fact table for this currency conversion-base cube is then a simple three-column table containing currency conversion rates for date and currency combinations. Conversion rates at non-leaf levels of the time dimension can be computed either as averages or medians of the leaf level rates, or they can be assigned values from in-database aggregates that are built explicitly to contain conversion rates used within an organization.

A cube containing the values to be converted is merged with the currency conversion cube to create a new virtual cube. In this scenario, the base cubes are merged along the time dimension. The currency dimension is exposed through the virtual cube. The assumption is that all monetary values in the primary base cube are stored in a single currency that is also the base currency for the conversion rates stored in the currency conversion cube. It is also important that the monetary measures in the primary base cube do not have any formatting applied to them, notably currency formatting.

In the simplest case, where the time dimensions of the two base cubes are at the same level and the currency conversion cube contains rates at all levels of the time dimension, each measure of the base cube that is to be exposed in the virtual cube can be exposed by manually creating a virtual calculated measure in the virtual cube. Each of these virtual measures is a merge of the nominated measure from the primary base cube and the `Conversion To Local` measure from the currency conversion cube. See "Example for non-conformed currency conversion cube" on page 238 for details about how to create this virtual calculated measure.

In cases where rates are not applicable to all dates in the base cube, it might be necessary to create a calculated measure in the virtual cube that selects or computes an applicable rate based on the date within a query. The virtual cube implementation of each base cube measure is then a multiplication of the measure value of the base cube with that of the new calculated measure.

With this scenario, you can view the value of a measure in each supported currency by projecting a measure along one axis of a report, for example, and one or more currency members along the other axis.

Note: In the case where a conversion cube that has roll up values in a currency other that the default currency of the base cube, the detail and summary values in a report might not add up. The summary values will not be the roll up of the converted base values from the nominated level. Instead, the roll-up uses the higher level values converted using the conversion rate at that level, or perhaps an average of lower-level roll-up values.

Example for non-conformed currency conversion cube

A common issue with currency conversion cube is the need to obtain values of a measure from one of the base cubes that is unaffected by the context of non-conformed dimensions from the other base cube. This section shows a simple example to explain how to resolve this issue.

Assume a virtual cube that brings together base cubes `RegionalSalesAnalysis` and `ConversionCube`. The `Time` dimension is common to both base cubes, and so it is merged automatically in the resulting virtual cube. Base cube `RegionalSalesAnalysis` has a non-conformed dimension and hierarchy called `Retailer`. Base cube `ConversionCube` has a non-conformed dimension and hierarchy called `Country`. The virtual cube `SalesConversion` exposes all the dimensions in the two base cubes. Figure 8-12 shows how to merge all these dimensions in the virtual cube.

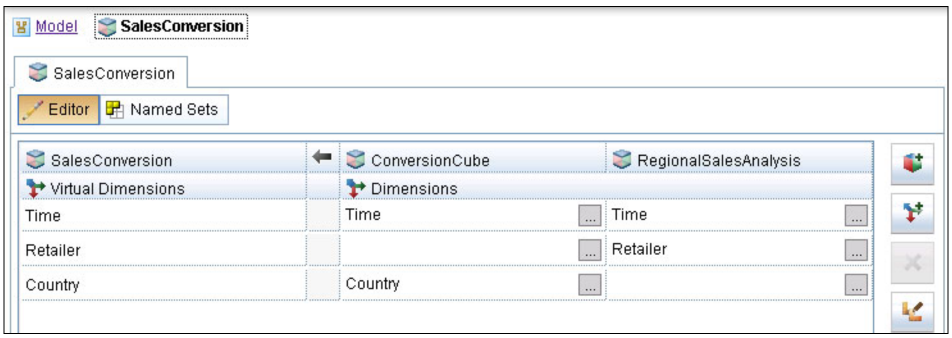


Figure 8-12 Virtual Dimensions

The `Conversion To Local` measure is only available in the ConversionCube, and the `Measure Unit sale price` measure is only available in the `RegionalSalesAnalysis` cube. Figure 8-13 shows how to include both measures in the virtual cube.

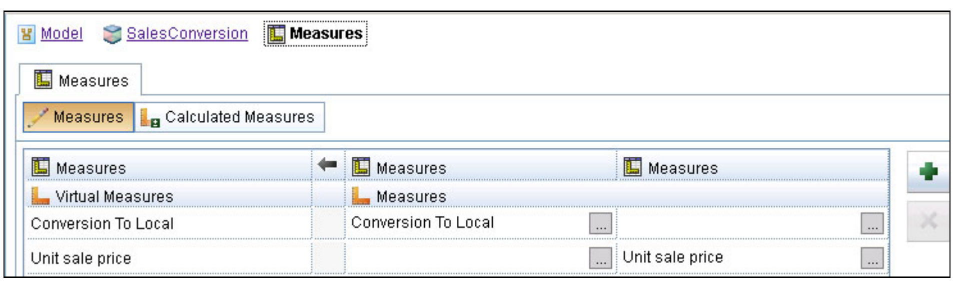


Figure 8-13 Virtual Measures

To get the virtual cube calculated measure to be computed in the context of the target currency and date, you must create a calculated measure. A typical first approach might be something like `[Unit sale price] * [Conversion To Local]`.

Unfortunately, as soon as the member in context of either of the non-conformed dimensions is not the `All` member, the values of the calculated measure suddenly becomes null. That is because there is a non-conformed dimension `Country` that is only in the base cube `ConversionCube`. So the measure `Unit Sale price` from `RegionalSalesAnalysis` cube only has value for the `All` member of the `Country` dimension. The same is true for measure `[Conversion To Local]`, which only has value for the `All` member of the `Retailer` dimension from `RegionalSalesAnalysis`.

To solve this problem, you must create two intermediate calculated measures. The intermediate measures are used in the final calculated measure. You can make these two intermediate calculated measures as hidden measures by setting their `Visible` property to `false` as show in Figure 8-14.

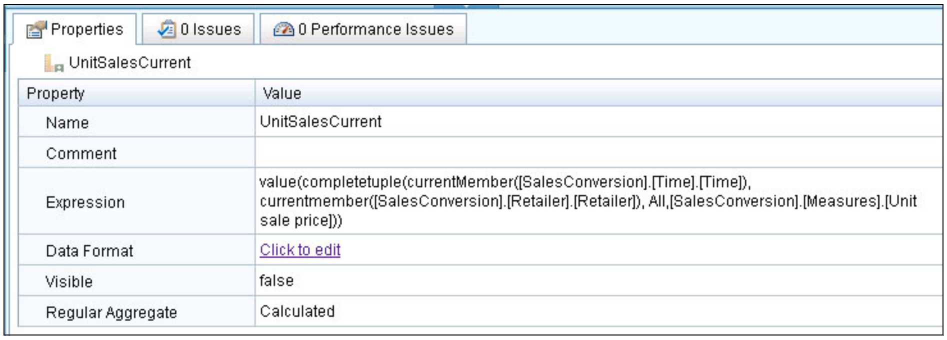


Figure 8-14 Hidden virtual calculated measure with property Visible set to false

`ConversionRate` and `UnitSalesCurrent` are the intermediate calculated measures, and `UnitSalesLocal` is the currency converted measure as shown in Figure 8-15.

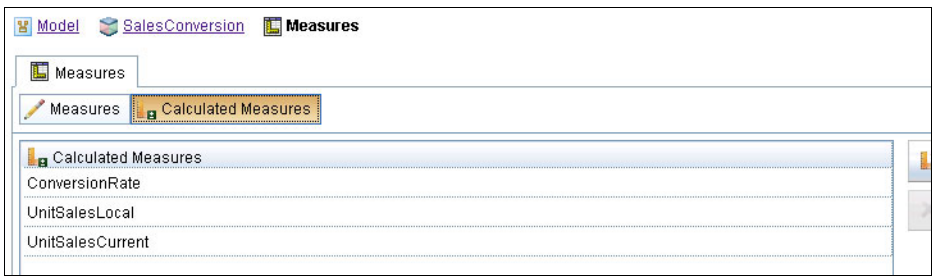


Figure 8-15 Virtual calculated measures

The expression for `ConversionRate` is a value of a tuple consisting of the current member for each of the hierarchies that are in the `ConversionCube` base cube, the ALL member for every non-conformed hierarchy in the dimensions from the `RegionalSalesAnalysis` base cube that occur in the virtual cube, and the `Conversion To Local` measure, which only exists in the `ConversionCube` base cube.

As the actual currency conversion might not occur daily, but instead only at preset days or at intervals such as monthly or weekly, the `ClosingPeriod` and `OpeningPeriod` functions can be used in this scenario. `ConversionRate` in Figure 8-16 is an example of using the `OpeningPeriod` function at the `Month` level. For this cube, the business wants currency conversion at the first Month level in the context of the `Time` current member. The `All` member referenced in the `completetuple` expression is the `All` member for `Retailer` hierarchy in the `Retailer` dimension.

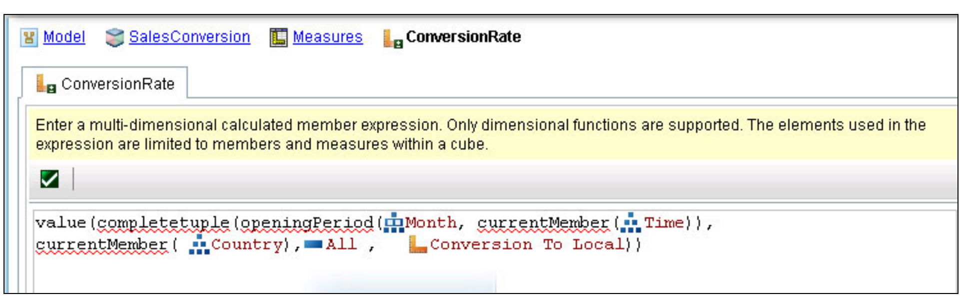


Figure 8-16 Intermediate calculated measure ConversionRate

The expression for `UnitSalesCurrent` is a value for a tuple consisting of the current member for each of the hierarchies that are in the RegionalSalesAnalysis base cube, the ALL member for every non-conformed hierarchy in the dimensions from ConversionCube base cube that occur in the virtual cube, and the `Unit sale price` measure that only exists in the `RegionalSalesAnalysis` base cube. In Figure 8-17, the `All` member referenced in the `completetuple` expression is the `All` member for the `Country` hierarchy in the `Country` dimension.

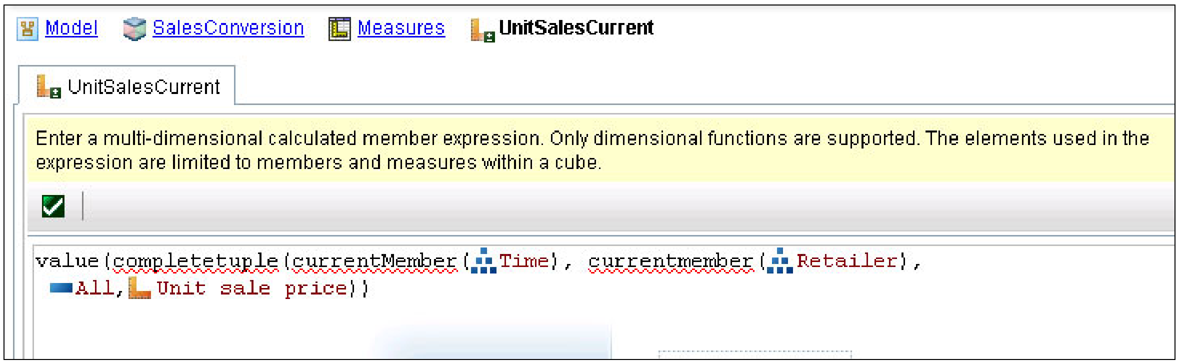


Figure 8-17 Intermediate calculated measure UnitSalesCurrent

The next step is to perform the conversion by multiplying the `ConversionRate` measure, which has been computed in the context of the date, by the `UnitSalesCurrent` measure, as shown in Figure 8-18.

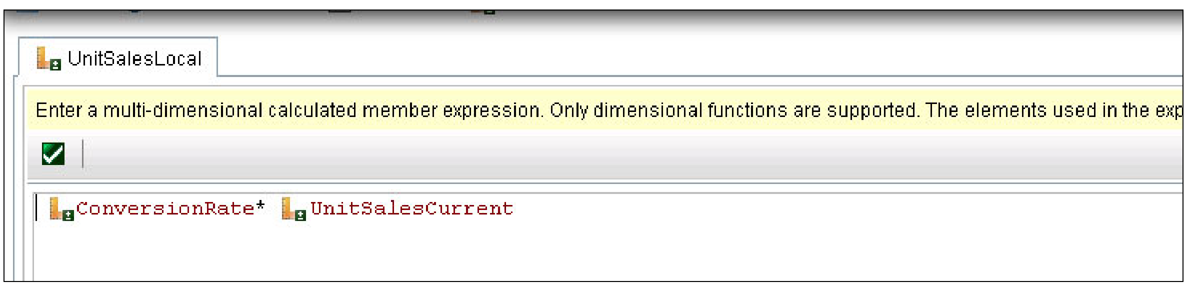


Figure 8-18 Final calculated measure UnitSalesLocal

Note: This example uses the `completetuple` function instead of the tuple function in the calculated measure expressions. The reason is that the default member of the unreferenced dimensions is locked in context when using the `completetuple` function. However, if you reference any member explicitly in the `completetuple` function, this member remains fixed in context.

Unlike the tuple function where the current member of any unreferenced hierarchy is added to the tuple, the `completetuple` function adds the default member of the hierarchy to the expression by default. Therefore, you do not need to explicitly reference `All` members for all non-conformed dimensions if the `All` member is the default member for each hierarchy. You just have to explicitly add currentMember for each hierarchy in the base cube having the measure referenced by this expression.

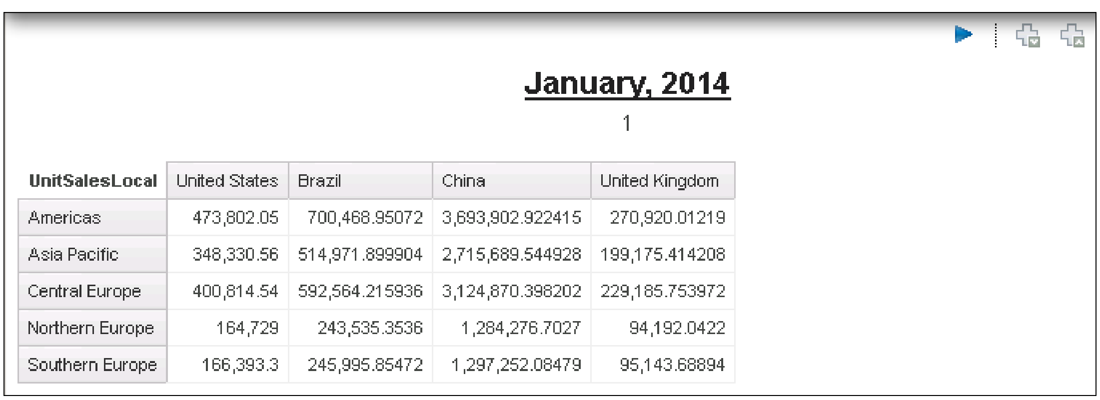
If any of the hierarchies in the non-conformed dimensions have a default member other than the `All` member, then the `All` member must be explicitly added to the `completetuple` function.

Sometimes a non-conformed hierarchy can be a multi-root hierarchy in which case this approach does not work. The virtual cube looks for the default member as the context of the hierarchies in the non-conformed dimensions. In a multi-root hierarchy, however, there is no `All` member to act as the default member, and no other member from the hierarchies of the non-conformed dimensions are in context of the measure, so the measure value will always be null.

The solution, then, is to change all the multi-root hierarchies back into hierarchies with an `All` member and to ensure that the `All` member is added to the calculated measure expressions as required in the virtual cube.

The general rule when creating intermediate calculated measures for currency conversion cube to avoid getting null value is as follows: If a measure is to be allocated constantly relative to a non-conformed dimension in a virtual cube, then the measure must have its context set explicitly for each hierarchy of those dimensions in a `completetuple` function along with the measure itself.

If the virtual cube has a report that contains the non-conformed hierarchy `Retailer` on rows and `UnitSalesLocal` for several different currencies on columns in January 2014, you get not null values as shown in Figure 8-19.


January, 2014

1

UnitSalesLocal	United States	Brazil	China	United Kingdom
Americas	473,802.05	700,468.95072	3,693,902.922415	270,920.01219
Asia Pacific	348,330.56	514,971.899904	2,715,689.544928	199,175.414208
Central Europe	400,814.54	592,564.215936	3,124,870.398202	229,185.753972
Northern Europe	164,729	243,535.3536	1,284,276.7027	94,192.0422
Southern Europe	166,393.3	245,995.85472	1,297,252.08479	95,143.68894

Figure 8-19 Report with Non-conformed Hierarchies

8.3.3 Multiple fact tables

With virtual cubes, you can build a cube that shares dimensions between cubes, and build reports that use facts from multiple cubes. One example is to have a virtual cube that includes a cube with actual values and a cube with target or budget values.

You can build calculated measures into the virtual cube that reference measures from both source cubes. One example is a measure that calculates the variance from plan by subtracting the budget or the target plan values from the actual values. If the remainder is a positive number, then the variance exceeded the budget or target plan. Another example is a measure that takes that variance and compares it to the budget of the target plan value to produce a percent of plan measure.

8.4 Managing virtual cubes

Chapter 7, “Administering dynamic cubes” on page 199 describes tasks that are involved in the administration of dynamic cubes. The administration of virtual cubes follows the same procedures described for dynamic cubes with some additional considerations specific to virtual cubes. This section describes the additional administration considerations required for virtual cubes.

The action of starting and stopping a virtual cube is much the same as that described in Chapter 7, “Administering dynamic cubes” on page 199. However, the dependency of a virtual cube on its source cubes introduces several important considerations. Source cubes can be base cubes, virtual cubes, or a combination of the two. The term *base cube* refers to the standard type of dynamic cube, that is, a cube that is not a virtual cube.

Managing virtual cubes includes these considerations:

- The virtual cube and both source cubes must all be configured on the same dispatcher.
- The source cubes of a virtual cube must be started and available before the virtual cube can be started.
- Dynamic cubes (either base or virtual) that are being used as the source of a virtual cube cannot be stopped or paused until after the dependent virtual cube is stopped.
- Refreshing the data and member cache of a source cube also refreshes the data and member cache of any associated virtual cubes. The refresh of a cache in one of the source cubes causes a subsequent refresh of the same caches in the dependent virtual cube.

  > **Note:** These caches cannot be refreshed at the virtual cube level.

- Not all actions that can be performed on a base cube are applicable to a virtual cube. Some of these actions, such as refreshing the data cache, must be performed at the base cube level. Other actions relate to features not available in a virtual cube, such as workload logging for in-memory aggregates because virtual cubes do not have in-memory aggregates.
- It is possible to create multiple tiers of virtual cubes, in which the source cubes include one or more virtual cubes. In this scenario, actions that can only be performed on base cubes must be traced down to the lower level source cubes that are not virtual cubes (that is, base cubes) and performed there.

Locating a virtual cube instance

As with base cubes, there might be more than one instance of a virtual cube configured in your environment. Instances of a virtual cube can be located within IBM Cognos Administration in two ways:

- Open the Cognos Dynamic Cubes scorecard and navigate through the list of configured cubes.
- Open the System scorecard and navigate through the list of dispatchers in the environment.

Starting on the Cognos Dynamic Cubes scorecard

The Cognos Dynamic Cubes scorecard presents a list of all the dynamic cubes configured in the system. Virtual cubes have a different icon to differentiate them from base cubes. From here the administrator can select a specific cube to identify all server groups and dispatchers that the cube is configured against, hence confirm the locations of the various instances of the cube.

From this scorecard it is possible to perform a series of actions against all instances of a virtual cube or just a single instance.

Perform the following steps to locate a virtual cube:

1. Launch IBM Cognos Administration.
2. Select the **Status** tab.
3. From the list on the left, click **Dynamic Cubes** to display the Dynamic Cubes scorecard. Virtual cubes can be identified by their unique icon.
4. Actions performed from the menu that is displayed when you click the arrow to the right of a cube in this view are performed on all instances of the cube.
5. To locate a specific instance of a virtual cube, click the virtual cube in the scorecard. This action displays the list of server groups with dispatchers hosting instances of this cube.
6. Click a server group to display the dispatchers against which the cube is configured. The listed dispatches are the locations of the individual configured instances of the cube. Your current location in the hierarchy of objects is displayed in a bread-crumb trail at the top of the scorecard.

Changing Cognos Dynamic Cubes scorecard view

The Cognos Dynamic Cubes scorecard defaults to a view that shows a list of all published dynamic cubes. This view can be filtered to show base cubes only, virtual cubes only, or server groups and then the dispatchers against which the cubes have been configured.

Perform the following steps to change the Cognos Dynamic Cubes scorecard view:

1. In the Scorecard pane, click the arrow to the right of **Data Stores - (All)** as shown in Figure 8-20. A menu is displayed.

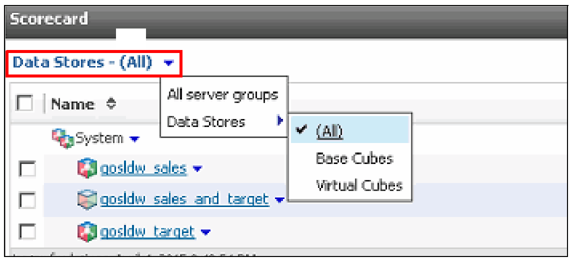


Figure 8-20 Changing the Dynamic Cubes scorecard view

2. ‘From the menu that is displayed when you click the arrow, select **All server Groups** or **Data Stores** followed by **Base Cubes** or **Virtual Cubes** as shown in Figure 8-20.

Starting on the System scorecard

The System scorecard presents a list of all the servers in the system. From here, the administrator can locate dynamic cubes instances by the query service to which they are configured. It is possible to navigate through a server to a dispatcher and then the associated query service. Alternatively, the scorecard view can be changed to show all server groups, all dispatchers, or all instances of the query service in the system.

Perform the following steps to locate a virtual cube:

1. Start IBM Cognos Administration.
2. Select the Status tab.
3. From the list on the left, click **System** to display the System scorecard.
4. Click a server to display its dispatchers.
5. Click a dispatcher to display its services, then click **Query Service** to display the cubes hosted by this query service.

Perform the following steps to change the System scorecard view:

1. In the System scorecard pane, access the Change view options (use the process described in step 1 on page 244). A menu is displayed.
2. From the menu, select **All server groups**, **All dispatchers**, or **Services** followed by **Query** (see Figure 8-21).

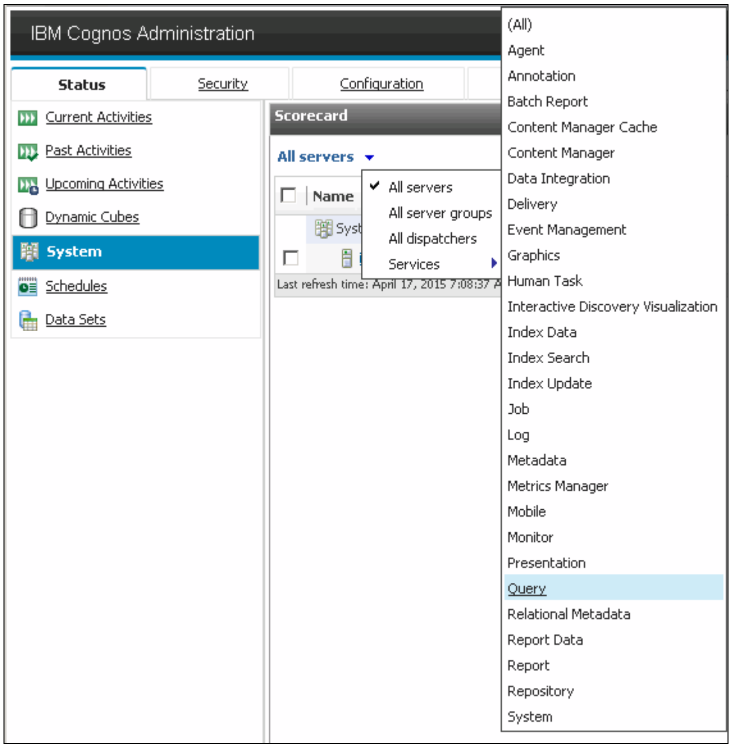


Figure 8-21   Changing the System scorecard view

The method described in “Starting on the Cognos Dynamic Cubes scorecard” on page 243 is used for the rest of this section.

8.4.1 Starting a virtual cube

This section follows the procedure described in “Starting on the Cognos Dynamic Cubes scorecard” on page 243.

Perform the following steps to start a virtual cube:

1. Start IBM Cognos Administration.
2. Select the Status tab.
3. Select **Dynamic Cubes**.
4. Locate the virtual cube that you want to start. To start all instances of this cube, proceed step 5. To start a specific instance of this cube, proceed to steps 6 on page 246 and 7 on page 246.

5. Starting all cube instances: Click the arrow to the right of the virtual cube and select **Start** from the menu that is displayed. A message indicating that your request was successfully submitted is displayed as shown in Figure 8-22.

Figure 8-22 Virtual cube start request successfully submitted

6. Starting a specific cube instance: Click the virtual cube, the server group, and then click the query service that is hosting the cube instance.
7. Click the arrow to the right of the query service and select **Start** from the menu that is displayed. A message indicating that your request was successfully submitted is displayed.

When you start a virtual cube, the system checks that the source cubes are both available. If one or both source cubes are paused or not started, a window opens. The window displays a warning message to inform you that to start a virtual cube, the system must also start the source cubes. Click **OK** to continue as shown in Figure 8-23.

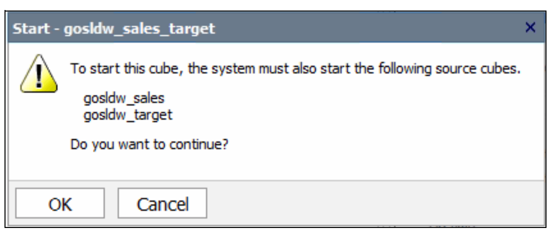


Figure 8-23 Starting the virtual cube

If you click **Cancel**, the request to start the virtual cube is canceled.

If you click **OK**, the source cubes that are listed in the window will be started, then when the source cubes are running the virtual cube will be started. Cognos Dynamic Cubes propagates this check through the various levels of nested cubes to ensure that all source cubes are in a running state before attempting to start the selected virtual cube.

Note: The check to confirm that the source cubes are running is only performed when starting a virtual cube from Cognos Administration. This feature is not supported for virtual cube start requests through the SDK, the Dutchman command-line tool, or Cognos Cube Designer.

Troubleshooting a virtual cube start

If the virtual cube does not start, locate the correct instance of the source cubes and verify that the status of each instance is *available*. It is important to remember that the virtual cube and its source cubes must all be on the same query service.

If a source cube did not start, then click the arrowto the right of the cube name and select **View recent messages** to determine why the cube did not start. Be sure to check both source cubes.

8.4.2 Stopping or pausing a virtual cube

You can stop the virtual cube from the same locations in Cognos Administration and with a similar process as that used to start the cube. Locate the instance of the virtual cube that you want to stop, and select **Stop after active tasks complete** from its menu. The virtual cube status will change to *unavailable*.

To pause the cube, use this same method but this time select **Pause** from the menu. The virtual cube status will change to *partially available*.

For more information about locating and starting a virtual cube, see 8.4.1, “Starting a virtual cube” on page 245.

8.4.3 Stopping the source cubes

Before stopping a dynamic cube that is being used as a source cube, you must stop all the associated virtual cubes. If you attempt to stop a source cube while an associated virtual cube is running, a window opens displaying a warning message and the list of dependent virtual cubes that must first be stopped as shown in Figure 8-24.

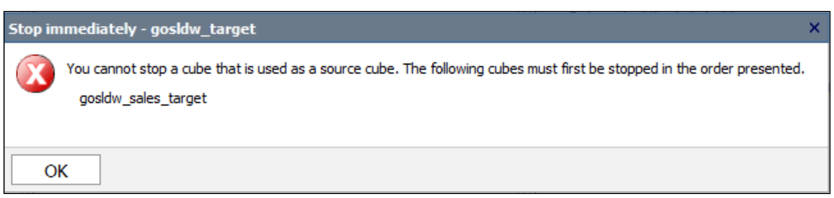


Figure 8-24 Stopping the source cube

If you click **OK**, the request to stop the selected source cube is canceled.

To stop the selected source cube, first locate the dependent virtual cubes and stop them in order that is listed in the warning message. After these cubes are all stopped, locate the source cube again and stop it.

8.4.4 Verifying virtual cube dependency

Each virtual cube depends on two other cubes, which are its source cubes. This dependency is defined in the cube model using Cube Designer, but is visible in the metrics of the various configured instances of the published virtual cube.

To verify which two source cubes are associated with a virtual cube, locate a specific instance of the virtual cube and refer to the Source Cubes metric. This metric lists the source cubes for the virtual cube you have located. The following example uses the sample cubes.

Perform the following steps to verify the list of source cubes that are associated to a virtual cube:

1. Start IBM Cognos Administration.
2. Select the Status tab.
3. From the list on the left, select **Dynamic Cubes**.
4. Click `gosldw_sales_target` from the list of cubes to display the server groups with dispatchers hosting this virtual cube.
5. Click `DefaultServerGroup` to display the dispatchers with configured instances of the selected virtual cube.
6. Select a query service from the list to display the metrics for the selected virtual cube instance.
7. In the metrics section, hover over the value of the `Source cubes` metric to see the list of source cubes that are associated to the selected virtual cube, as shown in Figure 8-25.

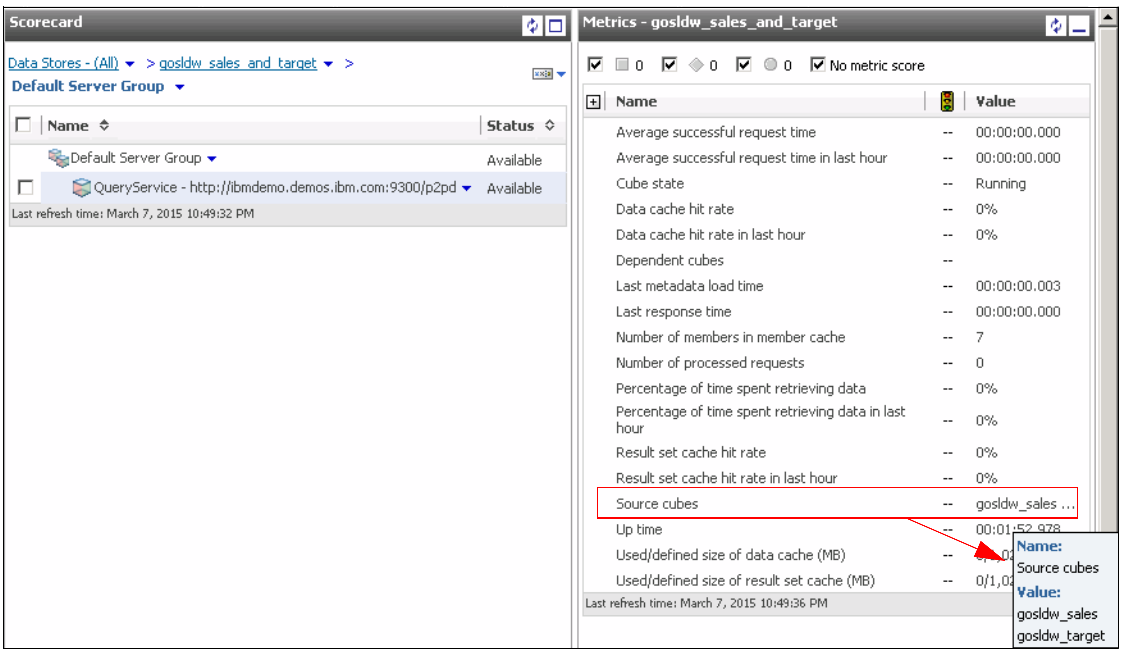


Figure 8-25 Source cube associated with virtual cube

The two source cubes can be base cubes, other virtual cubes, or a combination of the two. To verify whether a cube is being used as a source cube, locate a specific instance of the cube and hover over the value of its Dependent cubes metric. This metric shows the list of virtual cubes that are dependent on the cube you located, as shown in Figure 8-26.

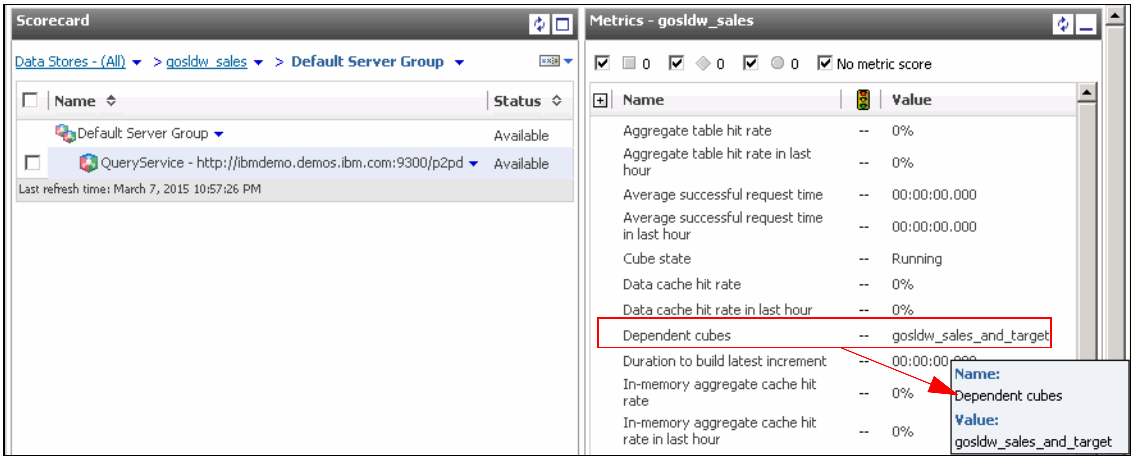


Figure 8-26 Dependent cubes

8.5 Virtual cube considerations

This section describes modeling, caching, and performance considerations for virtual cubes.

8.5.1 Modeling considerations for virtual cubes

Modeling considerations for virtual cubes include calculated measures and members, aggregate cubes, and support for multiple locales.

Source cube order

The order of the virtual cube's source cubes governs some behavior.

The virtual cube editor has three columns: The virtual cube list, the first source cube list, and the second source cube list.

The order in which the source cubes appear in the Select Base Cubes window determines which cube is the first source cube (left source cube column of the virtual cube editor) and which cube is the second source cube.

The Select Base Cubes are placed in this order:

1. Cubes and virtual cubes in the model in the order in which they are arranged in the Project Explorer.
2. Cubes from content store in their order of selection.

Source cubes selected from the content store are added to the bottom of the list of possible source cubes in the Select Base Cubes window.

In Figure 8-27, `gosldw_target` and `gosldw_sales` are cubes in the content store. `gosldw_target` is selected first.

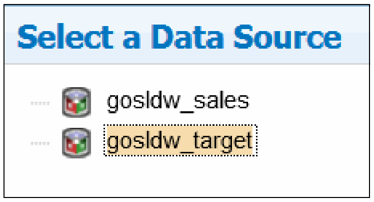


Figure 8-27 Two cubes in the content store.

In Figure 8-28, `gosldw_target` and `gosldw_sales` are cubes in the content store. They are selected in that order and added to the Select Base Cubes window. These cubes appear at the bottom of the list of source cubes in the window.

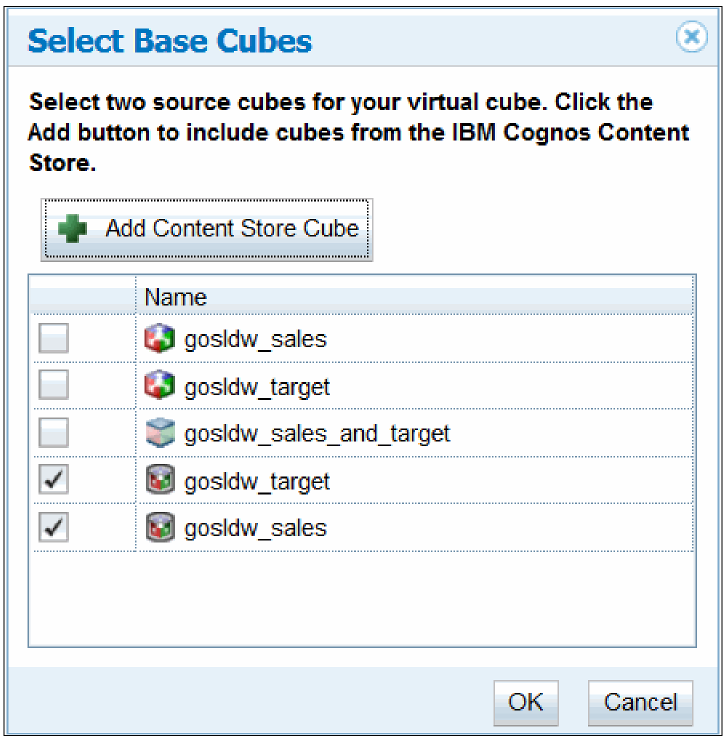


Figure 8-28 Selection order effect on the order of source cubes of a virtual cube

The order of source cubes that exist in the model Project Explorer tree determines their position in the Select Base Cubes window. The order of model cubes reads from the top to the bottom of Project Explorer. Any cube that appears in a namespace or folder is appended to the list of base cubes by top-down appearance.

This is important because the order of the source cubes in the virtual cube editor in turn governs behavior such as treatment of relative time members and merge order.

The order of the source cubes determines the naming of common virtual objects. Many virtual objects take their name from the name of the analogue source object. These virtual objects derive their names from the analogue source object of the first source cube. You might want to edit the virtual object to give it a more appropriate name.

The order of dimensions in the source cubes determines the order in which the virtual dimensions appear in the virtual cube. You can reorder the virtual dimensions by moving them in the Project Explorer.

If it is not possible to control the order of the source cubes in the Select Base Cubes window during the virtual cube creation process, add only the cube that you want to be the first source cube. After the virtual cube is created, add the second source cube from the virtual cube editor.

In addition, you can delete a source cube from the virtual cube by selecting it in the virtual cube editor and clicking **Delete**. The remaining source cube now becomes the first source cube. Then add a new source cube to the virtual cube, which becomes the second source cube. Virtual levels that no longer have any source levels remain in hierarchies. If this is the case, you might need to recreate the virtual dimension because it is not possible to delete virtual levels.

Source cube refresh

Modifications to the structure of a source cube, such as adding additional dimensions, measures, hierarchies, or attributes, are not propagated to the virtual cube. If you want to have the modifications added to the virtual cube, you must delete the source cube from the virtual cube, add it back again, and, if necessary, perform the required manual editing and merge specification.

Relative time members

You cannot define relative time members in the virtual cube. The virtual cube uses the relative time members that exist in the first source cube of the virtual cube.

Calculated measures and calculated members

Calculated measures and calculated members from source cubes are not added to a virtual cube. To use calculated measures or members from source cubes, you must manually define them in the virtual cube.

Named sets

Named sets from source cubes are not added to a virtual cube. To use named sets in a virtual cube, you must manually define them in the virtual cube.

Measures with non-distributive regular aggregates

Measures with non-distributive regular aggregates that are in a source cube should not be used in a merged measure. Because none of the merge operators allow for correct merge of count distinct, average, median, or standard deviation values, these values cannot be computed correctly across the merged measure.

A measure with a non-distributive regular aggregate from a source cube that is not merged with another measure will be computed correctly because a query that includes the measure returns values from the source cube only.

Measures with aggregate rules

Merged measures that have aggregation rules cannot compute the correct values across the merged cubes.

Aggregate cubes

Aggregate cubes are unavailable in a virtual cube because a virtual cube can retrieve data only from source cubes. If the source cube has aggregate cubes, they will be used through virtual cube queries.

Support for multiple locales

If source cubes include support for multiple locales, a virtual cube also has multiple locale support. A virtual cube automatically supports all locales defined in the source cubes. For example, in source cube 1, English and Portuguese are defined as supported locales. In source cube 2, English and Spanish are defined as supported locales. In the virtual cube, English, Portuguese, and Spanish are all included as supported locales.

A virtual cube also supports the use of multilingual names and captions for a virtual cube, virtual dimensions, virtual hierarchies, virtual levels, and virtual measures.

> **Note:** The All member caption, multilingual names, and captions from source cubes are not automatically added to a virtual cube. You must manually define them in the virtual cube.

8.5.2 Caching and performance considerations

This section describes caching and performance considerations for virtual cubes.

Refreshing virtual cube caches

Cache updates in virtual cubes are different from base cube cache updates. You cannot directly refresh data, a member, or a security cache of a virtual cube. However, when a cache is updated on a source cube, the update is propagated up the stack of its dependent cubes. For more information about how to list dependent cubes for a source cube, see 8.4.1, “Starting a virtual cube” on page 245.

The update is designed to work as an atomic transaction, in which all caches must update successfully, or the entire update fails and caches are rolled back to their previous versions.

In the delta cube scenario, you refresh the data cache of the smaller delta cube and the data cache update propagates to the virtual cube.

The member cache refresh for a virtual cube is almost as straightforward. Consider that if you have an explicit member merge defined, it is possible that one or more of the source members that were used as a source member reference for an explicitly defined virtual member might no longer exist after a member refresh.

For example, suppose that a row for that member was removed from the dimension table. If a source member cannot be found, it will be ignored. This results in a virtual member with only one source member reference. Default member merging does not have this problem because source members are not explicitly defined. However, you might also end up with a virtual member that has only one source member with default merging if the other source cube does not have the corresponding source member.

> **Note:** As with ordinary cubes, the definition of a virtual cube is updated only when the cube is restarted. Refreshing the member cache updates only the set of members for the cube, but does not update the definition of the cube. For example, if you add a newly explicitly defined virtual member, it will not be picked up after a member refresh. You must restart the virtual cube first.

Adjusting data cache size for virtual cubes

If a cube is used only as a virtual cube and it is not accessible from any reporting packages, it might not be necessary to assign a data cache to the cube because data will be cached in the virtual cube that is being accessed. Only caching data for the cubes that are accessible by users is important. Note that any cube, base or virtual, that is not accessible through a reporting package might have little need for a data cache.

If a virtual cube is built to combine history and recent data into a single cube, you might want to assign a data cache to the base cubes, because doing so can ensure fast query performance when the recent data cube is updated.

9

Dimensional security

This chapter describes the concepts of IBM Cognos Dynamic Cubes dimensional security, its hierarchies, security models, members, and filters.

This chapter contains the following sections:

- Overview of dimensional security
- Security model
- Grant versus deny approaches to security
- Visible ancestors
- Secured padding members
- Default members
- Calculated members
- Merging security views or security filters
- Security lookup tables
- Defining security in Cognos Cube Designer
- Publishing and starting the cube
- Applying security views
- Verifying the security model
- Transformer-style security

9.1 Overview of dimensional security

Cognos Dynamic Cubes dimensional security defines what users are allowed to access. It restricts the display and use of members in hierarchies, and the display and use of dimensions, measures, and attributes.

A report that refers to members that are denied to a user does not display those members. A report that refers to dimensions, measures, or attributes that are denied to a user does not run. Report authors are not able to see members and objects that are denied to them.

The hierarchy members that a user can access are defined by dynamic query expressions and security lookup tables.

Restricted members are either members that are explicitly denied or, when there are members explicitly granted, those members that are *not* explicitly granted.

Level-based ragged and unbalanced hierarchies in dynamic cubes are converted into balanced, non-ragged hierarchies by using padding members. To remain consistent with the manner in which hierarchies are presented, the application of dimensional security to a level-based hierarchy retains the non-ragged, balanced nature of the hierarchy by using secured padding members. Secured padding members are inserted in secured hierarchies where the restriction of members would otherwise have created an unbalanced hierarchy.

When applying member security to parent-child (*recursive*) hierarchies, secured padding members are not needed.

In addition to secured padding members, Cognos Dynamic Cubes also supports visible ancestors, where a member is visible, but any tuple value it is a part of will have a secured value.

Cognos Dynamic Cubes security does not support *visual totals*, which means that the summary values are not adjusted when some of the lower-level members are hidden (except as described in 9.14.5, “Cognos Transformer Exclude option” on page 317). For example, if measure values for cities in a country are returned along with the total for the country, the value of that total does not change if some of the cities are hidden due to security restrictions.

Dimensional security does not change the structure of the cube:

- Levels are not removed.
- Members remain on their original levels.
- Members keep their ancestry trees.

> **Note:** Security restrictions only apply to regular users of Cognos Dynamic Cubes. If a user is a member of System Administrators group, then no security restrictions are applied to that user.

9.2 Security model

The security model consists of security filters and security views.

Security filters are defined in dimensions. Security filters govern the display of members in a hierarchy.

Security views are defined in a cube. Security views define what you want to allow a set of users to see and work with. Security views encompass security filters; measure, dimension, and attribute security; and object security settings. You will have a security view for each set of users for which you want to define a common view of the cube.

You can have as many security filters in a hierarchy in a security view as you want.

The security model is defined in IBM Cognos Cube Designer.

9.2.1 Security filters

Zero or more security filters can be defined for each hierarchy within a cube. Each security filter has a name that is unique within the scope of the hierarchy in which it is defined.

Each hierarchy has a built-in security filter named *Grant All Members* that allows access to all members in the hierarchy.

Except for the built-in security filter, all security filters have a member set definition and scope. The member set can be defined by any dynamic query expression that returns a member or a set of members in the hierarchy, or by member key information in a security lookup table.

Five scope options exist: Four Grant options and one Deny option.

Grant options

Grant options have the following characteristics:

- The member set defines the members that users, if granted access to this filter, are allowed access.
- Ancestors of granted members, if not explicitly granted, are treated as visible ancestors. All other members are implicitly denied.
- Scope options for Grant:
  - Grant members: Only the members that are specified in the member set are allowed.
  - Grant members and descendants: The members that are specified in the member set and all their descendants are allowed.
  - Grant members and ancestors: The members that are specified in the member set and all their ancestors are allowed.
  - Grant members, descendants, and ancestors: The members that are specified in the member set, all their ancestors, and all their descendants are allowed.

Deny option

Deny option have the following characteristics:

- The member set defines the members that users, if granted access to this filter, are denied access. All other members are implicitly allowed.
- To ensure that hierarchies do not become ragged, denying a member also denies all its descendants.
- Scope option for Deny is to deny members and descendants.

Notes:

- Most dynamic query set expressions do not include calculated members. However, the descendants and ascendants scope flags include calculated members.
- If errors exist in your dynamic query expression, the hierarchy has all members denied and you are unable to query or explore the metadata of the cube.
- Measures are not secured through security filters. Instead, they are secured on the security views.

9.2.2 Security views

Security views combine any number of member security filters and security definitions for measures, attributes, and dimensions.

Security views are defined in a cube. They are published with the cube. You assign users, groups, and roles to security views in the Cognos Connection cube data source.

If a security view contains more than one filter for a hierarchy, the set of members that a user is allowed access is the difference of the union of the sets of allowed members and the union of the sets of denied members.

The union of security views in which users belong defines what they can see. No member or object is accessible if it is denied to a user in one security view, but allowed in another.

An object inherits the object security state of its parent. You will not be able to see an attribute if it belongs to a dimension that you have been denied.

Consider the following guidelines for views:

- If a user is granted access to multiple views, the consolidated view of the cube is the union of the allowed sets and the union of denied sets to which each view provides access.
- After a member is explicitly denied through a security filter, accessing it is not possible, even if the member is explicitly granted access in another filter.
- If security views are defined on the cube, any users who are not assigned to a security view have no access to the cube.
- If there are no security filters for a hierarchy, all members are visible in the hierarchy. However, absence of security filters on a hierarchy differs from the built-in *Grant All Members* security filter because explicit grant of access takes precedence over implicit grant.
- If there are security filters, but none of the filters have a grant scope, all members that are not explicitly denied are granted.

9.3 Grant versus deny approaches to security

In Cognos Dynamic Cubes dimensional security, deny has precedence over grant. After a member is explicitly denied, it cannot be accessed. Using a combination of deny filters further restricts an access of the user to the members in a hierarchy. Conversely, using a combination of grant filters builds out a user's access to the members in a hierarchy.

Using a grant scope without the ascendants option can lead to visible ancestors.

The possibility of hitting the tuple security, described in 9.8.1, “Secured tuples” on page 273, is found only with grant security when more than one hierarchy is secured. Using deny security avoids that scenario.

9.4 Visible ancestors

If ancestors of a granted member are not explicitly granted, the ancestors are visible ancestors. Visible ancestors are visible in the metadata tree and in reports, but their value will always be secured.

Visible ancestors ensure that there is a path from a root member of the hierarchy to any granted members. Without a path from a root member to granted members, Cognos Studios cannot properly display members.

Because Cognos Dynamic Cubes does not support visual totals, visible ancestors ensure that rollup values do not reveal information about secured descendants.

Visible ancestors occur only when using grant filters.

9.4.1 Example: Visible ancestors unnecessary

In this example, security view 1 grants access to United States, its ancestors, and its descendants. For all the granted members, their ancestor tree to the root is granted. There is no need for visible ancestors.

Security view 1: Grant United States ascendants and descendants

Figure 9-1 shows granted United States ascendants and descendants.

Figure 9-1 Granted United States ascendants and descendants (1)

Figure 9-2 shows the result.

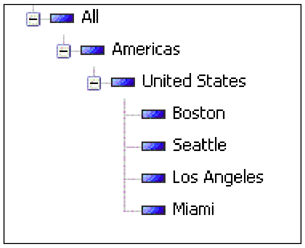


Figure 9-2 Granted United States ascendants and descendants (2)

Because Cognos Dynamic Cubes does not support visual totals, in this example the values for `All` and `Americas`, while unsecured, will not represent the rollup of their visible children, as seen in Figure 9-3.

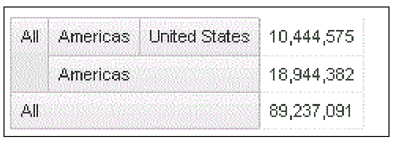


Figure 9-3 Values for all and Americas

Notice the values for `All` and `Americas` do not correspond to the value in the only visible descendant, `United States`.

9.4.2 Example: Basic visible ancestor

In this example, security view 2 grants access to United States and its descendants. The `All` member and `Americas` are not explicitly granted, but they are treated as visible ancestors.

Security view 2: Grant United States and descendants

Figure 9-4 shows granted `United States` and descendants.

Figure 9-4 Granted United States and descendants (1)

Figure 9-5 shows the result.

Figure 9-5 Granted United States and descendants (2)

`United States` and descendants are granted, so they have value. The `All` member and `Americas` are visible ancestors so they have their values secured, as the simple report shows in Figure 9-6. The values for any tuple that contains `All` or `Americas` are secured.

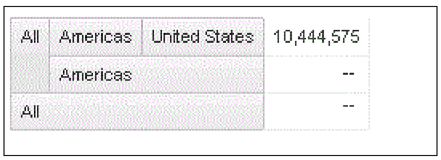


Figure 9-6 Report values for any tuple containing All or Americas

9.4.3 Example: Combination of visible and granted ancestors

Security view 3 illustrates the possibility of having a combination of granted and visible ancestors in a path to the hierarchy.

Security view 3: Grant Miami and descendants, grant Americas

Figure 9-7 shows the granted `Miami` and descendants and the `Americas`, as an example:

- Grant `Miami` and descendants
- Grant `Americas`

Figure 9-7 Granted Miami and descendants, the Americas (1)

Figure 9-8 shows the result.

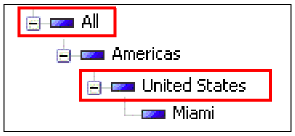


Figure 9-8 Granted Miami and descendants, the Americas (2)

For security view 3, `Miami` is granted, `United States` is a visible ancestor, `Americas` is granted, and the `All` member is a visible ancestor.

9.4.4 Examples: View merge impact on visible ancestors

Visible ancestors are based on the final view of the cube, based on security views of a user. If the combination of security views and filters results in a granted member being denied, a visible ancestor will not be included for that now-denied member. The following examples merge views with security view 4 to show the resulting visible ancestors. For more information about the consolidate view from merging views, see 9.8, "Merging security views or security filters" on page 272.

Security view 4: Grant Miami and descendants and grant Asia Pacific and descendants

Figure 9-9 shows the granted `Miami` and descendants and granted `Asia Pacific` and descendants, as an example:

- Grant Miami and descendants
- Grant Asia Pacific and descendants

Figure 9-9 Granted Miami and descendants and granted Asia Pacific and descendants (1)

Figure 9-10 shows the result.

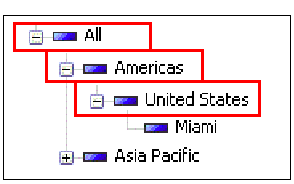


Figure 9-10 Granted Miami and descendants and granted Asia Pacific and descendants (2)

Security view 4 has visible ancestors in the `All` member, `Americas`, and `United States`.

Security view 5: Deny Americas

Figure 9-11 shows the `Americas` as denied.

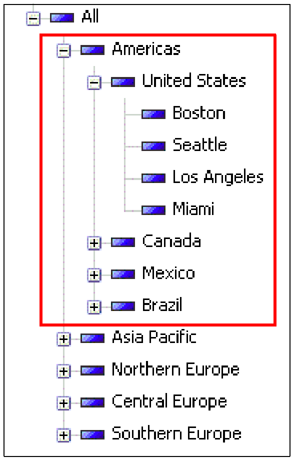


Figure 9-11 Denied Americas (1)

Figure 9-12 shows the result.

Figure 9-12 Denied Americas (2)

Security view 5 has no visible ancestors.

Consolidated view from merging security view 4 and security view 5

The visible ancestors, `Americas` and `United States`, that are required in view 4 are not in the consolidated view because the members were denied with view 5. The `All` member is still a visible ancestor in the consolidated view. Figure 9-13 shows this consolidated view.

Figure 9-13 Consolidated view of security views 4 and 5 (1)

Figure 9-14 shows the result.

Figure 9-14 Consolidated view of security views 4 and 5 (2)

Security view 6: Deny Miami and descendants

Figure 9-15 shows `Miami` and descendants as denied.

Figure 9-15 Denied Miami and descendants (1)

Figure 9-16 shows the result.

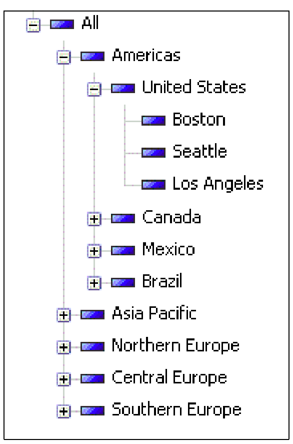


Figure 9-16 Denied Miami and descendants (2)

Security view 6 has no visible ancestors.

Consolidated view from merging security view 4 and security view 6

The visible ancestors, `Americas` and `United States`, that were in view 4 are not in the consolidated view because `Miami` is denied. Figure 9-17 shows this consolidated view.

Figure 9-17 Consolidated view of security views 4 and 6 (1)

Figure 9-18 shows the result.

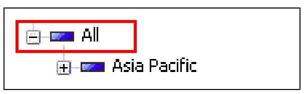


Figure 9-18 Consolidated view of security views 4 and 6 (2)

Security view 7: Grant Americas and descendants

Figure 9-19 shows the granted Americas and descendants.

Figure 9-19 Granted Americas and descendants (1)

Figure 9-20 shows the result.

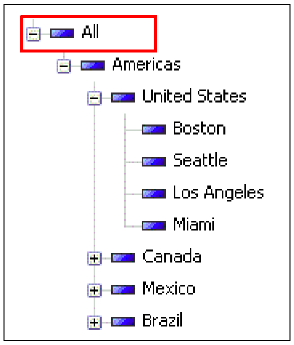


Figure 9-20 Granted Americas and descendants (2)

Security view 7 has a visible ancestor in the All member.

Consolidated view from merging security view 4 and security view 7

The visible ancestors, `Americas` and `United States`, that were in view 4 are now granted members in the consolidated view because of view 7. The `All` member is still a visible ancestor. Figure 9-21 shows this consolidated view.

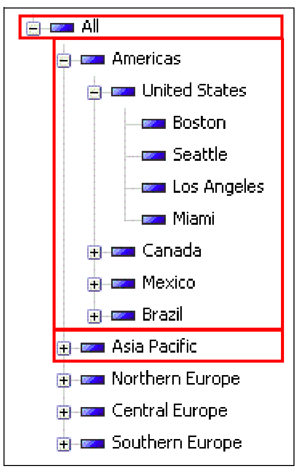


Figure 9-21 Consolidated view of security views 4 and 7 (1)

Figure 9-22 shows the result.

Figure 9-22 Consolidated view of security views 4 and 7 (2)

9.5 Secured padding members

Secured padding members ensure that the hierarchies remain balanced. Balanced, non-ragged hierarchies have better performance in the Cognos studios. Secured padding members are inserted into a secured hierarchy member tree when a granted member has all its children restricted. This situation (having the children restricted) is most common with grant members when granting only members and not including descendants in the scope. However, this situation can also occur with deny filters or with a combination of grant and deny filters.

Consider the following points:

- If a non-leaf member has all its descendants restricted, then secured padding members are inserted into all the levels below the non-leaf member.
- If all leaf members are restricted, padding members are inserted, but the leaf level is not removed.
- The caption of secured padding members is either empty or blank, or has the name of the parent. This is the same configuration setting for caption of padding member in ragged and unbalanced hierarchies.
- Secured padding members are secured in the same way that visible ancestors are secured.
- *Intrinsic* properties (properties that all members have, such as member unique name, level unique name, and so on) of secured padding members have correct values, but other member properties have null values.
- There is at most one secured padding member for each level under a parent member.

9.5.1 Example: Secured padding member unnecessary

This example describes securing a member and its descendants where a secured padding member is not needed.

Security view: Deny Americas and descendants

Figure 9-23 shows the denied `Americas` and descendants.

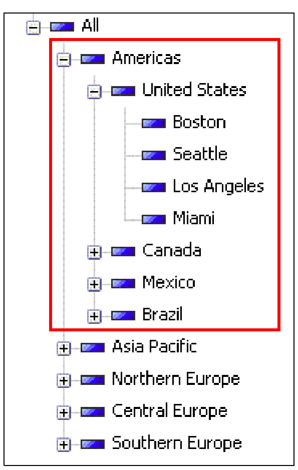


Figure 9-23 Denied Americas and descendants (1)

Figure 9-24 shows the result, which denies `Americas` and descendants.

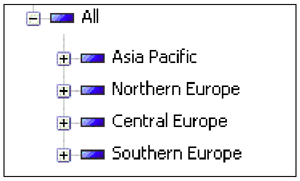


Figure 9-24 Denied Americas and descendants (2)

Securing Americas does not result in a ragged or unbalanced tree, so no secured padding member is needed.

9.5.2 Example: Secured padding member needed at leaf level

This example describes securing all children of a member at the leaf level, which results in an unbalanced hierarchy unless a secured padding member is inserted.

Security view: Deny children of United States

Figure 9-25 shows the denial of children of the `United States`.

Figure 9-25 Denial of children of the United States (1)

Figure 9-26 shows the result.

Figure 9-26 Denial of children of the United States (2)

Securing the children of the United States can result in an unbalanced hierarchy, so a secured padding member can be added to the member tree.

Note: The same security scenario can be created by using the `Grant` scope.

This security view grants the following items:

- All
- Asia Pacific, Northern Europe, Central Europe, Southern Europe, and their descendants
- Americas
- Canada, Mexico, Brazil, and their descendants
- United States

9.5.3 Example: Secured padding members needed on multiple levels

This example describes securing all children of a member, which requires a path of padding members to the leaf level.

Security view: Deny children of Americas and their descendants

Figure 9-27 shows the denial of children of `Americas` and their descendants.

Figure 9-27 Denial of children of Americas and their descendants (1)

Figure 9-28 shows the result.

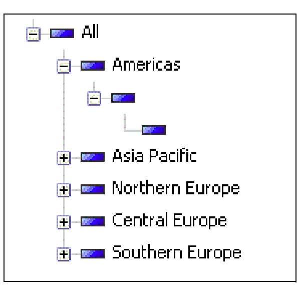


Figure 9-28 Denial of children of Americas and their descendants (2)

Securing the children of Americas can result in an unbalanced tree, so secured padding members are inserted at the leaf level.

9.6 Default members

When a hierarchy is secured, a new default member on the hierarchy can be specified for the user. The common scenario for modifying the default member is if a single member and descendants are granted access. In this case, it is as though the single member were the new root of the hierarchy, even when the member might not be at the root level.

The following steps determine the correct default member for a secured hierarchy:

1. The original default member is checked to ensure that it is not restricted and is not a visible ancestor. If the original default member is unsecured, then it remains the default member.
2. If the default member is secured, a *breadth first* search of the hierarchy is done to find the first level with an unsecured member:
   - If the first level with an unsecured member has only the one unsecured member, then the unsecured member is the new default member.
   - If the first level with an unsecured member has more than one unsecured member, or also has a visible ancestor on the level, then their common ancestor is the new default member. In some cases, this common ancestor can be a visible ancestor. In the case of a visible ancestor as a default member, any time a non-visible ancestor member is not the context in the report, the visible ancestor, whose value is always `ERR`, is the context.

> **Note:** If no member of a hierarchy where the default member is secured is included in a report, the data reflects the values associated with the default member selected this way. The data that is displayed can be confusing to the user, and can be improved by following these steps:
>
> - Create a calculated member that aggregates the members of the level at which security is applied. An example of expression for such member is:
>
>   `AGGREGATE(currentmeasure WITHIN SET members([Region]))`
> - Make this member the default member of the hierarchy.

Any time that a hierarchy with a visible ancestor as the default member is not explicitly included in the report, the default member is used in the context, and `ERR` is the cell value.

The same report run by a user with all access and a user with security policies normally hits the same cache. It varies, depending on the report structure, but in general, the secured user needs only a subset of the members that the unsecured user used, because security limits access to the members. However, when the default member differs between the two users, the slice of the cube differs, and a different section of the cache might be required.

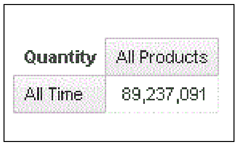
9.6.1 Default member examples

Take a simple crosstab report of `All Product` against `All Time on Quantity`. The security views have the `Branches` hierarchy secured, but the `Branches` hierarchy is not included in the report. The default member for the `Branches` hierarchy is the slicer for the report.

Security view 1: Grant all members

The default member for the `Branches` hierarchy remains `All Branches`.

The tuple value for security view 1 corresponds to `All Time`, `All Products`, `All Branches`, `Quantity`. See Figure 9-29.



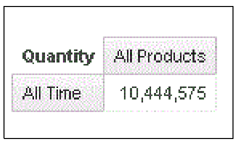
Quantity	All Products
All Time	89,237,091

Figure 9-29 Tuple value for security view 1

Security view 2: Grant United States and descendants

The default value for the `Branches` hierarchy is `United States`.

The tuple value for security view 2 corresponds to `All Time`, `All Products`, `United States`, `Quantity`. See Figure 9-30.



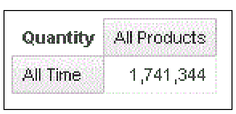
Quantity	All Products
All Time	10,444,575

Figure 9-30 Tuple Value for security view 2

Security view 3: Grant Brazil and descendants

The default member for the `Branches` hierarchy is `Brazil`.

The tuple value for security view 3 corresponds to `All Time`, `All Products`, `Brazil`, `Quantity`. See Figure 9-31.



Quantity	All Products
All Time	1,741,344

Figure 9-31 Tuple value for security view 3

Security view 4: Grant USA, Brazil, and their descendants

The default member for the `Branches` hierarchy is the common ancestor of `United States` and `Brazil`, and the visible ancestor is `Americas`.

The tuple value for security view 4 corresponds to `All Time`, `All Products`, `Americas`, `Quantity`. See Figure 9-32.

Quantity	All Products
All Time	--

Figure 9-32 Tuple value for security view 4

Because `Americas` is a visible ancestor, any tuple values it participates in are treated as error values.

9.6.2 Data caching

In the case of the unsecured user, the report in 9.6.1, “Default member examples” on page 270 uses the default member, `All Branches`, for the context of the `Branches` hierarchy. The tuple searched for in the data cache is (`All Time`, `All Products`, `All Branches`, `Quantity`).

In the case of a secured user assigned to security view 2 (United States and descendants), the report in 9.6.1, “Default member examples” on page 270 uses a different default member, `United States`, for the context of the `Branches` hierarchy. The tuple searched for in the data cache (`All Time`, `All Products`, `United States`, `Quantity`) differs from the unsecured user's tuple.

Because the tuples are not the same, reports that are run by one user do not populate the tuple value in data cache of the other. Also, because the `Branches` context is at different levels in the two tuples, the query structure to access the values in the underlying data source differs.

9.7 Calculated members

To secure a calculated member, the calculated member must be included in the dynamic query expression because most dynamic query expressions do not support calculated members. However, calculated members are included in the various scope options, such as Grant members, Grant members and descendants, and so on.

A calculated member is not accessible unless its parent member is accessible.

It might be possible that a calculated member definition references a secured member or measure. If a calculated member references a secured measure, a query with the calculated member returns an exception:

```
XQE-V5-0005 Identifier not found '[gosales_dw].[Measures].[Unit Sales]'.
```

If the calculated member references a secured member, the value of the secured member is treated as null in the calculation.

9.8 Merging security views or security filters

If a user is granted access to multiple views, the consolidated view of the cube is the union of the allowed sets, less the union of denied sets to which each view provides access.

Merging views where objects are secured follows the same behavior as merging security filters on a hierarchy. The consolidated view of the measures is the union of the granted objects, less the union of denied objects.

A user will never have access to a member if the member is *explicitly* denied access to the member in one or more of the security views to which the user is assigned.

Any explicit grant (including the built-in *Grant All Members* filter) takes precedence over a view that has no grant filters. In two scenarios, a security rule might not have any grant filters:

- If there are only deny filters for the hierarchy
- If there are no filters defined for the hierarchy

If there are no filters defined for the hierarchy, the security view likely has filters on other hierarchies.

A user who is assigned to one view with many filters should result in the same resulting view on the cube as one user who is assigned to many views, where the views have different filters. The only difference is if tuples are not possible in an underlying view, as described in 9.8.1, "Secured tuples" on page 273.

9.8.1 Secured tuples

Cognos Dynamic Cubes dimensional security supports defining only which member users have access. There is no support for defining security on specific tuples or cells. However, if a user is in multiple views, it is possible that the combination of views might display tuples that were not visible in any of the underlying views. If the tuple value is excluded from at least one of the underlying views, the tuple value is ERR in the final view.

For a tuple value to be visible, the tuple must be visible in at least one of the underlying views.

Security view 1

Security view 1 contains granted `United States`, `Outdoor Protection`, and their descendants. Figure 9-33 shows the tuple value.

Quantity		Outdoor Protection
Americas	United States	2,033,754

Figure 9-33 Tuple value for security view 1

Security view 2

Security view 2 contains granted `Brazil`, `Camping Equipment`, and their descendants. Figure 9-34 shows the tuple value.

Quantity		Camping Equipment
Americas	Brazil	752,338

Figure 9-34 Tuple value for security view 2

Combined security view 1 and view 2

Because the tuples (`Brazil`, `Outdoor Protection`) and `United States`, `Camping Equipment` are not visible in either of the underlying views, the tuples are ERR in the final views. Figure 9-35 shows the tuple values.

Quantity		Camping Equipment	Outdoor Protection
Americas	United States	--	2,033,754
	Brazil	752,338	--

Figure 9-35 Tuple values for combined security view 1 and view 2

9.8.2 Example 1

This example shows merging two views where the second view on its own has access to all but `Boston` and `Calgary`. However, because no grant filters are specified in security view 2, when merged with security view 1 that has grant filters, the grant filters take precedence. Also, the denied `Calgary` in security view 2 still applies in the consolidated view.

Security view 1

Figure 9-36 shows granting `Canada`, `Mexico`, `Brazil`, and their descendants.

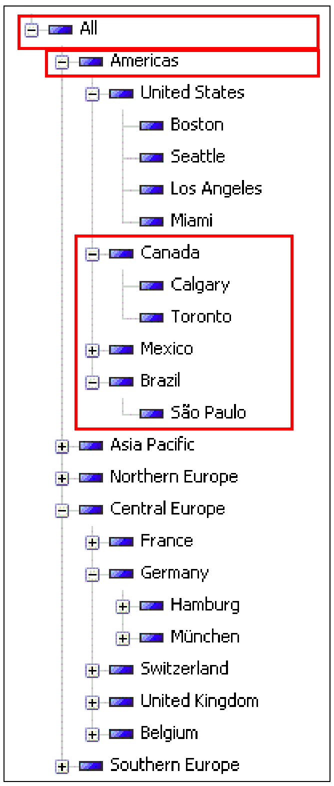


Figure 9-36 Security view 1: Grant Canada, Mexico, Brazil, and their descendants

Security view 2

Figure 9-37 shows denying `Boston` and denying `Calgary`.

Figure 9-37 Security view 2: Deny Boston, deny Calgary

Figure 9-38 shows the consolidated view for a user who is assigned to both the views.

Figure 9-38 Consolidated view for a user assigned to both views

The default member is `Americas`, even though it is a visible ancestor, because the first level to have an unsecured member, the `Country` level, had more than one unsecured member. Their common ancestor, `Americas`, is the default member.

9.8.3 Example 2

Similar to Example 1, the security view 2 has access to all but `Boston` and `Calgary`. However, because security view 2 was defined by using the `Grant All Members` filter, granting all members has the same weight as granting rules in security view 1.

Security view 1

Figure 9-39 shows granting `Canada`, `Mexico`, `Brazil`, and descendants.

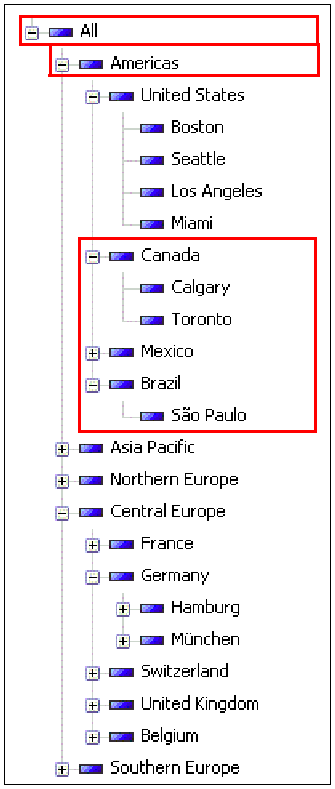


Figure 9-39 Security view 1: Grant Canada, Mexico, Brazil, and descendants

Security view 2

Figure 9-40 shows granting `All Members`, denying `Boston`, and denying `Calgary`.

Figure 9-40 Security view 2: Grant all members, deny Boston, deny Calgary

Figure 9-41 shows the consolidated view for a user who is assigned to both the views.

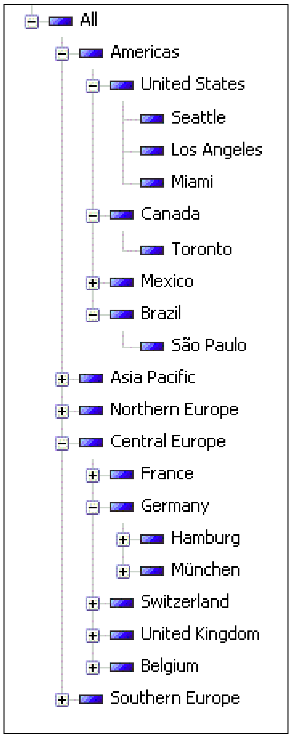


Figure 9-41 Consolidated view for a user assigned to both views

The default member continues to be `All` because it was granted in security view 2. `All` and `Americas` are not visible ancestors because they were granted in security view 2.

9.8.4 Example 3

This example shows how, when merging two views with only deny filters, the consolidated view has access to all member but those consolidated.

Security view 1

Figure 9-42 shows denying `Central Europe` and descendants.

Figure 9-42 Security view 1: Deny Central Europe and descendants

Security view 2

Figure 9-43 shows denying `Americas` and descendants.

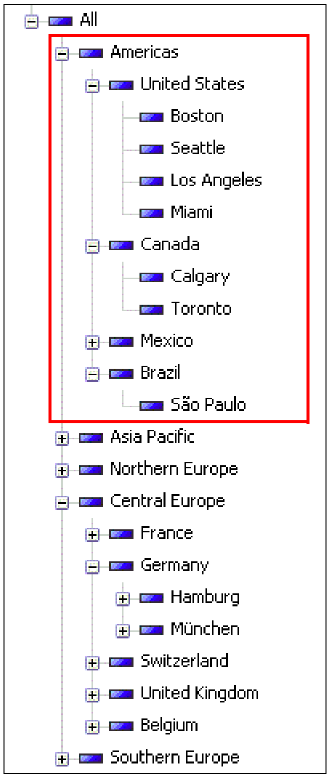


Figure 9-43 Security view 2: Deny Americas and descendants

Figure 9-44 shows the consolidated view for a user who is assigned to both views.

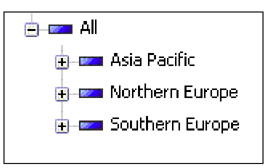


Figure 9-44 Consolidated view for a user assigned to both the views

The default member continues to be `All` because it was granted in security view 2.

9.8.5 Example 4

This example shows how merging two views can maintain secured padding members and how default members are chosen.

Security view 1

Figure 9-45 shows granting `United States` and descendants.

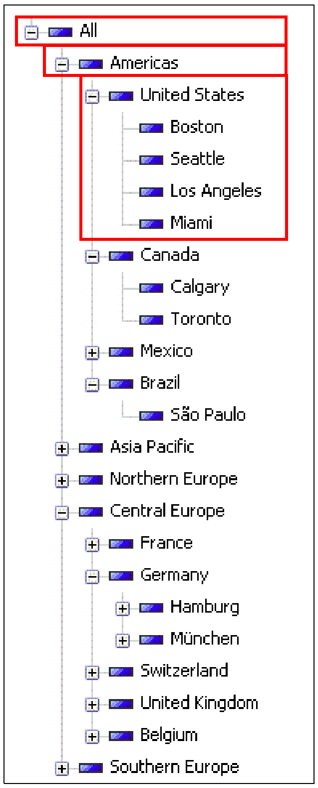


Figure 9-45 Security view 1: Grant United States and descendants

Security view 2

Figure 9-46 shows granting `Central Europe`, and denying children of `Central Europe` and their descendants.

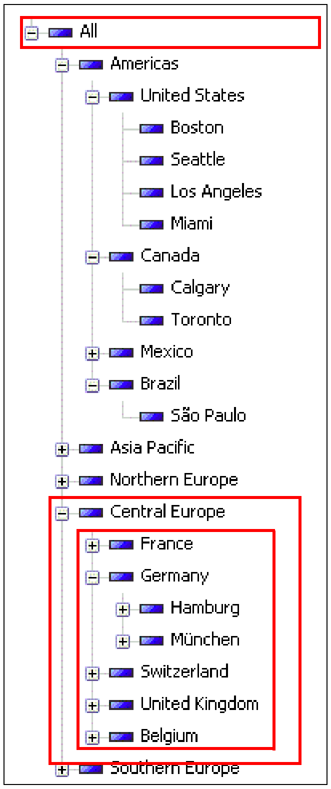


Figure 9-46 Security view 2: Grant Central Europe, deny children of Central Europe and their descendants

Security view 2 on its own has padding members under `Central Europe`. Figure 9-47 shows the consolidated view for a user who is assigned to both the views.

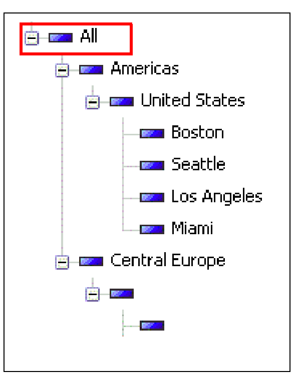


Figure 9-47 Consolidated view for a user assigned to both the views

The default member continues to be `All` because the first level with an unsecured member (`Central Europe`) also has a visible ancestor (`Americas`). Their common ancestor is the default member. `Americas` and the `All` member continue to be visible ancestors. Also, notice that the padding members are added under `Central Europe` because all the descendants were secured.

9.8.6 Example 5

This example shows how merging two views can require secured padding members.

Security view 1

Figure 9-48 shows granting `Americas`, and granting `United States` and descendants.

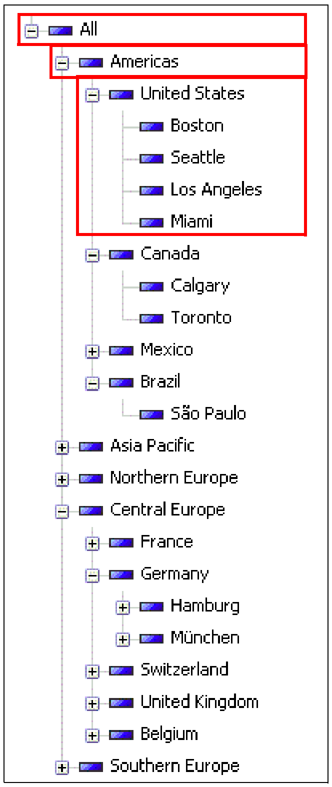


Figure 9-48 Security view 1: Grant America, grant United States and descendants

Security view 2

Figure 9-49 shows denying `United States` and descendants.

Figure 9-49 Security view 2: Deny United States and descendants

Figure 9-50 shows the consolidated view for a user assigned to both the views.

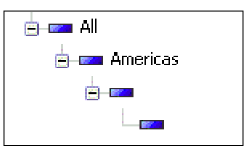


Figure 9-50 Consolidated view for a user assigned to both the views

The default member is `Americas`, and `All` member is a visible ancestor. Although neither security view 1 and security view 2 have padding members, their consolidated view requires padding members to stay balanced.

9.8.7 Example 6

This example shows how merging two views can restrict an entire hierarchy.

Security view 1

Figure 9-51 shows granting `United States` and descendants.

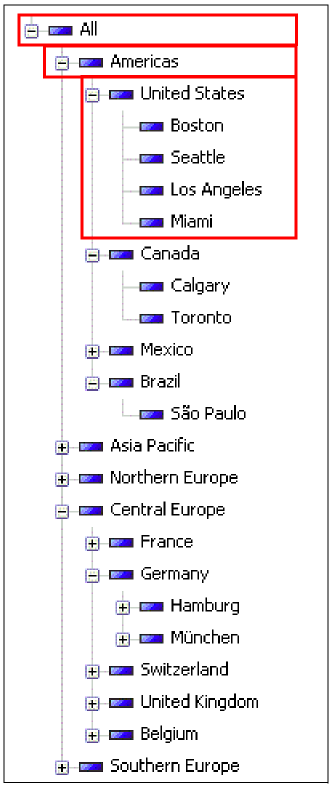


Figure 9-51 Security view 1: Grant United States and descendants

Security view 2

Figure 9-52 shows denying `United States` and descendants.

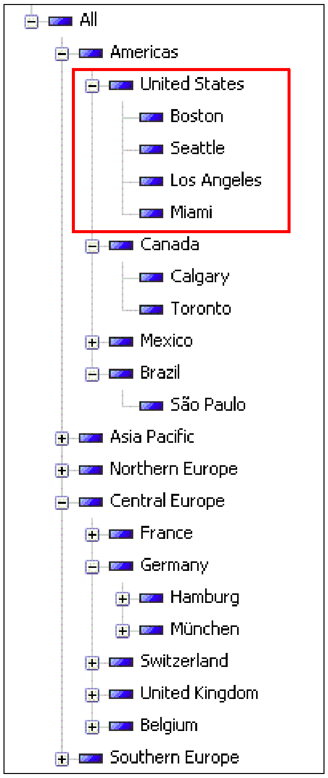


Figure 9-52 Security view 2: Deny United States and descendants

When the two views are merged, no unsecured members remain. The members that are granted in view 1 are denied in view 2. There are no members for the user to see and the hierarchy is secured.

9.8.8 Example 7

This example shows merging a view that has filters on a hierarchy with a view that has no filters for the hierarchy. Assume that security view 2 has filters on another hierarchy, but the `Branches` hierarchy is unsecured.

Security view 1

Figure 9-53 shows granting `United States` and descendants.

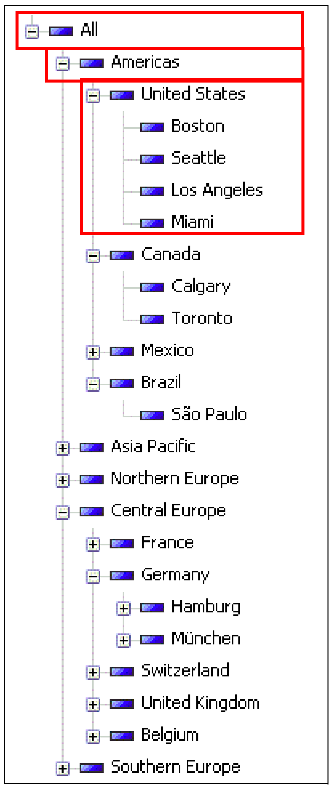


Figure 9-53 Security view 1: Grant United States and descendants

Security view 2

Figure 9-54 shows that no filters are on the hierarchy.

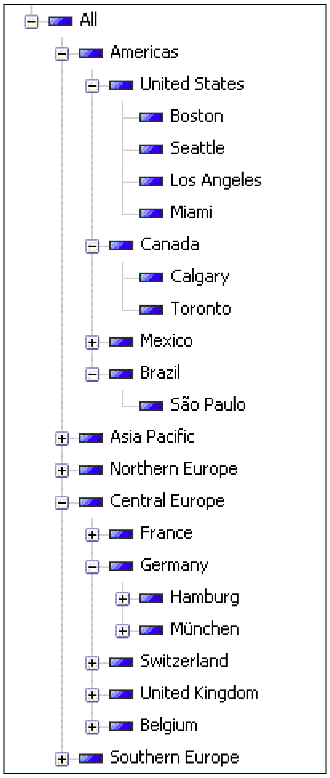


Figure 9-54 Security view 2: No filters on the hierarchy

Figure 9-55 shows the consolidated view for a user who is assigned to both the views.

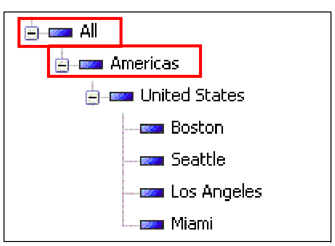


Figure 9-55 Consolidated view for a user who is assigned to both views

When the two views are merged, because there are no filters on security view 2, only the filters from security view 1 are applied. This behavior differs from view with a `Grant All Members` filter.

9.9 Security lookup tables

The security needs of large organizations might require providing fine-grained access to data for individual users. For example, in a sales organization each salesman might need to have access to a small part of a `Geography` hierarchy. When the number of such users grows, defining and managing individual security filters based on *member expressions* becomes labor-intensive.

In such scenarios, security filters that are based on *lookup tables* are preferred.

Security lookup table is a database table that contains information about users and the sets of members that those users are granted or denied access to. Because this is a regular table, it can be updated with SQL queries or stored procedures as part of an ETL process or edited manually with a database management tool.

Security lookup tables typically contain three types of data:

- *User identification.* Data used to identify a user, for example, user, group, and role names.
- *Security metadata.* Security rule type indicators (for example, `grant/deny`), rule scope (for example, `member`, `member and descendants`, and so on), and any other data that can be used to select the correct lookup table rows for the current rule type. These types of data columns are not strictly necessary, but they often help to define tables and lookup expressions in an efficient way.
- *Member keys.* Key values used to uniquely identify members in the hierarchy.

IBM Cognos samples databases currently do not contain security lookup tables. To demonstrate lookup table functions, you need to add one to the `gosldw_sales` data source.

To create a sample security lookup table, run the SQL statement shown in Example 9-1.

Example 9-1 SQL statement to create a sample security lookup table

```
CREATE TABLE "GOSALESDW"."PRODUCT_SECURITY_TABLE"
(
   USER_NAME varchar(20),
   SECURITY_TYPE varchar(20),
   PRODUCT_TYPE_KEY int,
   BASE_PRODUCT_KEY int
)
;
```

After the table is created, add the two rows to it as shown in Table 9-1.

Table 9-1 Sample security lookup table

USER_NAME	SECURITY_TYPE	PRODUCT_TYPE_KEY	BASE_PRODUCT_KEY
awhite	grant	967	
awhite	deny		100

Both rows in this table correspond to user `awhite` (Alan White). The first row is used to grant access to member with key value 967, which is `First Aid`. The second row is used to deny access to member with key value 100, which is `Insect Bite Relief`.

Note:

- A missing key value in a column is denoted by a NULL value or an empty string.
- NULL values or empty strings in all key columns in a lookup table row denote an `All` member (because `All` members have no corresponding key values).

A security lookup table rule takes advantage of the lookup table and ties together the user context information and keys for members that are granted or denied for the user or group of users.

A security lookup table rule consists of the following elements:

- *Rule scope.* This is the same scope as the one used for regular member security filters: `Grant Member`, `Grant Member` and descendants, and so on.
- *Query subject.* The reference to the query subject object defined in the cube model that references the columns that you need in a lookup table. Multiple query subjects can be defined.
- *Level key filters.* A mapping between level keys in the hierarchy and the key columns in the lookup table where these values can be found. Not all key fields must be present in the lookup table, only the ones that correspond to levels that you are securing.
- *Lookup filter expression.* The expression that filters the lookup table based on the current user context and other parameters. The lookup filter expression selects a set of keys for members that must be granted or denied access for the current user and the rule scope and type. It commonly uses macro expressions to access the current user name and other attributes in LDAP. Internally this expression becomes an SQL WHERE clause in the query that selects rows with appropriate member keys from the lookup table.

Tips:

- When defining the lookup filter expression, first try it as a filter in a relational report against the lookup table in Cognos Report Studio. This way that you can test the expression syntax and make sure that, depending on different user context, the expression selects the correct member keys from the lookup table.
- When using the equals (=) operator in the lookup filter expression, it might be necessary to trim the white space from the values being compared because extra white space might cause the values that are visually equivalent to not match.
- Filters based on member expressions can be used together with filters based on security lookup tables to simplify management of some security scenarios and receive benefits of both simplicity of expression-based security and the granularity of lookup table-based approach.

  For example, consider the need to secure access to sales region data for employees in the sales department in the situation where users from many other departments are also accessing the cube. A lookup table denying access to the Sales hierarchy to most users in the company, and selectively giving access to it to the salesmen can be difficult to manage because of its potential size.

  However, one might create an expression-based filter that denies access to the `Sales` hierarchy to anyone not in the `Sales` group, and then build a lookup table (and a corresponding filter) only for users from the sales department that controls access to different areas of the `Sales` hierarchy. Such table would be smaller and easier to maintain.

9.10 Defining security in Cognos Cube Designer

To create security views in Cognos Cube Designer, complete the following tasks:

1. Decide which hierarchies you are interested in having secured.
2. Define the appropriate security filters.
3. Decide which dimensions, measures, and attributes you want to secure.
4. Define the security views.
5. Publish the cube.
6. Assign users to security views in IBM Cognos Administration.

For this scenario, assume that you are interested in creating two security views that secure the `Products` hierarchy of the `gosldw_sales` cube.

Security view 1

Figure 9-56 shows granting access to `Outdoor Protection` and its descendants, and denying access to `Insect Repellents` and its descendants.

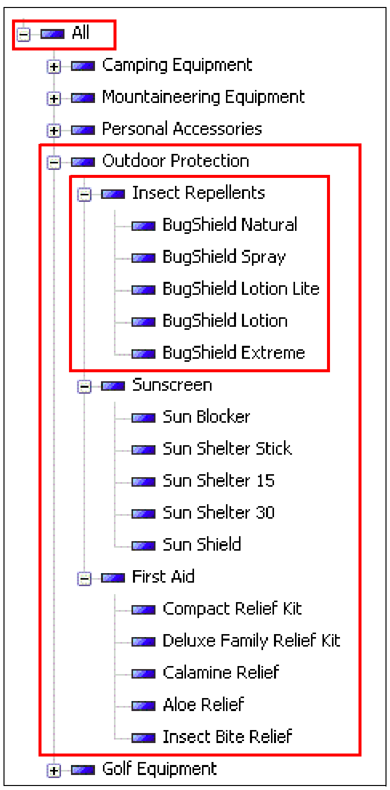


Figure 9-56 Security view 1

Figure 9-57 shows the result.

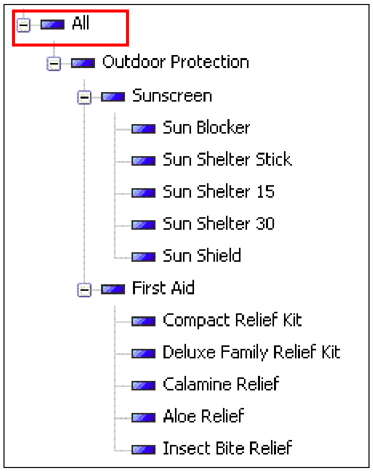


Figure 9-57 Result of security view 1

Security view 2

Figure 9-58 shows granting access to `First Aid` and descendants, and denying access to `Insect Bite Relief`.

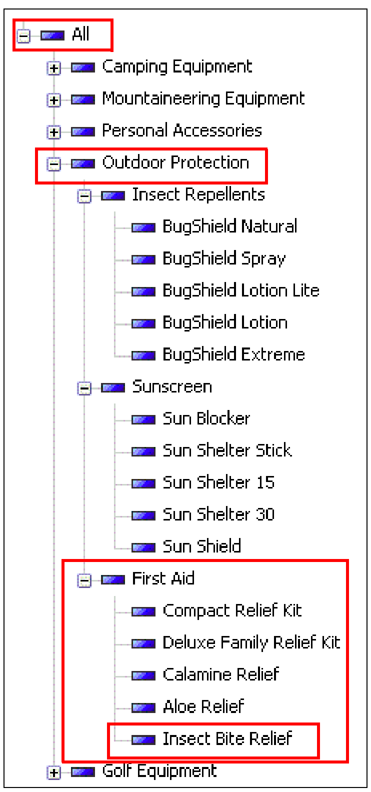


Figure 9-58 Security view 2

Figure 9-59 shows the result.

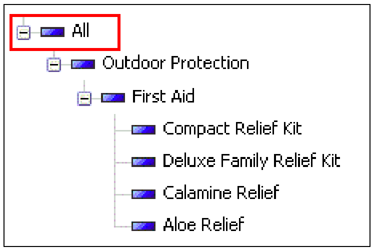


Figure 9-59 Result of security view 2

Figure 9-60 shows the resulting behavior when a user is assigned to both security view 1 and security view 2.

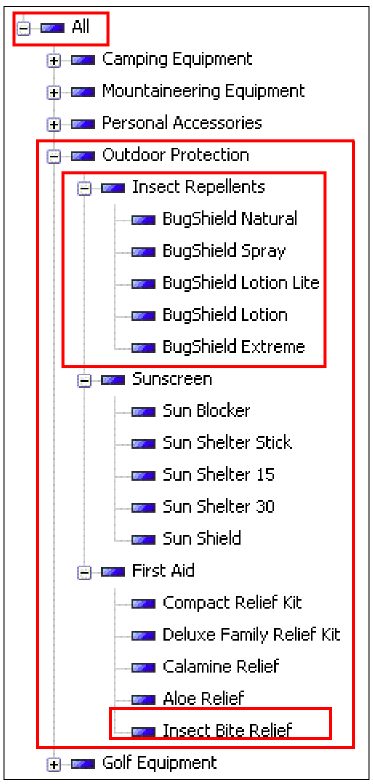


Figure 9-60 User is assigned to security view 1 and security view 2

Figure 9-61 shows the result.

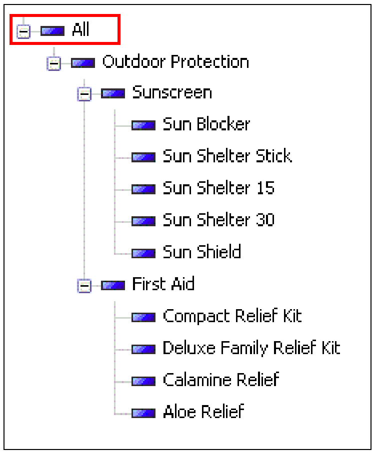


Figure 9-61 Result of user assigned to security view 1 and security view 2

9.10.1 Selecting the hierarchy on which to define the security filter

Use the following steps to model security on the Products hierarchy:

1. Open IBM Cognos Cube Designer with the cube model created.
2. In the Project Explorer, expand the `Products` dimension.
3. Double-click the `Products` hierarchy to open the level editor pane.
4. In the hierarchy editor pane, select the Security tab.

Figure 9-62 shows modeling security on the products hierarchy.

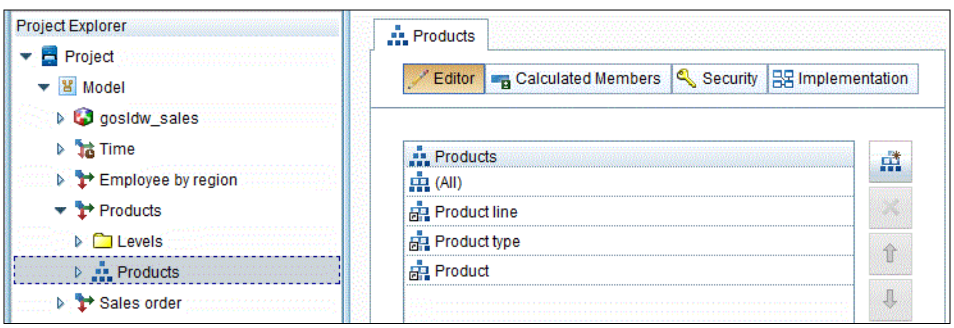


Figure 9-62 Modeling security on the Products hierarchy

Note the built-in security `All Members Granted` line in the `Security Filters` list (Figure 9-63). Every hierarchy has this built-in filter that grants access to all members in the respective hierarchy.

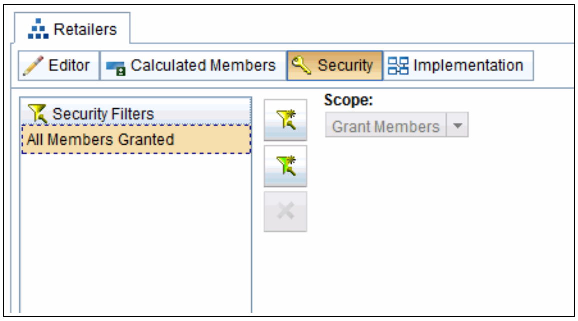


Figure 9-63 All member granted in the security filters

9.10.2 Creating security filters

Use the steps in this section to define the security filters required to model security view 1 and security view 2.

Grant Outdoor Protection and descendants

Complete the following steps:

1. Click the **Add Security Filter** icon to add a security filter (Figure 9-64).

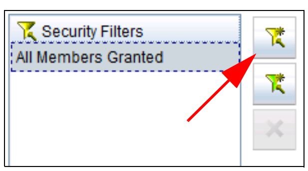


Figure 9-64 Adding a security filter

2. Select the **New Security Filter** and press F2 to rename the security filter to `Grant Outdoor Protection and Descendants`. Figure 9-65 shows how to rename the Security Filter.

Figure 9-65 Renaming the security filter

3. From the **Scope** menu (Figure 9-66), select **Grant Members and Descendants** as the appropriate scope.

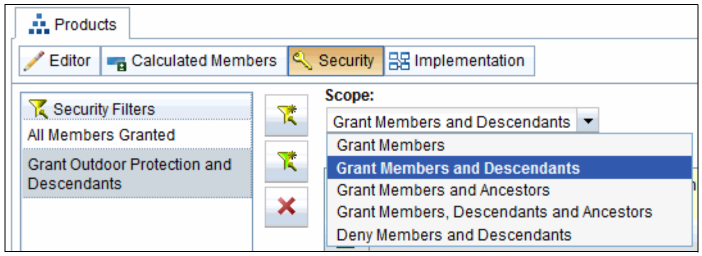


Figure 9-66 Scope drop-down list

4. Use the member browser to add members to the expression editor.
5. In the Project Explorer, expand the `Products` hierarchy.
6. Expand the `Members` folder.
7. Navigate to `Outdoor Protection` by expanding the `All` member.
8. Select the `Outdoor Protection` member and drag it into the expression editor.

   Figure 9-67 shows dragging `Outdoor Protection`.

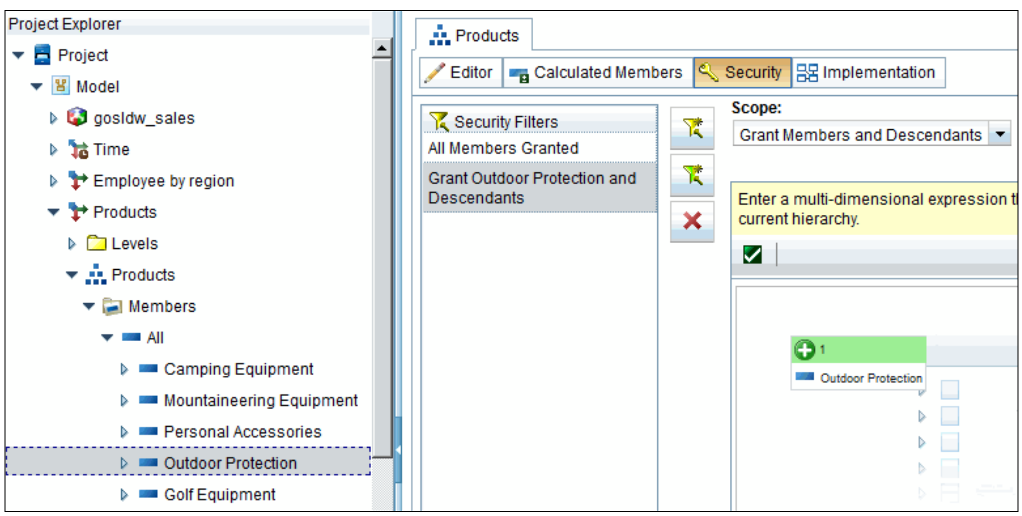


Figure 9-67 Dragging the Outdoor Protection

9. Click **Validate** (button with a check mark) to ensure that the expression is valid.

Deny Insect Repellents and descendants

Complete the following steps to create the second filter in security view 1:

1. Click the **Add Security Filter** icon to add a security filter.
2. Select the **New Security Filter** and press F2 to rename the security filter to `Deny Insect Repellents and Descendants`.
3. From the **Scope** menu, select **Deny Members and Descendants** as the appropriate scope.
4. In the Project Explorer, expand the `Products` hierarchy.
5. Expand the `Members` folder.

6. Navigate to `Insect Repellents` by expanding the `All` member, then `Outdoor Protection`.
7. Select the `Insect Repellents` member and drag it into the expression editor.

   Figure 9-68 shows creating the second filter in security view 1.

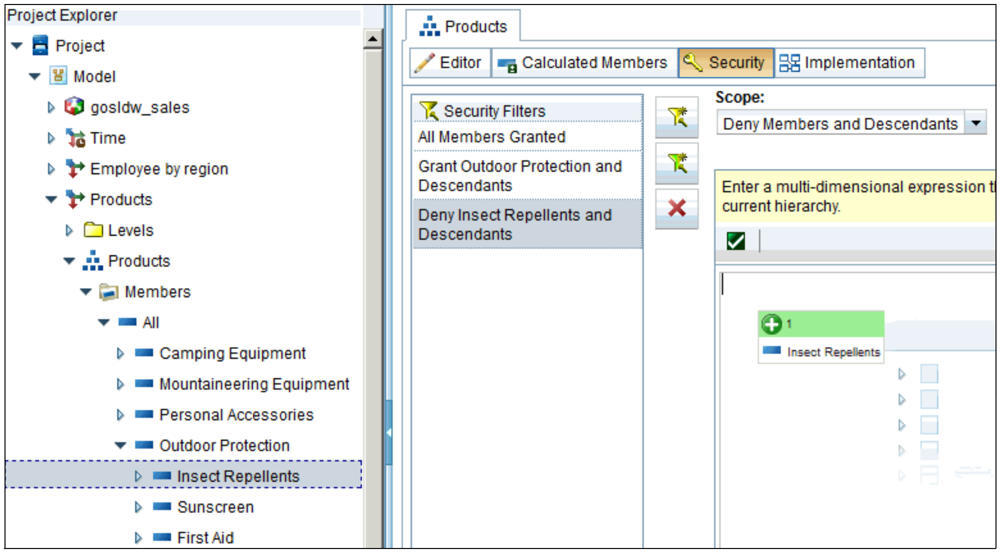


Figure 9-68 Creating the second filter in security view 1

8. Click **Validate** to ensure that the expression is valid.

Define query subject

Create a query subject based on the sample lookup table described in 9.9, “Security lookup tables” on page 288.

Complete the following steps to create a query subject element:

1. Right-click `Model` in the `Project Explorer` tree, and then select **New** → **Query Subject**.
2. Query subject New Query Subject is displayed. Press **F2** to rename it to `PRODUCT_SECURITY`.
3. Double-click the query subject name to open the editor.
4. In the Source pane, expand `Tables`, and then expand the PRODUCT_SECURITY_TABLE table.

5. Select four columns in the table (USER_NAME, SECURITY_TYPE, PRODUCT_TYPE_KEY, and BASE_PRODUCT_KEY), and drag the columns into the query subject editor pane as shown in Figure 9-69.

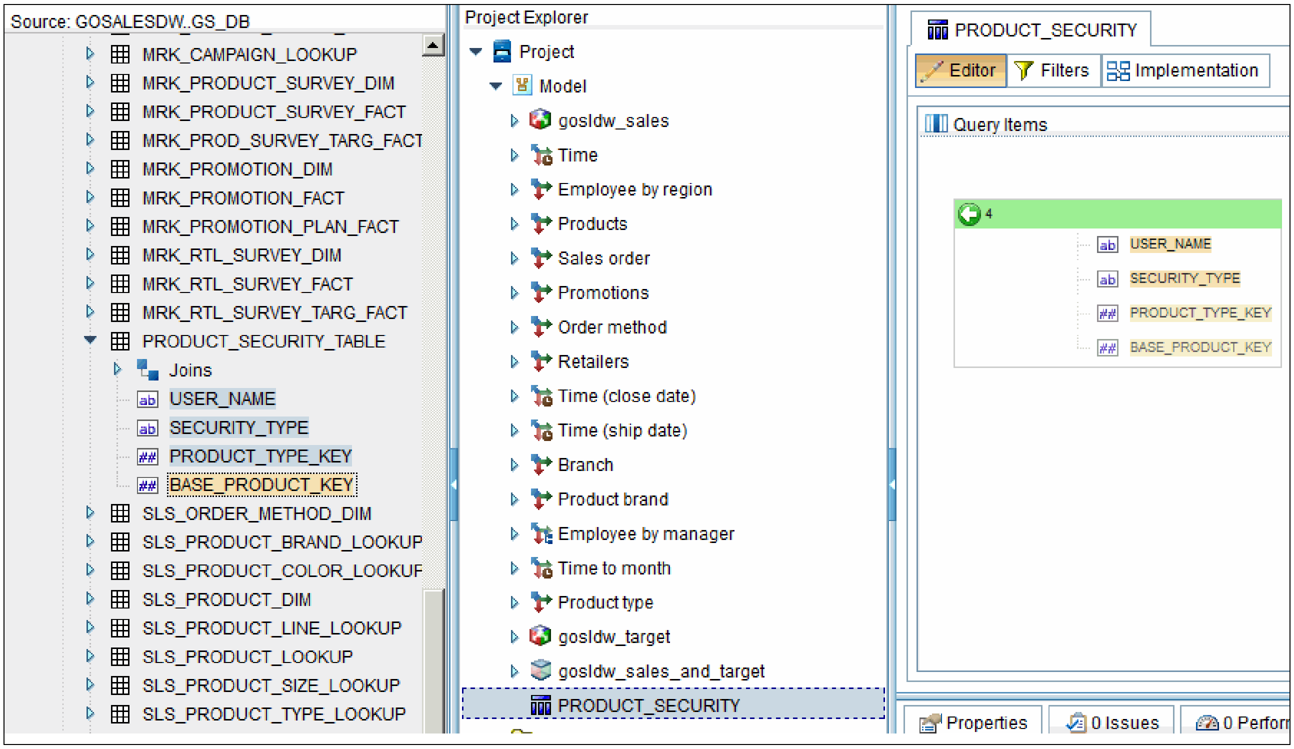


Figure 9-69 Query subject definition

6. Right-click the query subject and select **View Data** to verify that the query subject is working as expected and loads the data from the lookup table correctly.

Grant Products with Lookup Table

Complete the following steps to create the first filter in security view 2:

1. Traverse back to the `Products` hierarchy Security tab.
2. Click the **Add Lookup table based Security Filter** icon to add a security filter as shown in Figure 9-70.

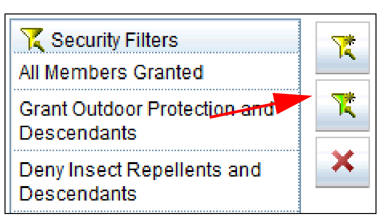


Figure 9-70 Adding a Lookup table based Security Filter.

3. Select the **New Lookup Security Filter** and press **F2** to rename the security filter to `Grant Products with Lookup Table`.
4. From the **Scope** menu, select **Grant Members and Descendants** as the appropriate scope.
5. From the **Query Subject** menu, select PRODUCT_SECURITY.

6. Edit the **Level Key Filters** field to map the `[Product type].[Product type key]` level key to `Product Type Key` filter item, and the `[Product].[Base product key]` level key to the `Base Product Key` filter item.
7. Expand the PRODUCT_SECURITY query subject in the model tree, and drag the items `User Name` and `Security Type` into the **Lookup Filter Expression** field.
8. Edit the **Lookup Filter Expression** field to make dropped fields part of the expression, as shown in Figure 9-71. This filter grants access to members and their descendants defined in the lookup table key columns (`First Aid` in this example). The records in the lookup table matches the current user name in the User Name column and contains `grant` in the Security Type column.

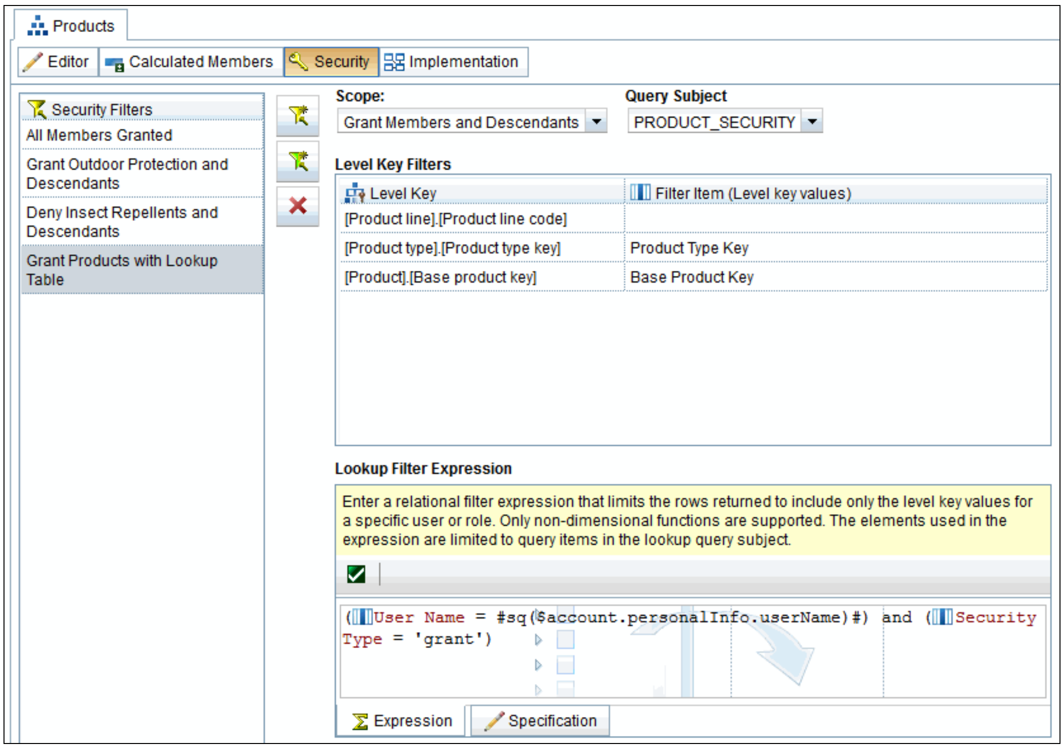


Figure 9-71 Lookup table grant filter

9. Click **Validate** to ensure that the expression is valid.

Deny Products with Lookup Table

Complete the following steps to create the second filter in security view 2:

1. Click the **Add Lookup table based Security Filter** icon to add a security filter.
2. Select the **New Lookup Security Filter** and press **F2** to rename the security filter to `Deny Products with Lookup Table`.
3. From the **Scope** drop-down list, select **Deny Members and Descendants** as the appropriate scope.
4. From the Query Subject drop-down list, select PRODUCT_SECURITY.
5. Edit the **Level Key Filters** field to map the `[Product type].[Product type key]` level key to `Product Type Key` filter item, and the `[Product].[Base product key]` level key to the `Base Product Key` filter item.
6. Expand the PRODUCT_SECURITY query subject in the model tree, and drag the items `User Name` and `Security Type` into the **Lookup Filter Expression** field.

7. Edit **Lookup Filter Expression** field to make dropped fields part of the expression, as shown in Figure 9-72**.** This filter denies access to members and their descendants defined in the lookup table key columns (`Insect Bite Relief` in this example). The records in the lookup table match the current user name in the User Name column and contain `deny` in the Security Type column.

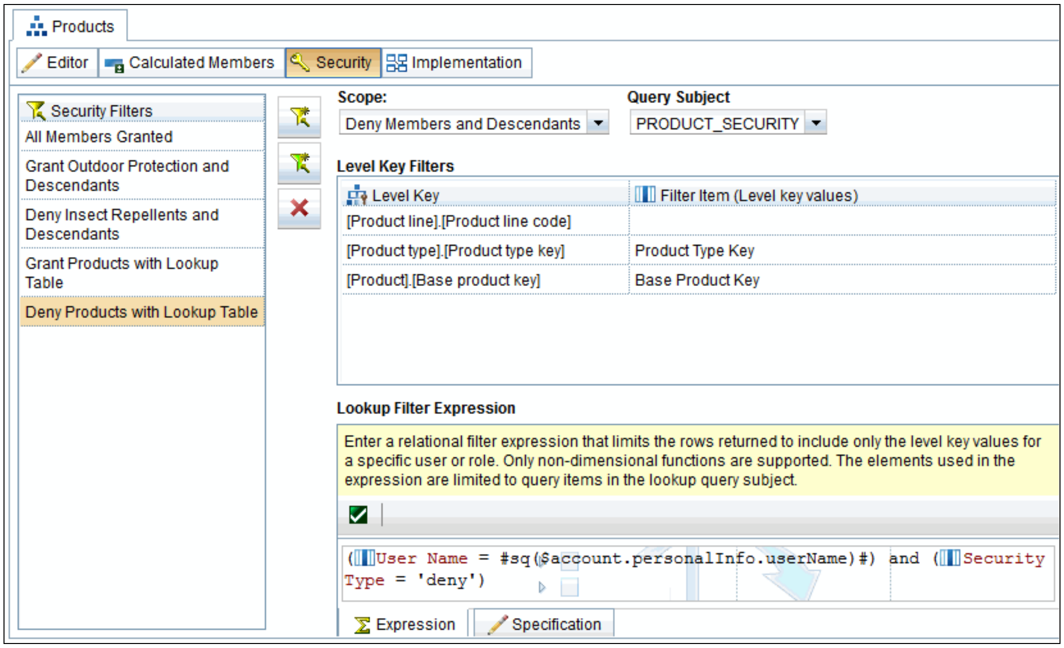


Figure 9-72 Lookup table deny filter

8. Click **Validate** to ensure that the expression is valid.

9.10.3 Creating security views

This section describes how to create security views after creating the necessary security filters.

Security view 1

Complete the following steps to create security view 1:

1. In Project Explorer**,** click the `gosldw_sales` cube.
2. Select the Security tab.
3. Click the **Add Security Filter** icon to create a security view. Figure 9-73 shows creating a security view.

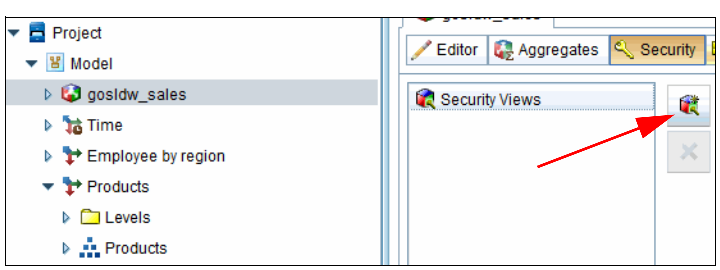


Figure 9-73 Creating a security view

4. Select **New Security View** and press **F2** to rename the security view to `Grant Outdoor Protection and Deny Insect Repellents`.
5. Select the Members tab.
6. Click the **Add Secured Member** icon to add security filters to the security view. Figure 9-74 shows adding security filters to the security view.

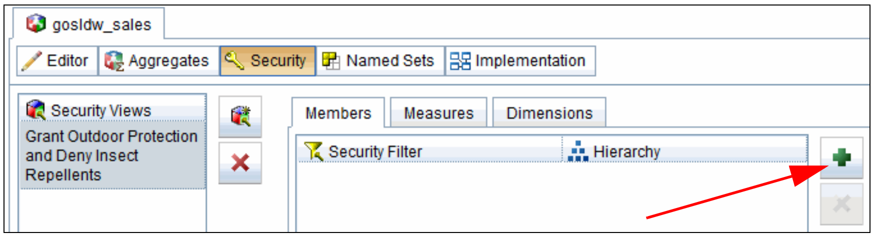


Figure 9-74 Adding security filters to security view

7. In the Add Security Filters window, select the security filters that you want to add to the security view. For security view 1, expand the `Products` hierarchy, select **Grant Outdoor Protection and Descendants** and **Deny Insect Repellents and Descendants** (Figure 9-75).

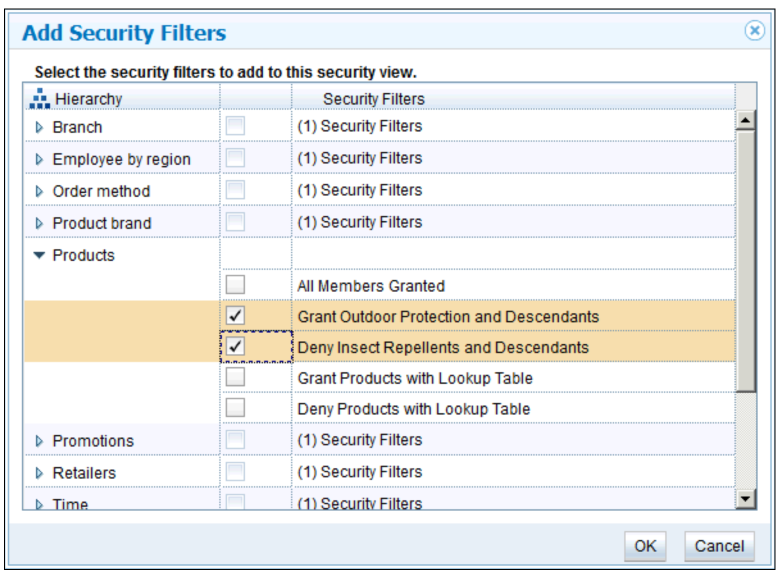


Figure 9-75 Selecting security filters

8. Click **OK**. Figure 9-76 show the selected security filters.

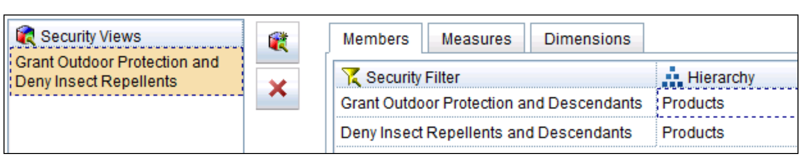


Figure 9-76 Displays the selected security filters

Security view 2

Complete the following steps to create security view 2:

1. Click the **Add Security Filter** icon to create a security view.
2. Select **New Security View** and press **F2** to rename the security view to `Secure with Lookup Table`.
3. Select the Members tab.
4. Click the **Add Secured Member** icon to add security filters to the security view.
5. Select the security filters that you want to add to the security view from the Add Security Filters window. For security view 2, select **Grant Products with Lookup Table** and **Deny Products with Lookup Table.**
6. Click **OK**. Figure 9-77 shows the selected security filters.

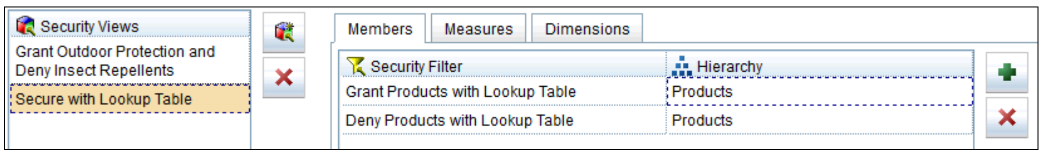


Figure 9-77 Displays the selected security filters

9.10.4 Securing measures

Measures can also be secured, but are modeled in a slightly different way than hierarchy members. The grant and deny rules work the same for measures as they do for hierarchy members.

Complete the following steps to secure the measures in `Grant Outdoor Protection` and `Deny Insect Repellents`:

1. Select the Measures tab.
2. Click the **Add Secured Measure** icon to the right of the Secured Measures editor (Figure 9-78).

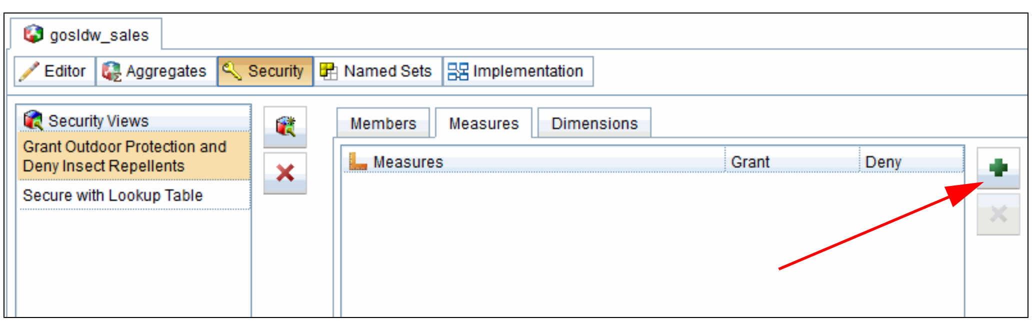


Figure 9-78 Adding the secured measures

3. In the Add Measures window, select the access to the measures you are selecting, either grant or deny. For this example, select **Grant Select Members.**
4. Select the measures you want to grant access to: **Unit Cost**, **Unit Price**, and **Unit Sale Price**.

Figure 9-79 shows selecting the Grant Select Members.

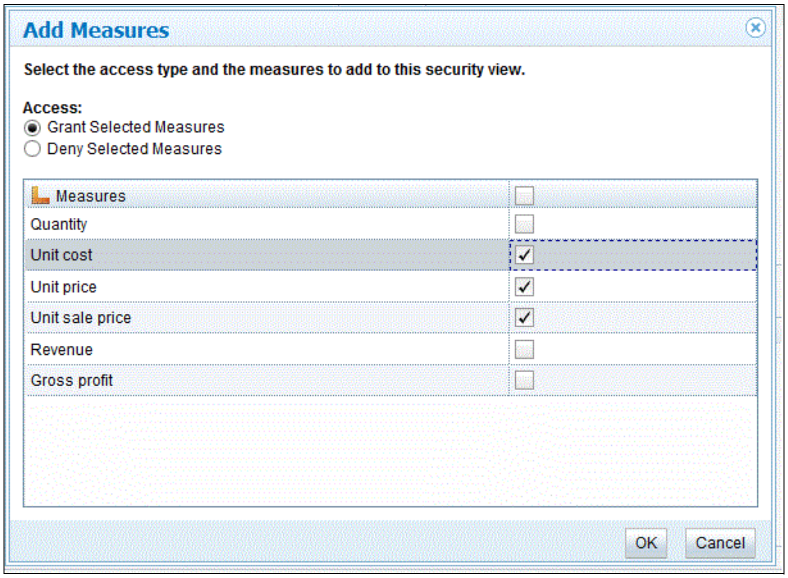


Figure 9-79 Select Grant Selected Measures

5. Click **OK.** The Secured Measures pane now shows the granted measures. If changes were warranted, the other radio button can be selected or the measure can be removed from the list. Figure 9-80 shows the granted measures.

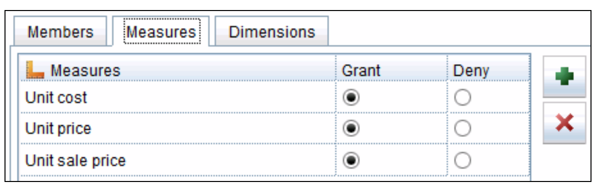


Figure 9-80 Displays the Granted Measures

6. Click the **Add Secured Measure** icon to the right of the Secured Measures editor.
7. In the Add Measures window, notice that only the measures that were not previously selected are listed. Click **Deny Select Members**.
8. Select the check box for the measures that you want to deny access to, **Gross Profit**.

Figure 9-81 shows selecting **Deny Selected Measures**.

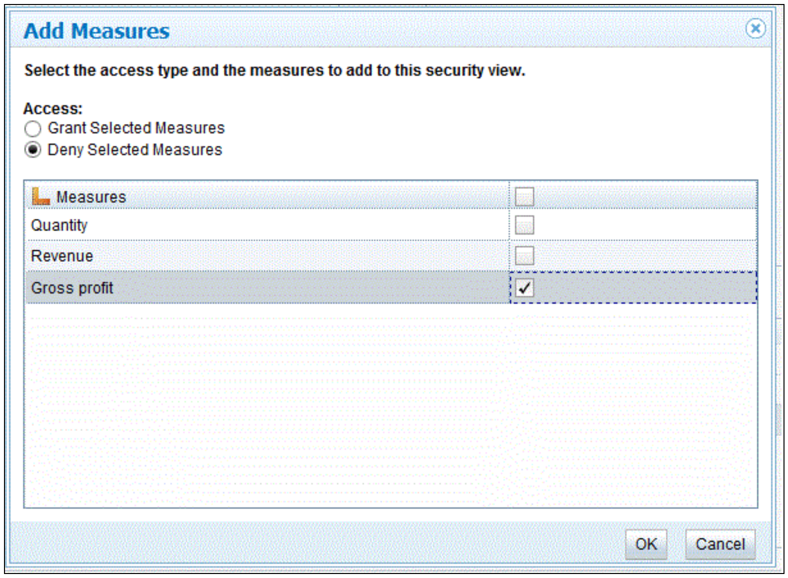


Figure 9-81 Selecting the Deny Selected Measures access

9. Click **OK**.

   The Secured Measures pane now shows the granted measures and the denied measures for view 1. Revenue and Quantity are not included in the secured measures list (implicitly denied due to an explicit grant), similarly to hierarchy members not being included in a grant or deny filter expression.

   Figure 9-82 shows granted measures and denied measures of `Grant Outdoor Protection` and `Deny Insect Repellents`.

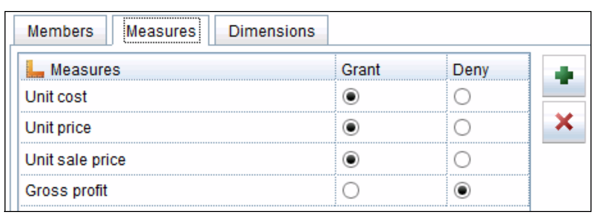


Figure 9-82 Granted Measures and Denied Measures for Grant Outdoor Protection and Deny Insect Repellents

9.10.5 Securing dimensions

The process to secure dimensions is similar to the process to secure measures.

Complete the following steps to secure the dimensions in `Grant Outdoor Protection` and `Deny Insect Repellents`:

1. Select the **Dimensions** tab. You will see two editors. On the top is the Secured Dimensions editor. On the bottom is the Secured Attributes editor.
2. Click the **Add Secured Dimension** icon to the right of the Secured Dimensions editor (Figure 9-83).

Figure 9-83 Adding secured dimension

3. Click **Deny Selected Dimensions**.
4. Select **Product brand** as shown in Figure 9-84.

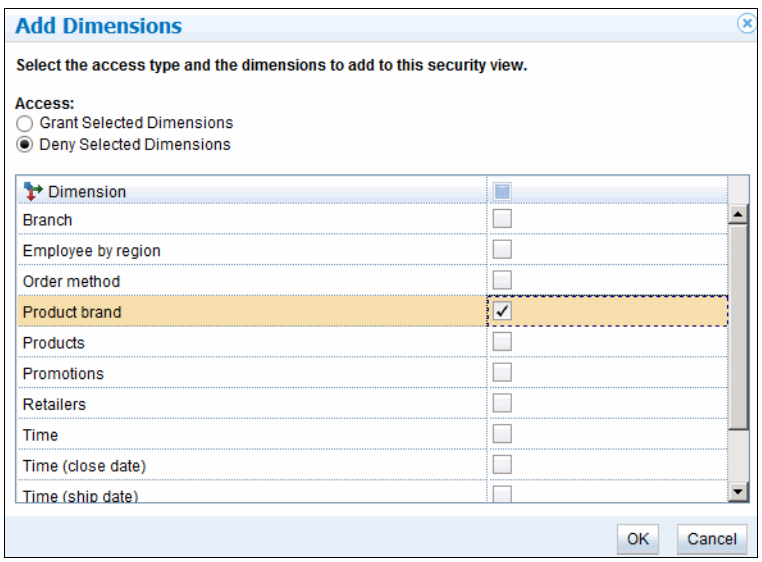


Figure 9-84 Denying Product Brand dimension

5. Click **OK**.

   The secured Dimensions editor now shows that `Product` brand has been denied for `Grant Outdoor Protection` and `Deny Insect Repellents`.
6. Click **Secure with Lookup Table**.

   The Secured Dimensions editor now shows that no dimension has been denied or granted for the Secure with Lookup Table.
7. Click **Grant Outdoor Protection and Deny Insect Repellents**.

9.10.6 Securing attributes

The process to secure attributes is similar to the process to secure measures and dimensions.

Complete the following steps to secure the attributes in `Grant Outdoor Protection` and `Deny Insect Repellents`:

1. Select the **Dimensions** tab. You will see two editors. On the top is the Secured Dimensions editor. On the bottom is the Secured Attributes editor.
2. Click the **Add Secured Attribute** icon to the right of the Secured Attributes editor (Figure 9-85).

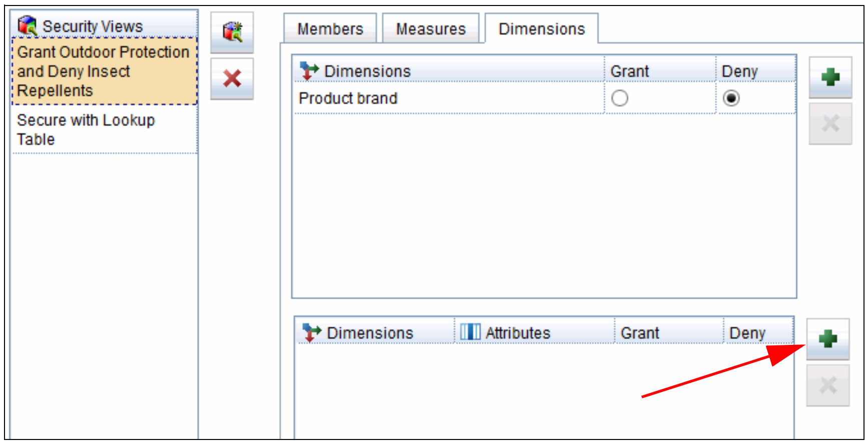


Figure 9-85 Adding secured attribute

3. Click **Deny Selected Attributes**.
4. Expand `Branch`.
5. Select the check boxes for **[Branch].[Address 1]**, **[Branch].[Address 2]**, **[Branch].[Province or State]**, **[Branch].[Address 1 (multiscript)]**, **[Branch].[Address 2 (multiscript)]**, **[Branch].[City (multiscript)]**, **[Branch].[Province or State (multiscript)]**, and **[Branch].[Postal Zone]** (Figure 9-86).

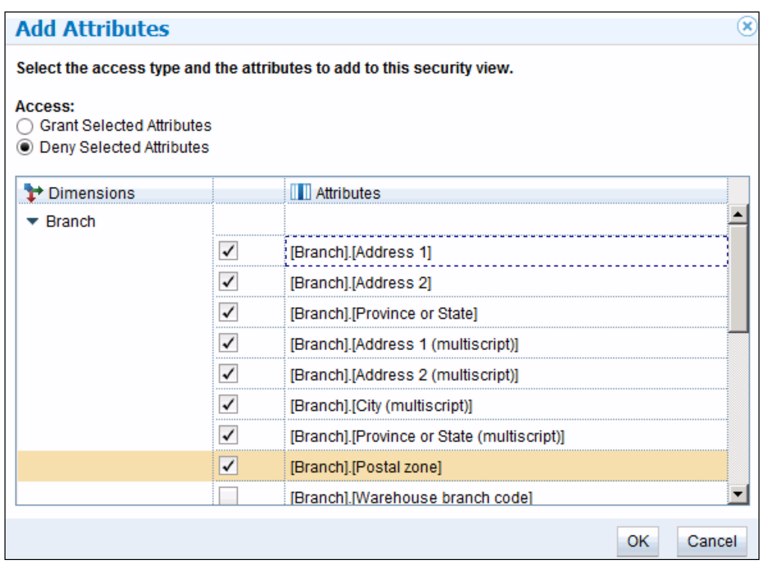


Figure 9-86 Securing attributes

6. Click **OK**.

The Secured Attributes editor now shows that those attributes have been denied for `Grant Outdoor Protection` and `Deny Insect Repellents`.

9.11 Publishing and starting the cube

For a new security model, or for changes to an existing security model to be applied, the cube must be redeployed to the content store. If the changes affect the structure of the existing cube, then the cube must be restarted for the changes to take effect.

If the changes only affect security, the administrator can run the **`Refresh security settings`** command on the running cube to only refresh security and avoid the cube restart. The same command can be used to reload the changed data in the security lookup table.

For more information, see the following sections:

- For instructions about publishing a cube from the IBM Cognos Cube Designer, see 5.8, “Deploying dynamic cubes for reporting and analysis” on page 154.
- For instructions about starting a dynamic cube, see 7.2.1, “Starting a cube” on page 205.

9.12 Applying security views

Cognos Dynamic Cubes security works by matching security views to individual users in the security name space. Follow these steps to assign users to security views:

1. Log on to IBM Cognos Administration as a user with the privileges to assign users to security views.
2. On the Configuration tab, click **Data source Connections**.
3. Click the **gosldw_sales** dynamic cube data source (Figure 9-87).

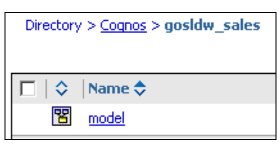


Figure 9-87 Click gosldw_sales

4. Click **model** to see the security views that are defined on the cube (Figure 9-88).

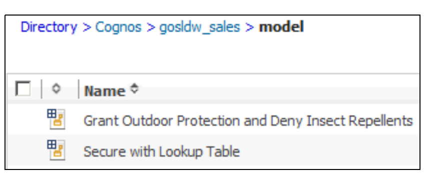


Figure 9-88 Click model

5. Users, group, and roles can be assigned to the security views. For this example, map both views to the same user. Click the **Set Properties** icon in the Actions column for `Grant Outdoor Protection and Deny Insect Repellents`. Figure 9-89 shows the Set properties for `Grant Outdoor Protection and Deny Insect Repellents`.

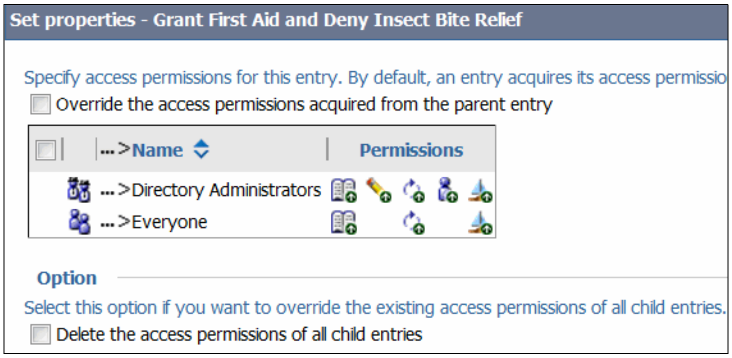


Figure 9-89 Set properties for Grant Outdoor Protection and Deny Insect Repellents

6. Select the **Override the access permissions acquired from the parent entry** option for the ability to add or remove access permissions (Figure 9-90).

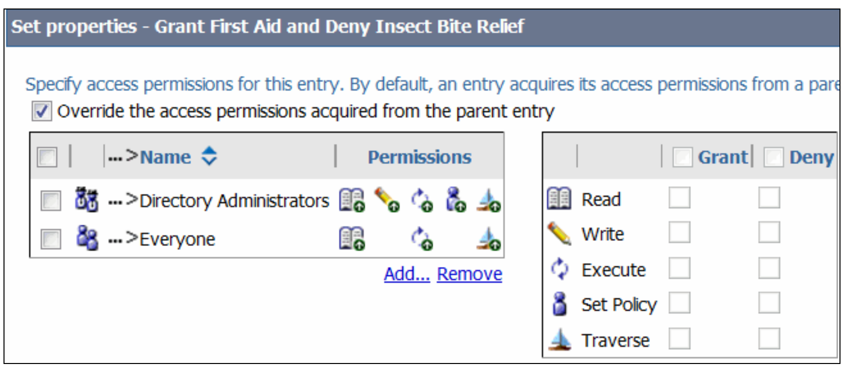


Figure 9-90 Select Override the access permissions acquired from the parent entry

7. Click **Add** to add user `Alan White` to `Grant Outdoor Protection and Deny Insect Repellents`.
8. Select **Directory Administrators** and **Everyone**, and then click **Remove** to remove them from `Grant Outdoor Protection and Deny Insect Repellents` (Figure 9-91).

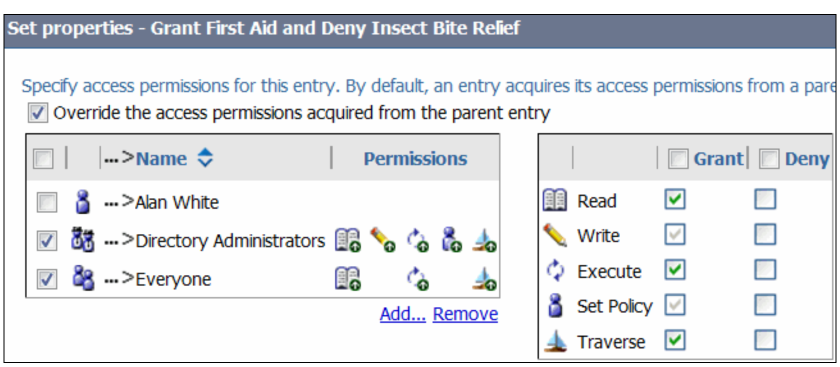


Figure 9-91 Removing Directory Administrators and Everyone

9. Select **Alan White** and then select the **Read** check box (under Grant) to give Alan White access to the security view. See Figure 9-92.

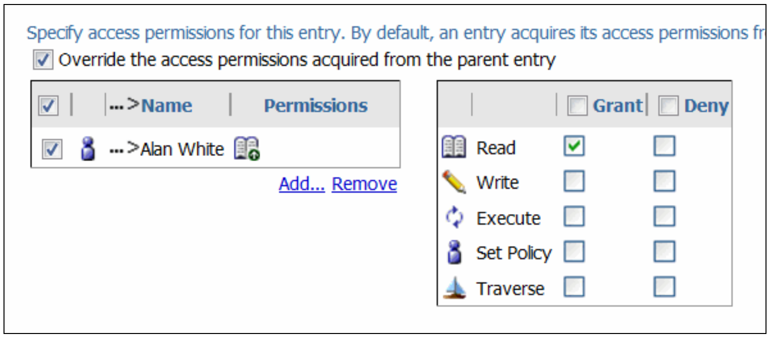


Figure 9-92 Giving Alan White access to security view

10. Click **OK**.

> **Note:** It is not necessary to repeat the same steps for the Secure with Lookup Table view because user access is controlled by the lookup table.

11. Start the `gosldw_sales` cube.
12. Select the **View recent messages** check box to ensure that there are no errors with the security views.

> **Note:** When IBM Cognos is installed, the group Everyone (which includes all users) is added as a member to the `System Administrators` group. Therefore, everyone is a system administrator and security is not activated. When you have defined security for your dynamic cube, remove `Everyone` from the `System Administrators` group to make sure that individual users are only allowed to see data as defined by security views.

9.13 Verifying the security model

If the package was queried by a user who is a member of the `System Administrators` group or if the cube had no security defined, the metadata tree shows all dimensions, members, measures, and attributes.

Figure 9-93 shows the `gosldw_sales` metadata tree.

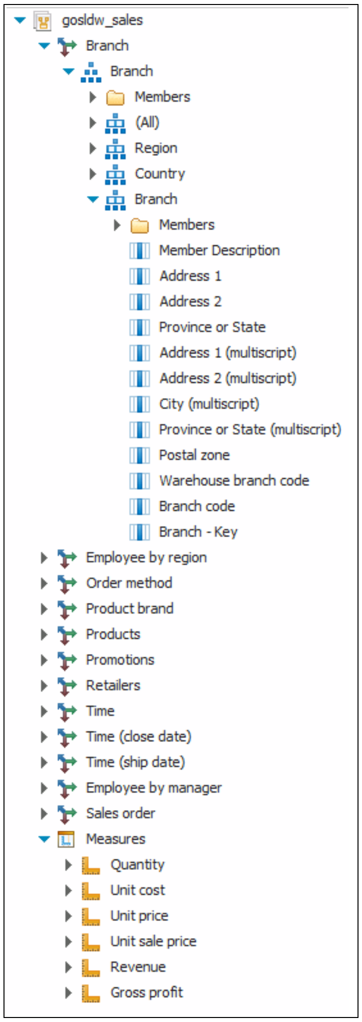


Figure 9-93 Metadata tree

A basic crosstab of `Unit Cost` and the `Products` hierarchy shows all the members under `Outdoor Protection` (Figure 9-94).

Unit cost				2011
All	Camping Equipment			90.81
	Mountaineering Equipment			
	Personal Accessories			56.01
	Outdoor Protection	Insect Repellents	BugShield Natural	1.86
			BugShield Spray	1.83
			BugShield Lotion Lite	1.88
			BugShield Lotion	2.33
			BugShield Extreme	2.42
		Sunscreen	Sun Blocker	1.95
			Sun Shelter Stick	1.96
			Sun Shelter 15	1.79
			Sun Shelter 30	1.85
			Sun Shield	2.76
		First Aid	Compact Relief Kit	9.10
			Deluxe Family Relief Kit	24.04
			Calamine Relief	2.83
			Aloe Relief	1.92
			Insect Bite Relief	2.76
	Golf Equipment			265.69

Figure 9-94 Members under Outdoor Protection

An administrator can see all the members under `Outdoor Protection`, which is the same view you get when you have no security defined.

If the package is queried by user `Alan White`, because he is assigned to `Grant Outdoor Protection and Deny Insect Repellents` and secured by lookup table view `Secure with Lookup Tables`, he sees a restricted view of the metadata tree with only the `Unit Cost`, `Unit Price`, and `Unit Sales Price` measures visible, the `Product` brand dimension hidden, and eight attributes from the `Branch` dimension hidden (Figure 9-95).

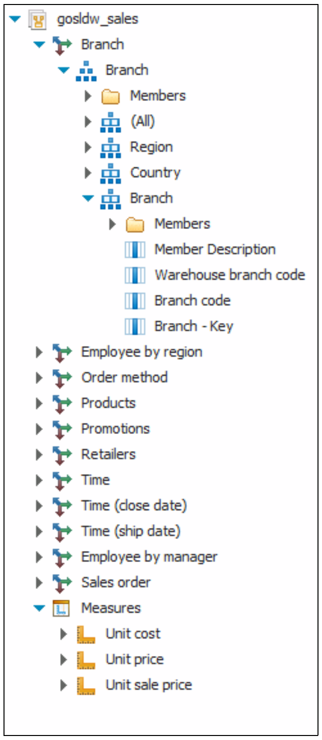


Figure 9-95 Restricted view of metadata tree

When the users query the same report as the administrator, they see a different result because of dimensional security (Figure 9-96).

Unit cost				2011
All	Outdoor Protection	Sunscreen	Sun Blocker	1.95
			Sun Shelter Stick	1.96
			Sun Shelter 15	1.79
			Sun Shelter 30	1.85
			Sun Shield	2.76
		First Aid	Compact Relief Kit	9.10
			Deluxe Family Relief Kit	24.04
			Calamine Relief	2.83
			Aloe Relief	1.92

Figure 9-96 Report run for restricted user

The members `Camping Equipment`, `Mountaineering Equipment`, `Personal Accessories`, and `Golf Equipment` are not visible to `Alan White` under the `All` level. Only `Outdoor Protection` is visible under the `All` member.

Also, under `Outdoor Protection`, `Insect Repellents` and its descendants, and `Insect Bite Relief` under `First Aid` are not visible to the user.

9.14 Transformer-style security

Many IBM Cognos customers are familiar with the different forms of member security in IBM Cognos PowerCubes. For that audience, it might be helpful to discuss how Cognos Dynamic Cubes member security can be used to achieve, or approximate, the security schemes in PowerCubes.

IBM Cognos Transformer and IBM Cognos PowerPlay users are used to the term *category* when describing elements of levels and hierarchies. For the purposes of this example, replace it with an equivalent term, *members*, that is used in Cognos Dynamic Cubes.

Start with a simple hierarchy, along with a measure value at each node that you will use as the basis for highlighting the various forms of member security. For simplicity, use the `Time` hierarchy of the `gosldw_sales` cube that you will reduce in size and only keep Q1 and Q2 of years 2011 and 2012 (see Figure 9-97).

	Quantity
All	21,820,846
2011	9,982,776
Q1	4,877,058
January	1,604,216
February	1,607,491
March	1,665,351
Q2	5,105,718
April	1,542,002
May	1,704,903
June	1,858,813
2012	11,838,070
Q1	6,258,427
January	2,047,167
February	2,227,565
March	1,983,695
Q2	5,579,643
April	1,917,087
May	1,800,216
June	1,862,340

Figure 9-97 Unsecured view of modified Time hierarchy

9.14.1 Cognos Transformer suppress option

The *suppress* option is used when a member is not required for reporting purposes. A suppressed member is replaced with a placeholder member with no associated measure values. The descendants and ancestors of the member are still visible in the hierarchy.

It is possible to approximate this form of security in Cognos Dynamic Cubes, except that the members being secured are still present, but without any measure values. This security option is achieved by granting access to the other members of the hierarchy, but not the member being suppressed. This techniqueimposes an implicit *grant* on the member in question. However, this result is not achievable with a leaf level member.

For example, if you wanted to suppress node 2012 Q1, create the security filter shown in Figure 9-98:

```
except (descendants ([All], 3, self beforewithmember), set ([Q1]))
```

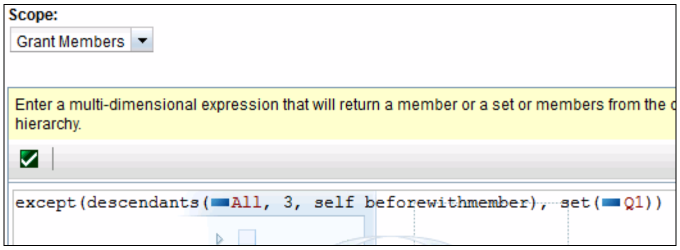


Figure 9-98 Suppress-style security filter

The effect of applying suppress-style security can be seen in Figure 9-99.

	Quantity
All	21,820,846
2011	9,982,776
Q1	4,877,058
January	1,604,216
February	1,607,491
March	1,665,351
Q2	5,105,718
April	1,542,002
May	1,704,903
June	1,858,813
2012	11,838,070
Q1	--
January	2,047,167
February	2,227,565
March	1,983,695
Q2	5,579,643
April	1,917,087
May	1,800,216
June	1,862,340

Figure 9-99 Hierarchy with suppress-style security applied

9.14.2 Cognos Transformer Apex option

The *Apex* option is used to hide the ancestors of a member, siblings, and all but its immediate descendants (children).

The Apex option can be approximated in Cognos Dynamic Cubes by using a combination of security filters. It is an approximation because of the way Cognos Dynamic Cubes always creates balanced, non-ragged hierarchies. If there is a single Apex member in a PowerCube, that member is the root of the hierarchy. In Cognos Dynamic Cubes, the ancestors of a member are still present, although their data values are secured.

As an example, assume that you want to apex on member 2011. Create the following security filters (see Figure 9-100):

```
[2011]
```

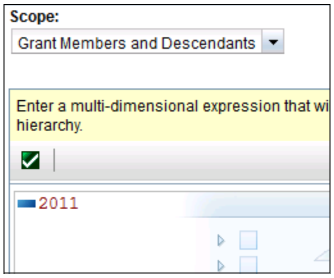


Figure 9-100 Apex-style security grant filter

Figure 9-101 shows this filter:

```
descendants ([2011], 2)
```

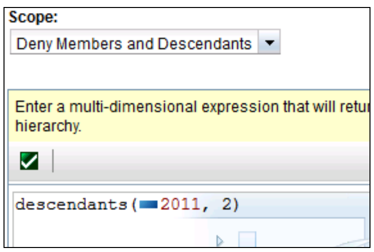
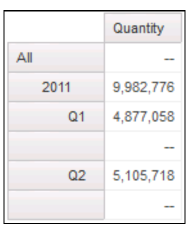


Figure 9-101 Apex-style security deny filter

The effect of applying apex-style security can be seen in Figure 9-102.



	Quantity
All	--
2011	9,982,776
Q1	4,877,058
	--
Q2	5,105,718
	--

Figure 9-102 Hierarchy with apex-style security applied

9.14.3 Cognos Transformer Cloak option

The *Cloak* option omits a member and its descendants from a hierarchy, but it includes the values of those members in the rollups of the hierarchy.

The Cloak option is identical to the *deny* logic of Cognos Dynamic Cubes, except that Cognos Dynamic Cubes always balances hierarchies, so filler members, if required, are added to a hierarchy to ensure that the hierarchy remains balanced after security has been applied.

As an example, assume that you want to cloak member `2012`. Create the following security filter (see Figure 9-103):

```
[2012]
```

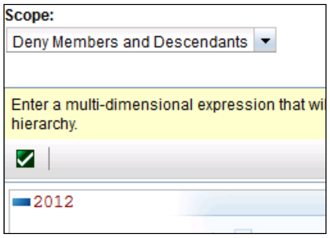


Figure 9-103 Cloak-style security filter

The effect of applying cloak-style security can be seen in Figure 9-104.

	Quantity
All	21,820,846
2011	9,982,776
Q1	4,877,058
January	1,604,216
February	1,607,491
March	1,665,351
Q2	5,105,718
April	1,542,002
May	1,704,903
June	1,858,813

Figure 9-104 Hierarchy with cloak-style security applied

9.14.4 Cognos Transformer Summarize option

The *Summarize* option secures the descendants of a member, but retains their values in the rollups of the hierarchy.

The Summarize option is identical to the *deny* logic of Cognos Dynamic Cubes, except that Cognos Dynamic Cubes always balances hierarchies, so filler members, if required, are added to a hierarchy to ensure that the hierarchy remains balanced after security has been applied.

As an example, if you wanted to summarize member 2011, create the following security filter (see Figure 9-105):

```
descendants ([2012], 2, before)
```

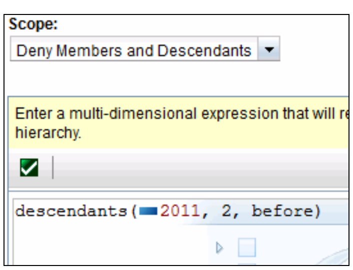


Figure 9-105 Summarize-style security filter

The effect of applying summarize-style security can be seen in Figure 9-106.

	Quantity
All	21,820,846
2011	9,982,776
	--
	--
2012	11,838,070
Q1	6,258,427
January	2,047,167
February	2,227,565
March	1,983,695
Q2	5,579,643
April	1,917,087
May	1,800,216
June	1,862,340

Figure 9-106 Hierarchy with summarize-style security applied

9.14.5 Cognos Transformer Exclude option

The *Exclude* option removes a member and its descendants from a hierarchy, excluding the value of these members from the rollup values and member counts of the ancestors of the member. This is sometimes referred to as *visual totals* because, even with security applied, totals match the rollup of the visible detail values.

There is no simple, direct manner in which this type of security can be applied within Cognos Dynamic Cubes, although it can be replicated, albeit with a few caveats:

- This security option introduces a duplicate of each hierarchy being secured.
- Performance can be impacted due to reaggregation of data that is required to compute the visual totals.

The solution relies on the introduction of a duplicate of the hierarchy that is being secured and applying a separate set of security filters to the hierarchythat produce the effect of excluded security. Unfortunately, it is not possible to hide this duplicate hierarchy, so it is best to change its name to indicate that it should not be used (for example, append the text (`do not use`) to the name).

To achieve exclude behavior, complete these steps:

1. Grant access to members (members, descendants, and ancestors) within the hierarchy to which a user is allowed access.
2. Create another hierarchy within the same dimension that is structured the same as the previous hierarchy and name it in a way that makes it clear it should not be used.
3. Grant access to the same set of members (but only member and descendants) as in the previous hierarchy.
4. Add a calculated member to the second hierarchy with the following expression:

   ```
   AGGREGATE(CURRENTMEASURE WITHIN SET MEMBERS( <level> ) )
   ```

   where <level> is the lowest level at which the secured members reside.
5. Grant access to the calculated member and make it the default member of the hierarchy.
6. Add the security filters to a security view in the cube, publish the cube, and if necessary, assign users to the security view.

This approach has these characteristics:

- Users are able to traverse the members of the hierarchy to which they have been granted access.
- Measure values are computed by the newly created calculated member that rolls values up from the members to which a user has access, thus producing visible totals.

As an example, if you exclude member `2012 Q1`, the hierarchy appears as seen in Figure 9-107 (note the updated total values).

	Quantity
All	15,562,419
2011	9,982,776
Q1	4,877,058
January	1,604,216
February	1,607,491
March	1,665,351
Q2	5,105,718
April	1,542,002
May	1,704,903
June	1,858,813
2012	5,579,643
Q2	5,579,643
April	1,917,087
May	1,800,216
June	1,862,340

Figure 9-107 Hierarchy with exclude-style security applied

Follow these steps to implement exclude-style security based on cube `gosldw_sales` (for brevity, some steps described earlier in the chapter are omitted):

1. In IBM Cognos Cube Designer add a duplicate hierarchy of the Time hierarchy, and rename it to `Time (do not use)` (see Figure 9-108).

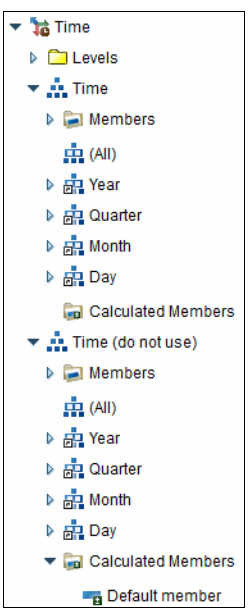


Figure 9-108 Original and duplicate hierarchies for exclude-style security

2. Grant access to the members in the original hierarchy (members, descendants, and ancestors). See Figure 9-109.

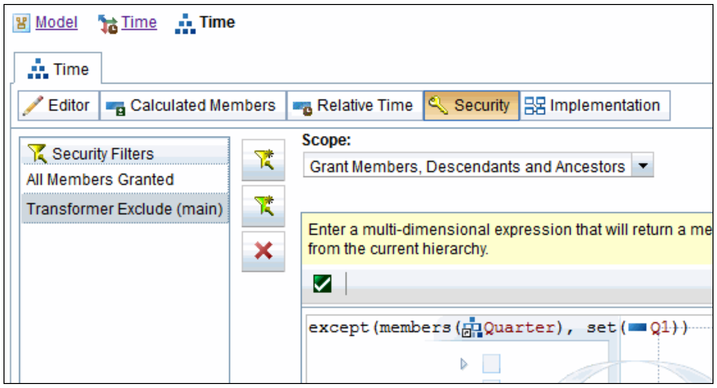


Figure 9-109 Granting access to members in original hierarchy

3. Create a calculated member called `Default member` in the duplicate hierarchy (see Figure 9-110).

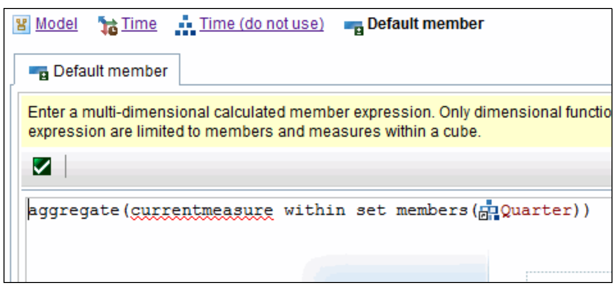


Figure 9-110 Calculated member definition

4. Make the calculated member the default member in that hierarchy (see Figure 9-111).

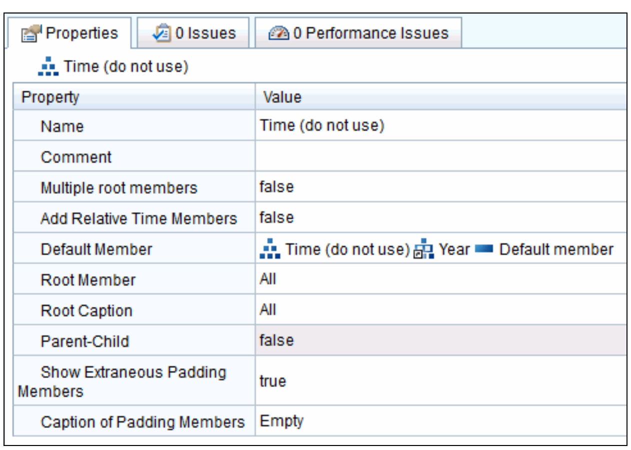


Figure 9-111 Setting up the default member in a hierarchy

5. Grant access to the members in the duplicate hierarchy and new calculated member (members and descendants) as shown in Figure 9-112.

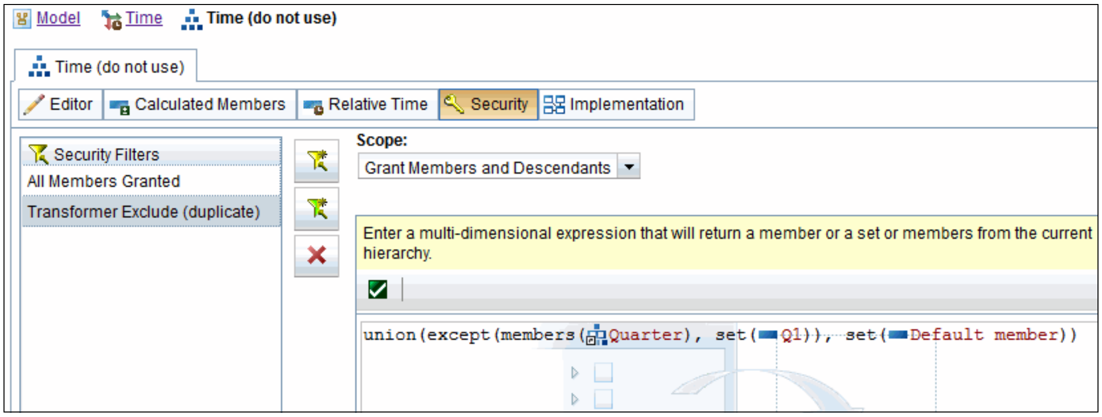


Figure 9-112 Granting access to members in the duplicate hierarchy

6. Complete security setup by adding the security filters that you have created to a view and securing that view in the IBM Cognos Business Intelligence (BI) portal.

To test this security setup create a crosstab in IBM Cognos Analysis Studio to include `Product Line`, `Years`, and `Quantity`.

Without member security enforced, the reports output is as follows (Figure 9-113).

Quantity	2011	2012	All
Camping Equipment	3,022,894	3,638,097	**6,660,991**
Mountaineering Equipment		1,318,869	**1,318,869**
Personal Accessories	3,642,690	4,170,232	**7,812,922**
Outdoor Protection	2,749,696	2,023,377	**4,773,073**
Golf Equipment	567,496	687,495	**1,254,991**
All	**9,982,776**	**11,838,070**	**21,820,846**

Figure 9-113 Unsecured report output

With the exclude-style member security enforced, the member `2012 Q1` is hidden, and the totals are adjusted (see Figure 9-114).

Quantity	2011	2012	All
Camping Equipment	3,022,894	1,591,861	**4,614,755**
Mountaineering Equipment		638,443	**638,443**
Personal Accessories	3,642,690	2,045,164	**5,687,854**
Outdoor Protection	2,749,696	970,173	**3,719,869**
Golf Equipment	567,496	334,002	**901,498**
All	**9,982,776**	**5,579,643**	**15,562,419**

Figure 9-114 Report output with exclude-style security

Note: When implementing exclude-style security, make sure that you reimplement the calculated member and reassign it as the default member because calculated members from base cubes are not carried forward into a virtual cube.

10

Securing the IBM Cognos BI environment

This chapter describes roles and capabilities that are required to perform different tasks related to dynamic cubes, and steps for securing development and production environments.

This chapter contains the following sections:

- Roles and capabilities required to manage dynamic cubes
- Securing cube data
- Securing the development environment
- Securing the production environment

10.1 Roles and capabilities required to manage dynamic cubes

IBM Cognos administrators can use groups, roles, and capabilities to define the access permissions that are required for modeling, configuring, managing, and optimizing dynamic cubes. Table 10-1 describes the user roles that are associated with managing dynamic cubes and the typical tasks that these roles perform. Administrators must ensure that these roles are created in the Cognos namespace in IBM Cognos Administration.

Table 10-1 Roles and tasks associated with managing dynamic cubes

Roles	Tasks
Dynamic cubes modelers	Model and publish dynamic cubes, assign cubes to dispatchers, and start cubes. If needed, this role can be further broken down to limit the capabilities of individual users (as outlined in Table 10-2)
Dynamic cubes security administrators	Assign users, groups, and roles to dynamic cubes security views.
Dynamic cubes configuration administrators	Assign cubes to server groups and dispatchers, and configure the query service and individual cubes.
Dynamic cubes managers	Perform interactive administrative commands on cubes, and create and schedule query service administrative tasks.
Dynamic cubes optimizers	Save in-memory aggregate recommendations from Aggregate Advisor to the content store. To perform other Aggregate Advisor tasks, administrators only need access to IBM Cognos Dynamic Query Analyzer and an account in IBM Cognos Business Intelligence.
Dynamic cubes administrators	Perform all of the operations described in this table. This role is assigned to all the roles described in Table 10-2 and Table 10-3 on page 329.

Each role requires an associated IBM Cognos BI capability to perform specific tasks on dynamic cubes. Grant correct permissions to appropriate roles to grant access to a capability. For example, dynamic cube modelers who create models need execute and traverse permissions for the Import relational metadata capability.

Table 10-2 lists the roles and the capabilities that these roles require for managing dynamic cubes. The capabilities column in the table refers to the corresponding capability in IBM Cognos Administration graphical user interface (GUI).

Table 10-2 Roles and corresponding capabilities

Roles	Capabilities	Required access permissions
Dynamic cube modelers (creating new models)	Import relational metadata	Execute, traverse

Roles	Capabilities	Required access permissions
Dynamic cube modelers (starting cubes)	Administration	Execute, traverse
	Administration → Configure and manage the system	Execute, traverse
Dynamic cubes modelers (generating cubes or dimensions with data samples)	Specification Execution	Execute, traverse
Dynamic cubes security administrators	Administration	Execute, traverse
	Administration → Data Source Connections	Execute, traverse
Dynamic cubes configuration administrators	Administration	Execute, traverse
	Administration → Administration tasks	Execute, traverse
	Administration → Configure and manage the system	Execute, traverse
	Administration → Data Source Connections	Execute, traverse
Dynamic cubes managers	Administration	Execute, traverse
	Administration → Administration tasks	Execute, traverse
	Administration → Configure and manage the system	Execute, traverse
	Administration → Query Service Administration	Execute, traverse
	Administration → Run activities and schedules	Execute, traverse
	Scheduling	Execute, traverse
	Cognos Viewer	Execute, traverse
	Cognos Viewer → Run With Options	Execute, traverse
Dynamic cubes optimizers (saving in-memory recommendations)	Administration → Configure and manage the system	Execute, traverse

Tip: Capabilities are also referred to as *secured functions* and *features*. This distinction is helpful when dealing with two-level capabilities, such as the Administration capability. In this case, capabilities such as Configure and manage the system, Data Source Connections, and Query Service Administration are secured features of the Administration capability, which itself is a secured function.

For more information about the IBM Cognos BI capabilities, see the security sections in the *IBM Cognos Business Intelligence Administration and Security Guide:*

http://public.dhe.ibm.com/software/data/cognos/documentation/docs/en/10.2.2/ug_cra.pdf

Example: Granting capabilities to dynamic cubes security administrator

This example shows how to grant all required capabilities to dynamic cubes security administrators. Dynamic Cubes Security Administrators is created in the Cognos namespace before this procedure.

Follow these steps to grant required capabilities to Dynamic Cubes Security Administrators:

1. In IBM Cognos Administration, select the Security tab.
2. Click **Capabilities** as shown in Figure 10-1.

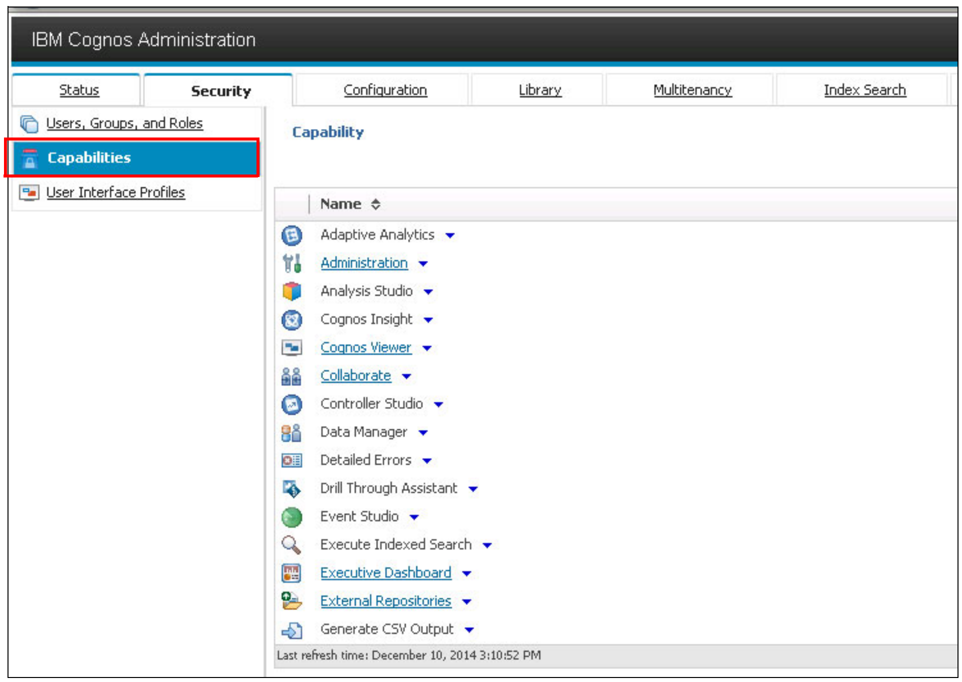


Figure 10-1 Capabilities

3. From the **Administration** list, select **Set Properties** as shown in Figure 10-2.

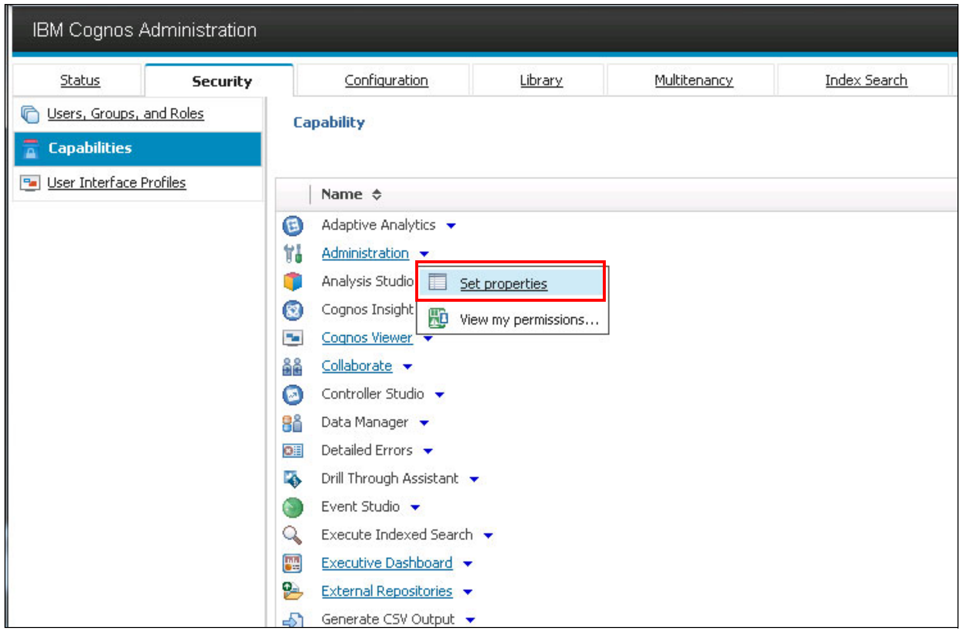


Figure 10-2 Set Properties for Administration

4. In the Set properties - Administration window, select the Permissions tab.
5. Click **Add** as shown in Figure 10-3.

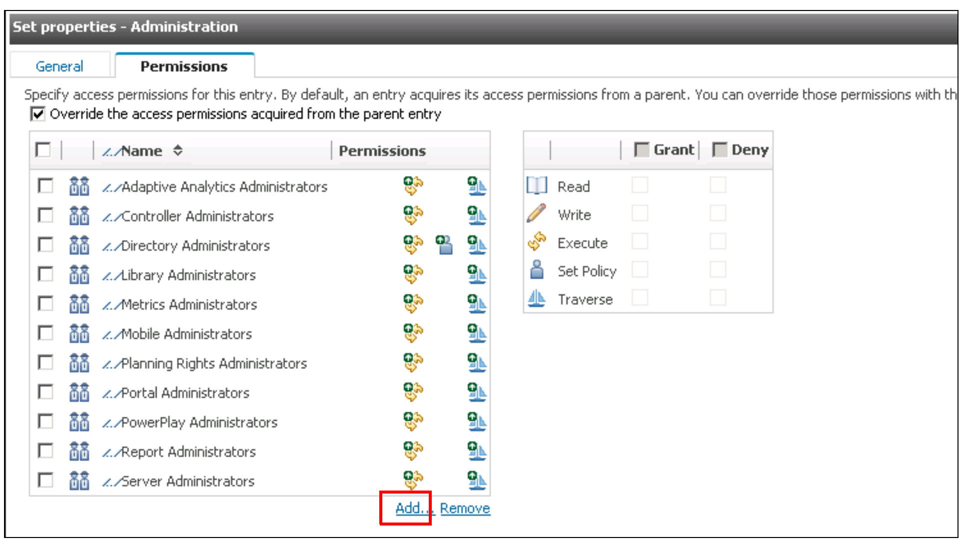


Figure 10-3 Add Permissions

6. Click the folder **Cognos** to find available entries to add as shown in Figure 10-4.

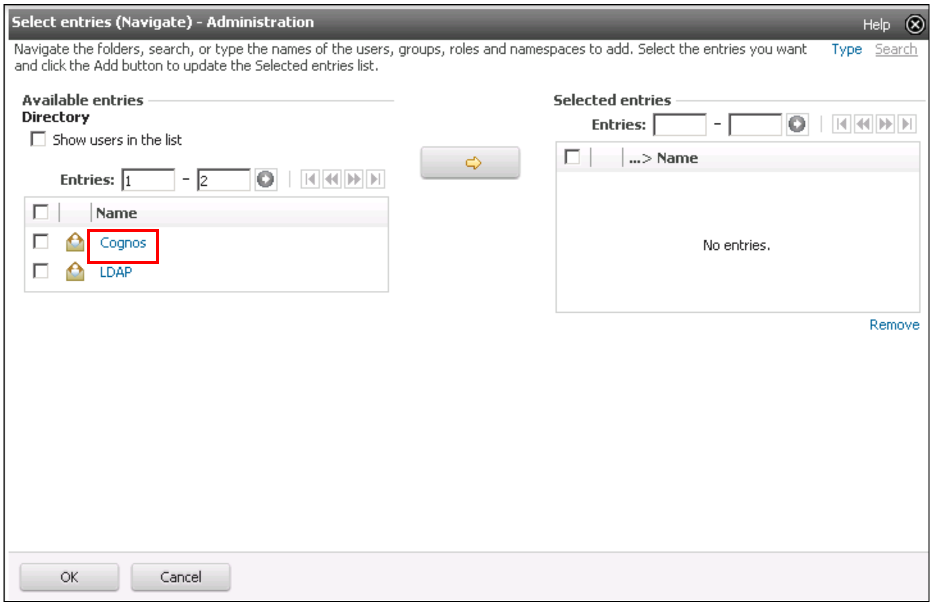


Figure 10-4 Listing the available entries

7. Select the role **Dynamic Cubes Security Administrators** and add it to **Selected entries** by clicking the yellow arrow button and then click **OK** as shown in Figure 10-5.

i

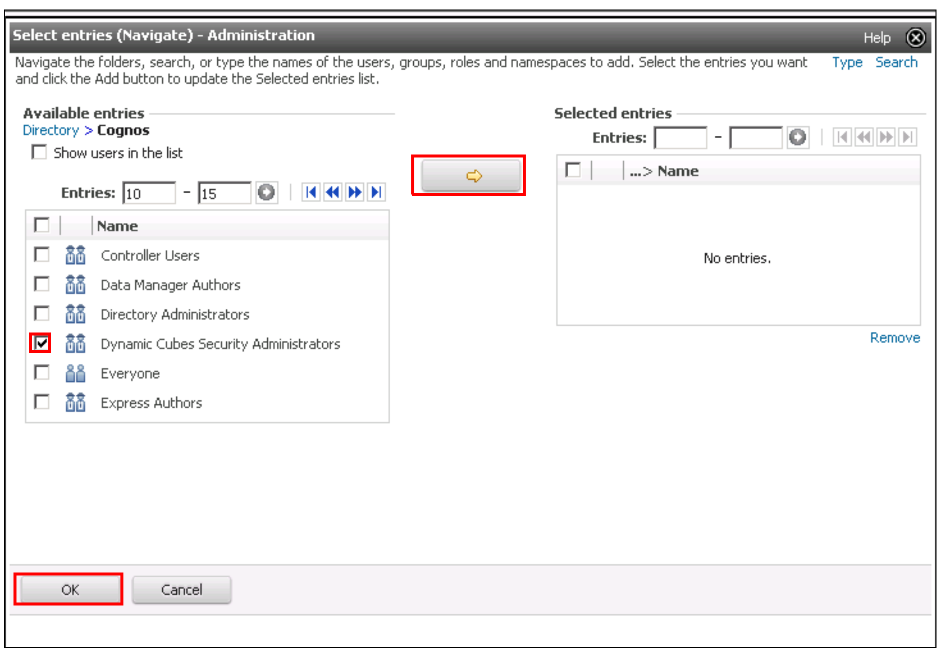


Figure 10-5 Select entries

8. Grant **Execute** and **Traverse** permissions to **Dynamic Cubes Security Administrators** as shown in Figure 10-6, and then click **OK**.

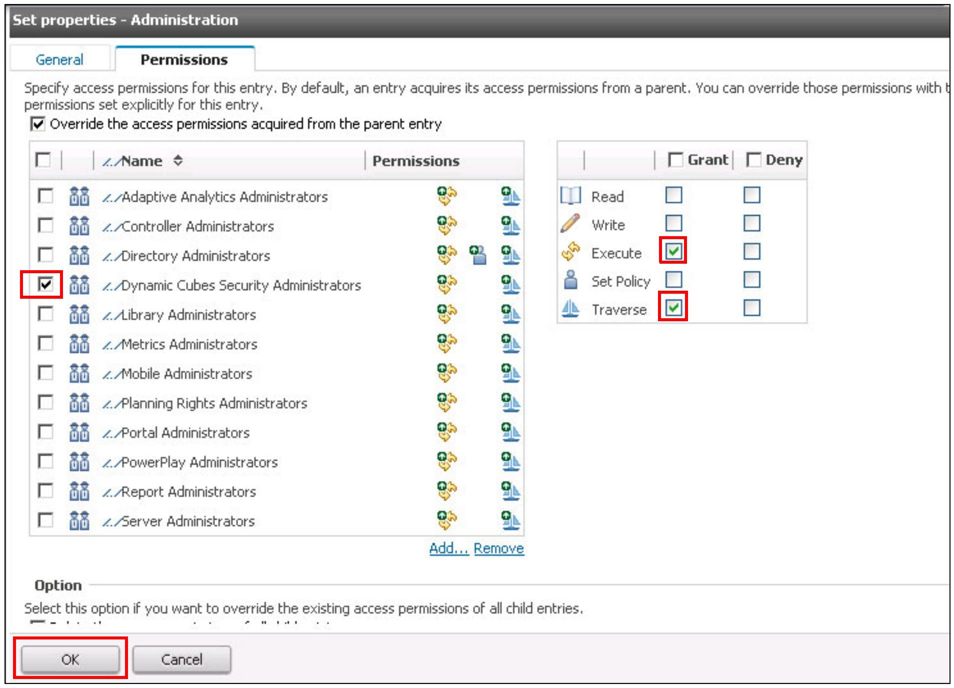


Figure 10-6 Access permissions

9. Administration is a two-level capability. Grant permissions to secured features the same way that you grant permissions to secured functions. Click **Administration** as shown in Figure 10-7.

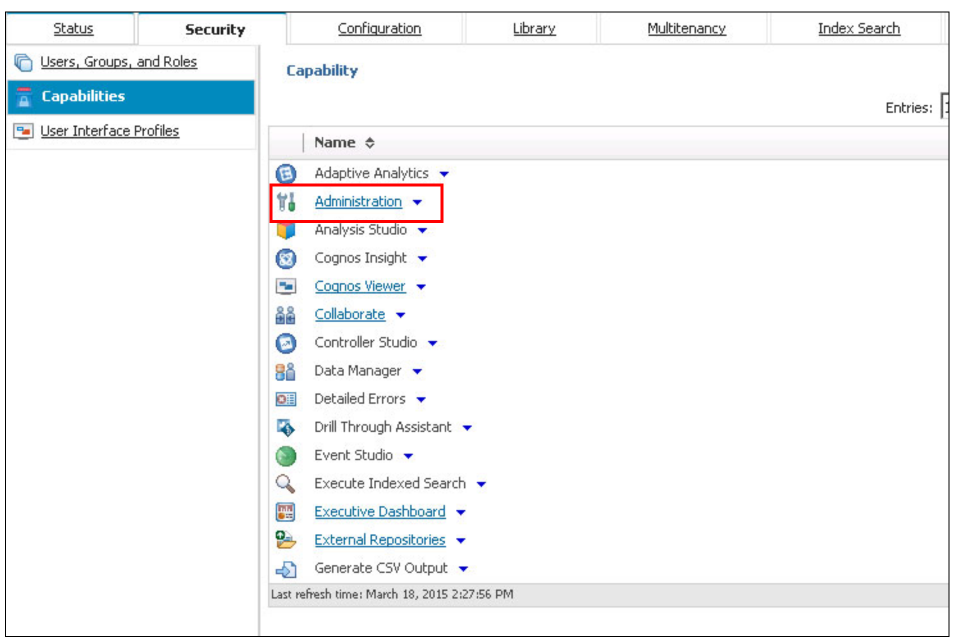


Figure 10-7 Two-level capability - Administration

10. You can see all the secured features under **Administration** as shown in Figure 10-8.

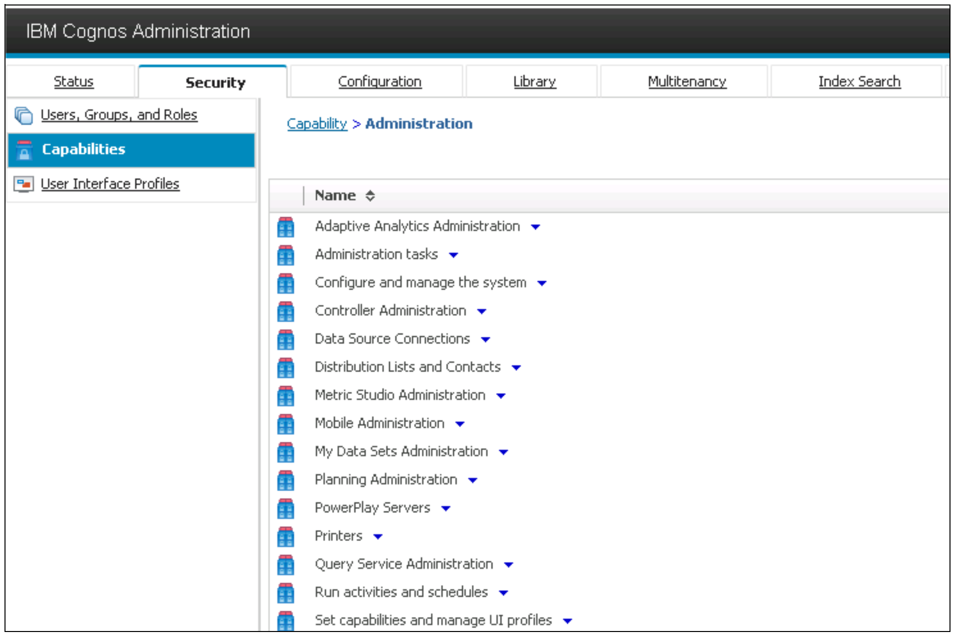


Figure 10-8 Secured features for Administration

11. From the **Data Source Connections** menu, select **Set Properties**.
12. In the Set properties - Data Source Connections windows, select the Permissions tab.
13. Select the role **Dynamic Cubes Security Administrators**. Add **Dynamic Cubes Security Administrators** to **Selected entries** by clicking the yellow arrow button**,** and then **OK**.

14. Grant permissions **Execute** and **Traverse** to **Dynamic Cubes Security Administrators** as shown in Figure 10-9. Click **OK**.

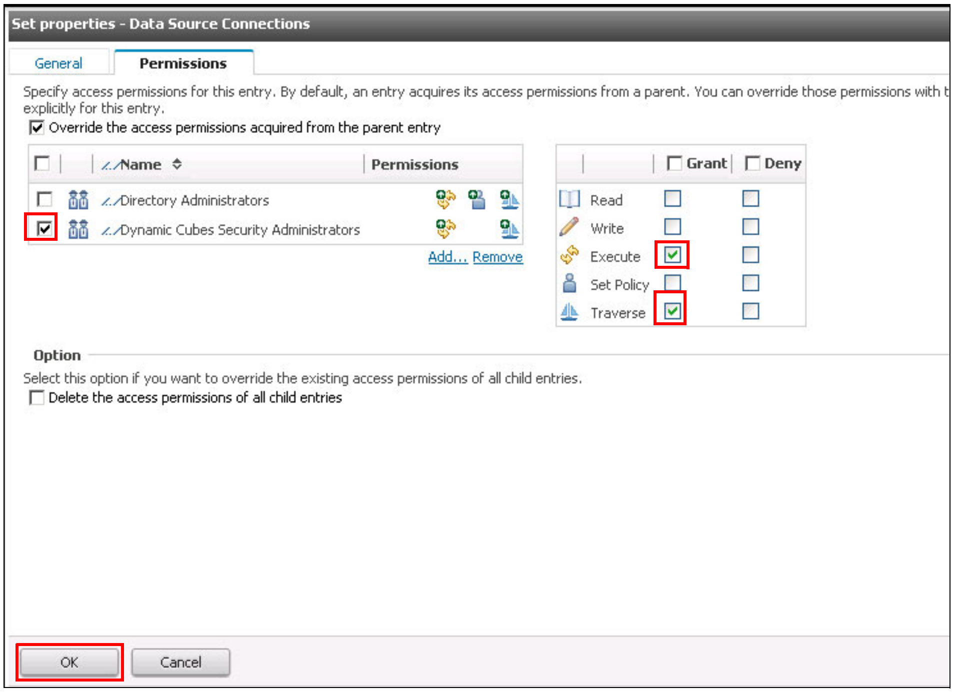


Figure 10-9 Grant permissions to Dynamic Cubes Security Administrators

In addition to capabilities, dynamic cube administrators need the correct combination of access permissions for the content store objects. Table 10-3 specifies the objects and the permissions that are required for specific roles. The content store object names in the Content store object column in the table match the Cognos Administration GUI.

Table 10-3 Content store objects permissions for roles

Role	Content store object	Required access permissions
Dynamic cubes modelers (publishing cube to server)	Configuration, Data Source Connections, Directory, Cognos	Read, write, execute, traverse
Dynamic cubes modelers (publishing a package)	My Folders, Public Folders	Read, write, traverse
Dynamic cubes modelers (assigning cube to a dispatcher)	Query Service (on one or more dispatchers) Configuration, Dispatchers, and Services	Read, write, execute, traverse
Dynamic cubes security administrators	Configuration, Data Source Connections, Directory, Cognos	Read, write, execute, traverse, set policy
Dynamic cubes configuration administrators	Configuration, Query Service (on one or more dispatchers), Dispatchers, and Services	Read, write, execute, traverse

Role	Content store object	Required access permissions
Dynamic cubes managers	Query Service (on all dispatchers on which cubes are managed) Configuration, Content Administration	Read, write, execute, traverse
Dynamic cubes optimizers (saving in-memory recommendations)	Configuration, Data Source Connections, Directory, Cognos	Read, write, execute, traverse

Example: Granting content store permissions to security administrators

This example shows how to grant the required permissions for content store objects to Dynamic Cubes Security Administrators.

Perform the following steps:

1. In IBM Cognos Administration, select the Configuration tab.
2. Click **Data Source Connections**.
3. In the Configuration tab, click the **Set properties - Cognos** icon as shown in Figure 10-10.

Figure 10-10 Click icon Set properties - Cognos

4. Select the Permissions tab.
5. Click **Add** as shown in Figure 10-11.

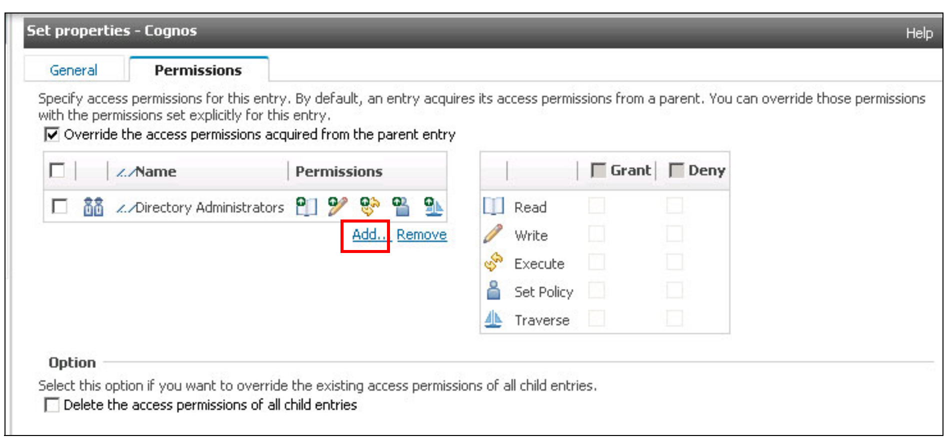


Figure 10-11 Add permissions

6. Click the folder **Cognos** to find the available entries.
7. Select the role **Dynamic Cubes Security Administrators**. Add this role to **Selected entries** by clicking the yellow arrow button and then clicking **OK**.

8. Grant all permissions to Dynamic Cubes Security Administrators as shown in Figure 10-12, and then click **OK**.

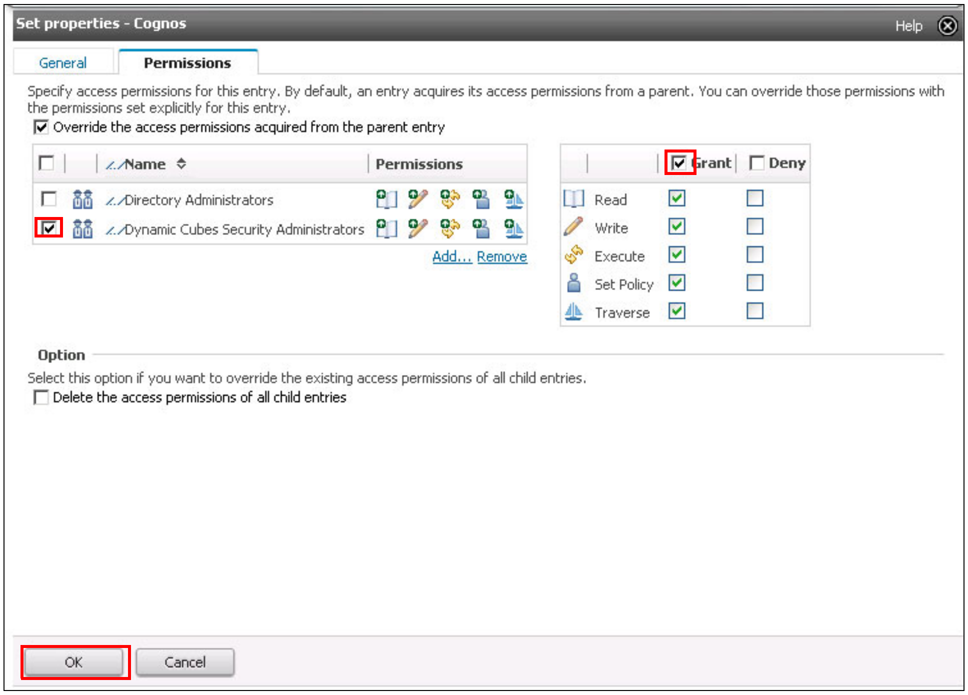


Figure 10-12 Grant Permissions

10.2 Securing cube data

Source dynamic cubes in a secured environment are assigned an *access account*. An access account is a user account that dynamic cube functionality impersonates to gain access to stored database signon information for loading cube data and metadata, and to run cube trigger jobs.

It is a common practice to restrict this account to access to the data source that it requires to load the cube data. If the cube has a trigger defined access account, it should also be granted the `Run activities and schedules` capability.

10.2.1 Setting up the access account

The access account must be assigned before the cube can be used in the secured environment. Usually this assignment happens automatically when the cube is published from IBM Cognos Cube Designer. However often it is necessary to set or change the access account in IBM Cognos Administration.

Follow these steps to set the cube access account:

1. Go to **IBM Cognos Administration** → **Status** → **Dynamic Cubes**.
2. Click the drop-down menu icon next to the cube name and select **Set properties** (see Figure 10-13).

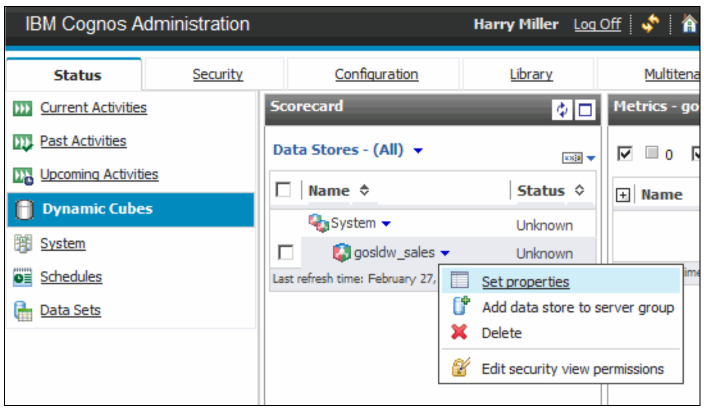


Figure 10-13 Set properties menu

3. From the Set properties page, click **Select the access account** (see Figure 10-14).

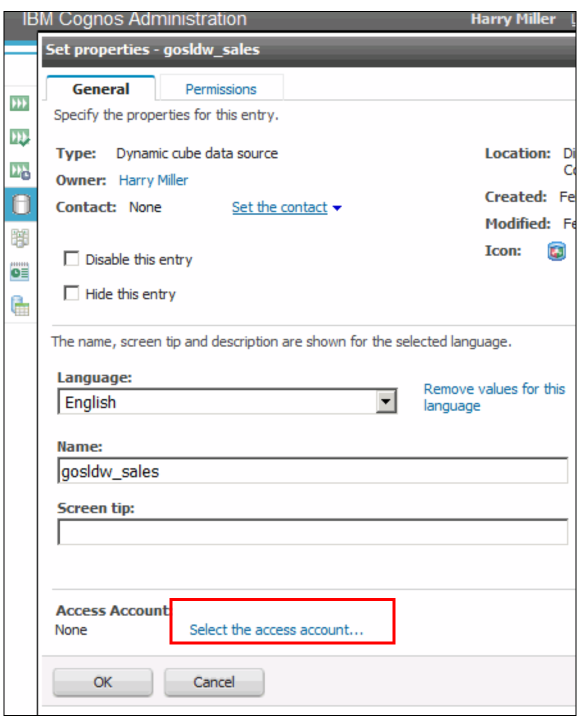


Figure 10-14 Selecting access account

4. Select the access account from the list of user accounts and click **OK** (see Figure 10-15). Currently, you can only use real user accounts as access accounts, and not groups or roles. Click **OK** again to close the Set properties window.

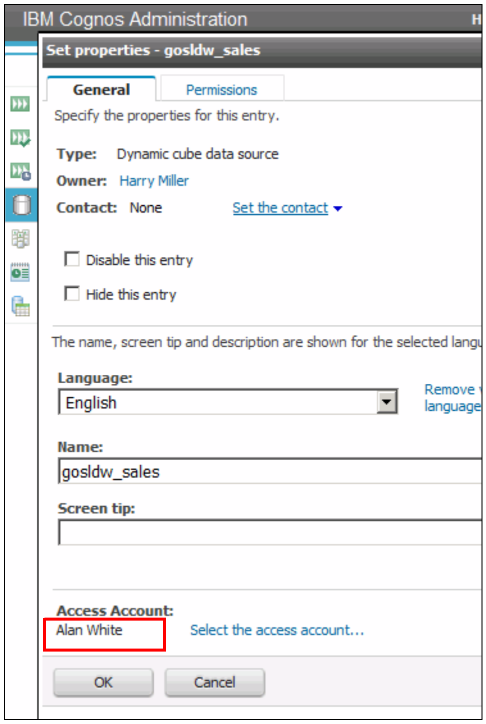


Figure 10-15 Access account selected

5. Now the user account (`awhite`) is set up as the access account. Before it can be used, the credential for it must be saved.

   Log in as the user who holds the access account, and click **My Preferences** → **Personal**.

6. The Credentials section shows the following text:

   `Specify the users, groups or roles that can use the credentials to run activities.`

   – If you do *not* have a valid credential, then after the text, a hyperlink shows `First, you must create the credentials.`

   – If you see the following text, then you already have a valid credential:

     `You can also renew the credentials. Renew the credentials.`

Figure 10-16 shows an example.

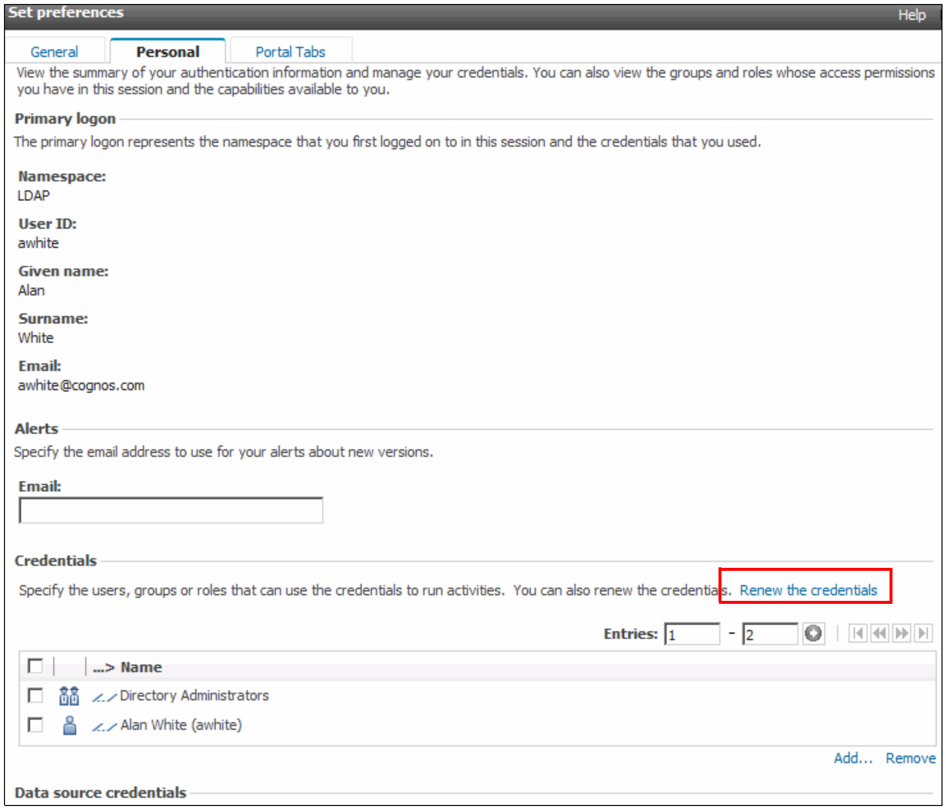


Figure 10-16 Saving account credentials

7. Click the hyperlink to refresh the page and display your credentials. Click **OK** to close the Properties window.

The cube can now be started and will use the access account to load data from the relational data source.

10.2.2 Securing data source access to exclude the system administrator

An IBM Cognos system administrator has access to all data within a dynamic cube. However, a dynamic cube does not necessarily expose all of the data accessible through the relational data source connection.

Some customer scenarios require you to restrict access to sensitive data in the relational data source in such a way that it is *only* accessible to the access account. This means that even the system administrator must not be allowed to access it.

The system administrator controls object access permissions in the content store, so if the data source signon is stored in the content store, the administrator can always acquire access to it. To restrict this access, you need to store relational database access credentials *in the access account* instead of in the data source connection object.

To secure access to the data source excluding the system administrator, complete the following steps:

1. Set up the cube account as described in 10.2.1, “Setting up the access account” on page 332. Select user `awhite` for the cube access account.
2. In the IBM Cognos BI portal, log in as the system administrator.
3. On the Security tab in IBM Cognos Administration, click **Capabilities**.
4. Give user `awhite` the capability **Manage own data source signons** (the process is described in 10.1, “Roles and capabilities required to manage dynamic cubes” on page 322).
5. Start the cube `gosldw_sales` as described in 7.2.1, “Starting a cube” on page 205.
6. On the Configuration tab, click **Data Source Connections**.
7. Click the relational data source name (**great_outdoors_warehouse**), and then click the name of the data source connection.
8. Select the check box next to the name of the stored data source signon and click the **Delete** icon *to* delete the signon (see Figure 10-17).

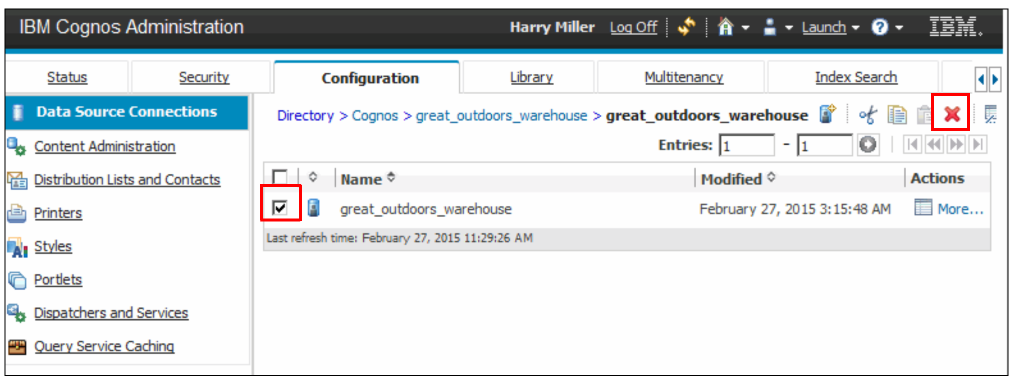


Figure 10-17 Deleting the data source signon

9. Log out and log back in with the access account user `awhite`.
10. Open **Query Studio** in package `gosldw_sales`. Expand **Measures** and select **Quantity**. Click **Insert**.

11.You are prompted for the database signon credentials. Enter the user ID and password, and select **Remember my user ID and password when connecting to this data source**. Click **OK** to save (see Figure 10-18).

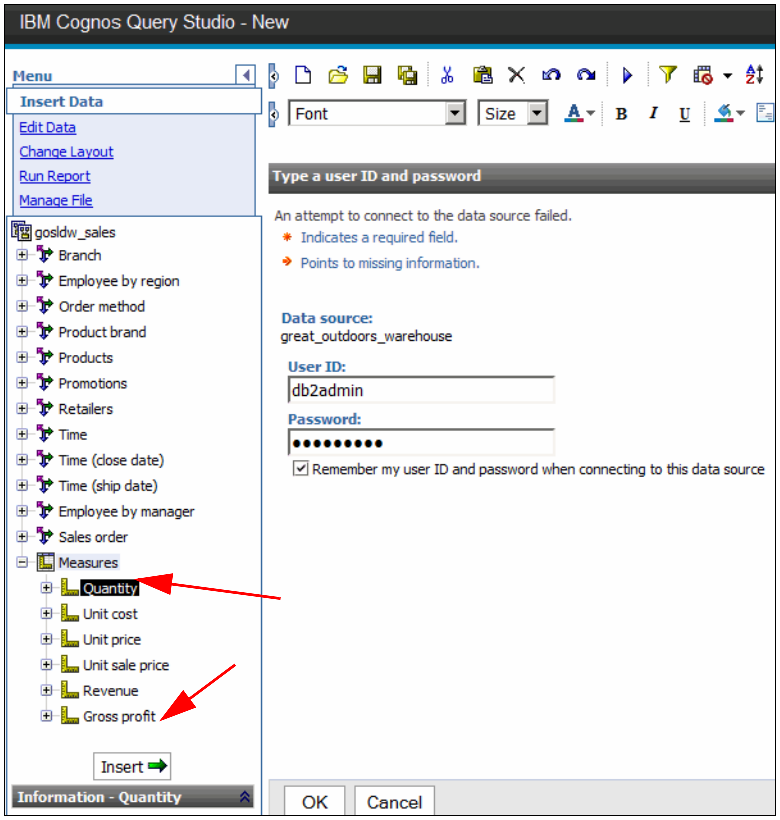


Figure 10-18 Entering data source credentials

The data source credentials are now saved in the user account. They can be viewed and deleted in the user preferences page (by clicking **My Preferences** → **Personal**) (see Figure 10-19).

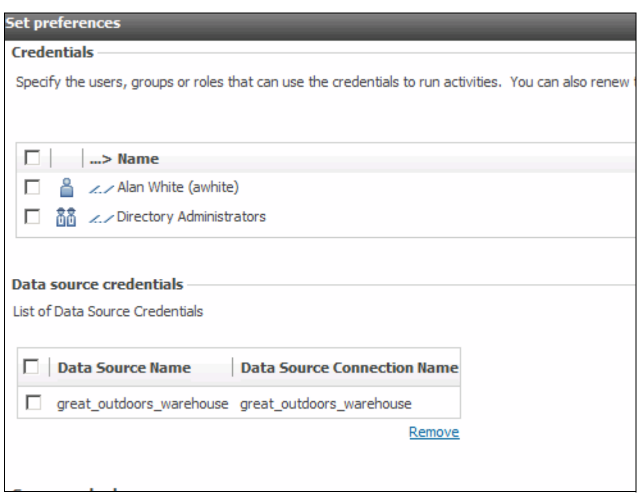


Figure 10-19 Data source credentials can be managed in the user account preferences

If an external namespace is used for authentication to an external data source, there is no signon that the system administrator can use to access the relational data source. In this case, the trusted dynamic query mode server impersonates the user of the data access account to sign on to the relational database.

10.3 Securing the development environment

Developers of applications that are based on dynamic cubes need a specific set of access permissions to be able to model, deploy, and manage a dynamic cube without requiring help from the IBM Cognos system administrator.

To address this need of application developers, create the role of a Dynamic Cube Developer, in addition to the standard dynamic cube roles that are documented in 10.1, “Roles and capabilities required to manage dynamic cubes” on page 322.

As a system administrator who creates the Dynamic Cubes Developer role, you must carefully consider which types of access permissions and capabilities you will grant to this role to enable the developers to do their job without compromising the security of the system.

Table 10-4 specifies the tasks that a member of the Dynamic Cubes Developer role performs, and the restrictions that the system administrator should consider to impose when granting access permissions to developers in the context of these tasks.

Table 10-4 Restrictions when granting access permissions to the Dynamic Cubes Developer role

Task	Restrictions
Import relational metadata into Cognos Cube Designer.	Grant access only to a specific set of relational data sources that the developers need to import metadata.
Publish a cube to the content store.	Grant permissions to either create dynamic cube data sources or update existing dynamic cube data sources.
Create a package in the content store.	Grant access only to specific folders where the developers can create packages.
Assign a user account to the data access account of a cube.	Grant access only to those accounts that can be assigned to access the relational data source. Developers should not have the permission to edit data source connections or signons.
Assign a cube to a dispatcher and modify the cube configuration.	Restrict the dispatchers to which the developers can assign a cube and modify the cube configuration.
Perform administration commands on a cube.	Restrict the dispatchers on which the developers can manage a cube. Do not allow the developers to perform any other administration commands, such as stopping or starting the query service.
Create and execute administration tasks for dynamic cubes.	The ability to create and edit administrative tasks cannot be restricted to dynamic cubes only. Allowing a user to create administration tasks for dynamic cubes allows them to create and execute these types of tasks for the entire system.

The following steps are performed by the Cognos system administrator to secure the development environment:

1. Create the Dynamic Cubes Developer role in the `Cognos` namespace in IBM Cognos Administration.
2. Specify access permissions for the Dynamic Cubes Developer role.

   The following list specifies which access permissions are required for each task that the members of the Dynamic Cubes Developer role perform.

   > **Note:** If a user owns a specific cube, denying access permissions on this cube has no effect for that user.

   – Import relational metadata into Cube Designer:
     - Deny all access permissions for relational data sources that cannot be imported. Doing so also disallows the use of the data source in Framework Manager.
     - Grant read and execute permissions, and deny write permissions for the relational data sources that can be imported.

   – Publish a cube to the content store:
     - Grant read, write, execute, and traverse permissions for existing dynamic cube data sources that the user can update.
     - Grant read, execute, and traverse permissions, and deny write permissions for existing dynamic cube data sources that the user cannot update.

   – Create a package in the content store:
     - Deny all permissions for folders to which the users are denied access. The users cannot see these folders in Cognos Administration, but they might see them in Cognos Cube Designer. The users cannot publish packages to these folders.
     - Grant read, execute, and traverse permissions, and deny write permissions for folders that the users can view but cannot update.
     - Grant read, write, execute, and traverse permissions for folders that the users can update.
     - Grant read, execute, and traverse permissions, and deny write permissions for packages that the users can view but cannot update.
     - Grant read, write, execute, and traverse permissions for packages that the users can update.
     - Grant read, write, execute, and traverse permissions for the dynamic cube data sources.

   – Assign a user account to the data access account of a cube:
     - Deny all permissions for the connection and signon objects of the relational data source that the dynamic cube is based on.
     - Provide a limited number of signons for the relational data source connection to control the data access because the developer can assign any user account to a dynamic cube. Grant read, execute, and traverse permissions on selected connection and signon objects.

– Assign a cube to a dispatcher and edit the cube configuration:
  - Grant read, write, execute, and traverse permissions for the query service on the dispatchers that the users can access.
  - Grant execute and traverse permissions, and deny read and write permissions for the query service on the dispatchers that the users cannot access.
  - Deny all permissions for the dynamic cube data sources that cannot be configured.

– Perform administrative commands on a cube:
  - Grant read, write, execute, and traverse permissions for query service on the dispatchers that the users can access.

– Create and edit only dynamic cubes administrative tasks:

  If a developer who can manage dynamic cubes has no need to create administration tasks for dynamic cubes, you might choose not to assign the following capabilities to the developers:
  - **Administration → Run activities and schedules**
  - **Scheduling**
  - **Cognos Viewer**
  - **Cognos Viewer → Run With Options**

3. Add users to the `Dynamic Cubes Developer` role.
4. Add the `Dynamic Cubes Developer` role to one or more of the dynamic cube administrative roles that are documented in 10.1, “Roles and capabilities required to manage dynamic cubes” on page 322.

If a dynamic cube developer does not need to create dynamic cubes administration tasks, then the capabilities in Table 10-5 are not required. An administrator might choose not to assign these capabilities because allowing a developer to create administration tasks allows them to create and run all types of administration tasks on the system.

Table 10-5 Capabilities that might not be required by Dynamic Cube Developer

Role	Capabilities not required by dynamic cube developer
Dynamic cube managers (Assuming no need to create administration tasks)	**Administration → Run activities and schedules**
	Scheduling
	Cognos Viewer
	Cognos Viewer → Run With Options

10.4 Securing the production environment

Securing the production environment is similar to securing the development environment: You are restricting users access to different features of dynamic cubes to make sure that only a selected group of users has access to just the features and capabilities they need.

The following are some important differences for securing the production environment:

- No modeling capability. Modeling is usually done in a development environment.
- No cube publishing capability. Cube publishing is also reserved for development environments.

- Tightened system administration capabilities. A limited number of system administrators need to share administration duties, and access to administration functions needs to be carefully controlled.
- Limited number of users have optimization capabilities. Optimization is primarily performed as part of the development process, although some of it is done in production to fine-tune the cube performance.
- Administration task capability is reintroduced. Although it is not usually applicable to the development environment, ability to create and schedule administration tasks becomes important in production.

In large organizations and organizations with strict security environments, it is common to define different administrator roles that have different capabilities. This approach helps to split duties between different people and ensure that access to different system capabilities is compartmentalized for better security of data. Administrator roles defined in Table 10-1 on page 322 (except for Dynamic cube modelers) are commonly used in dynamic cubes production environments.

In addition to cube administrators managing the cube, different classes of users use cube data and metadata. Security for those users is managed by assigning them to security views defined for the cube and by creating and maintaining security lookup tables. Assigning users to security views is described in 9.12, “Applying security views ” on page 307. Securing cubes through security lookup tables is described in 9.9, “Security lookup tables” on page 288.

11

Optimization and performance tuning

This chapter describes the use of the IBM Cognos Dynamic Cubes Aggregate Advisor to generate aggregate recommendations to optimize query performance, and the techniques and scenarios for performance tuning.

This chapter contains the following sections:

- Performance implications of OLAP-style reports
- Aggregate awareness in Cognos Dynamic Cubes
- Overview of the Aggregate Advisor
- In-memory aggregates
- Database aggregates
- Modeling in-database aggregates
- Modeling user-defined in-memory aggregates
- Cache Priming Techniques
- Cache size effective practices
- Scenarios for performance tuning

11.1 Performance implications of OLAP-style reports

Query performance is mostly determined by how much data is fetched from the caches and the database, and how much information the report is trying to show the user. In general, when querying against large warehouse data, use context filters to constrain the data that is processed.

An expectation is that most online analytical processing (OLAP)-style reports and analyses that run against Cognos Dynamic Cubes show data at high levels of hierarchies. For example, the report on the Gross Profit for Retailers Region and Products Product Line.

Any deeper analysis or drilling-down has filters to highly constrain the resulting data. For example, showing gross profit for a particular product in a specific retailer site.

A report is unlikely to be at the deepest level of all hierarchies. If it is at the bottom of a large dimension hierarchy, it probably is highly constrained. For example, showing data for one employee by region, or the top 10 products.

> **Note:** Reporting and analysis users need to be aware of the performance implications if reports are not filtered or scoped appropriately. Most OLAP-style reports and analyses are filtered and scoped to show data at high levels of aggregation. Any deeper analysis should have filters to highly constrain the resulting data. For example, showing data for customers within a region, as opposed to including all leaf-level customers of an entire dimension.

11.2 Aggregate awareness in Cognos Dynamic Cubes

Use of aggregates can significantly improve performance of queries by providing data that is aggregated at levels higher than the grain of the fact.

Database aggregates are tables of precomputed data that can improve query performance by reducing the number of database rows that are processed for a query. Not all relational databases have built-in aggregate support, or if they do have built-in support, do not do it well for all types of queries. Cognos Dynamic Cubes brings up the aggregate routing logic into its query engine where the multi-dimensional OLAP context is preserved and can better determine whether to route to aggregates.

Cognos Dynamic Cubes can also use aggregate tables that the database optimizer does not know about, or the database optimizer might not route to because of complex OLAP-style SQL, even in cases where there is good built-in aggregate support in the database.

The aggregate tables can be regular tables that happen to hold aggregated data. The Great Outdoors warehouse has an existing database aggregate table:

```
AGGR_TIME_PROD_OM_FACT
```

For a description of how to include database aggregates in the model to enable the cube to be database aggregate-aware, see 11.6, “Modeling in-database aggregates” on page 394. By enabling the cube to be aware of database aggregate tables, Cognos Dynamic Cubes ensures routing to them should they apply to the query.

In addition to guaranteed routing to aggregates in the database, a major performance feature of Cognos Dynamic Cubes is its support of aggregates that are in memory. Not only do in-memory aggregates provide precomputed data, they can greatly improve query performance because the aggregated data is stored in memory in the aggregate cache of the dynamic cube.

For warehouses that do not yet have aggregates, or want to supplement existing database aggregates with in-memory and other in-database aggregates, run the Aggregate Advisor as part of an optimization workflow to get recommendations for aggregates, both in-memory and in-database.

> **Note:** The easiest way to get aggregates that will significantly improve query performance is to run the Aggregate Advisor. The Aggregate Advisor provides in-memory aggregates for the aggregate cache, and recommendations for database aggregates.

11.3 Overview of the Aggregate Advisor

The Aggregate Advisor offers aggregate recommendations that provide coverage for OLAP queries against Cognos Dynamic Cubes based on cube model analysis and, optionally, a query workload. Before running the Aggregate Advisor, the expectation is that the dynamic cube is published to the content store, can be started successfully, and that reports and analysis run and return correct results. The Aggregate Advisor is available in the IBM Cognos Dynamic Query Analyzer.

11.3.1 Model-based analysis

At the core of the Aggregate Advisor logic is its theoretical, model-based evaluation. The Aggregate Advisor evaluates the cube model and experiments with potential slices of the cube, and evaluates these candidates on various heuristics. Some examples of heuristics include the balance between coverage in relation to other candidates so that aggregates can be logically derived from one another and number of manageable aggregates, and selectivity based on effective size relative to the underlying fact data.

As of Cognos Business Intelligence version 10.2.2 it is also possible to define your own in-memory aggregates in your model. These in-memory aggregates are taken into account during any analysis that uses the Aggregate Advisor to prevent duplication.

11.3.2 Workload-based analysis

The aggregate recommendations can be much more relevant to actual user queries if information from a representative query workload is captured. If workload logging is enabled for the cube, any report that is run, drill behavior, and any user action in the Cognos studios that results in a query that is processed by the Cognos Dynamic Cubes engine are logged in a separate workload log.

Running the Aggregate Advisor to consider workload information in the workload log will significantly weigh the measures and slices that are actually affected by users so that the aggregate recommendations are likely to be more reflective of query usage of the data.

For more information about how to enable workload logging, see “Enabling workload logging” on page 347.

Note: The Aggregate Advisor can be run without having any workload information. To get better, more relevant aggregate recommendations, a representative workload should be captured and used by the Aggregate Advisor.

11.3.3 Preparing your environment for running the Aggregate Advisor

IBM Cognos Dynamic Query Analyzer (DQA) is a tool to analyze queries for optimization, and it is the client tool to run and manage Aggregate Advisor recommendations. See 3.5, "Installing Cognos Dynamic Query Analyzer" on page 76 for information about installing DQA.

Configuration settings before running the Aggregate Advisor

Before running the Aggregate Advisor, you must ensure that the user running the Aggregate Advisor has write access on the cube data source and configure DQA to point to the Cognos server.

Setting user permissions

For a secured Cognos server, the user who is used in DQA should have write access on the Cognos Dynamic Cubes data source. By having write access on the Cognos Dynamic Cubes data source, the user can save a set of in-memory aggregate recommendations, and clear a set of in-memory aggregate recommendations from the content store. By default, the Cognos Dynamic Cubes data source inherits the permissions from the Cognos namespace with only read, execute, and traverse permissions.

If the specified user does not have write permission on the Cognos Dynamic Cubes data source, the user is still able to start a run of the Aggregate Advisor. However, the user will not be allowed to save or clear any in-memory aggregates that are associated with the dynamic cube.

Use the following steps to set write access on the Cognos Dynamic Cubes data source:

1. Go to IBM Cognos Administration.
2. Select the Configuration tab.
3. Select **Data Source Connections.**
4. For the `goslw_sales` dynamic cube data source, select **Set properties.**

5. Select the Permissions tab (Figure 11-1).

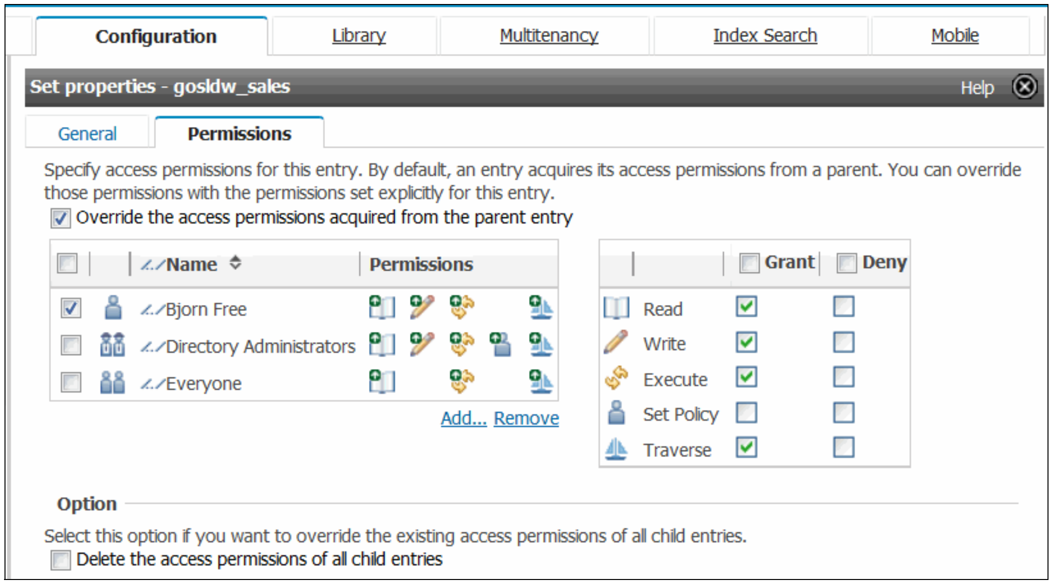


Figure 11-1 Set permissions on the Cognos Dynamic Cubes data source

6. If the user is not already granted read, write, execute, and traverse permission, complete these steps:
   a. Select **Override the access permissions acquired from the parent entry**.
   b. Click **Add** to add the selected entries.
   c. Click **OK**.
   d. Select the newly added selected entries and grant read, write, execute, and traverse permissions.
7. Click **OK.**

Setting preferences to Cognos server

Instances of the Aggregate Advisor run within the Cognos Dynamic Cubes engine that is a subcomponent of the query service. Therefore, the first step is to configure DQA to point to the dispatcher with the query service where instances of the Aggregate Advisor, started from DQA, will be run.

Use the following procedure and see Figure 11-2 to set the configuration values:

1. Start **Cognos Configuration** from the start menu or from the Server Preferences window in Dynamic Query Analyzer.
2. Specify locations for the Dispatcher URI and the Gateway URI for the Cognos server with the published `gosldw_sales` cube. The Dispatcher URI should be the dispatcher that the query service is running on because that is where the workload information is stored.

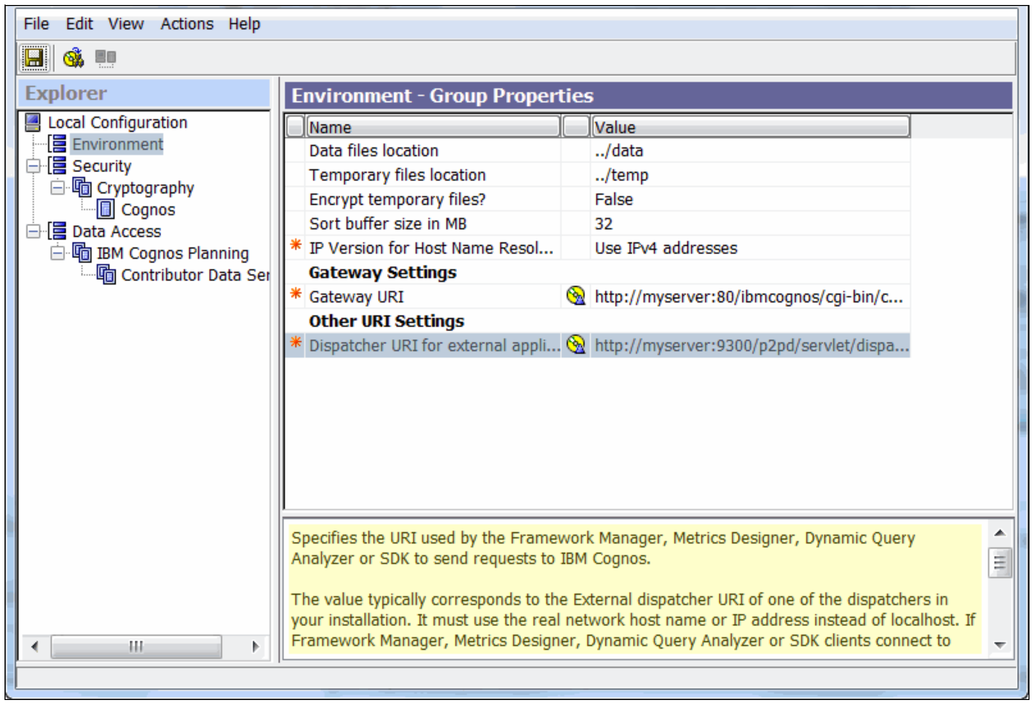


Figure 11-2 Cognos Configuration for Dynamic Query Analyzer

For a secured Cognos server, to set your credentials follow the procedure below. Note that you are prompted the first time that you connect to the server if you choose not to set the credentials beforehand. See Figure 11-3 on page 347.

1. Start Dynamic Query Analyzer.
2. Click **Window** → **Preferences**.
3. Select **Cognos Server**.
4. Specify the **Name**, **Password**, and **Namespace** to use.
5. Click **OK.**

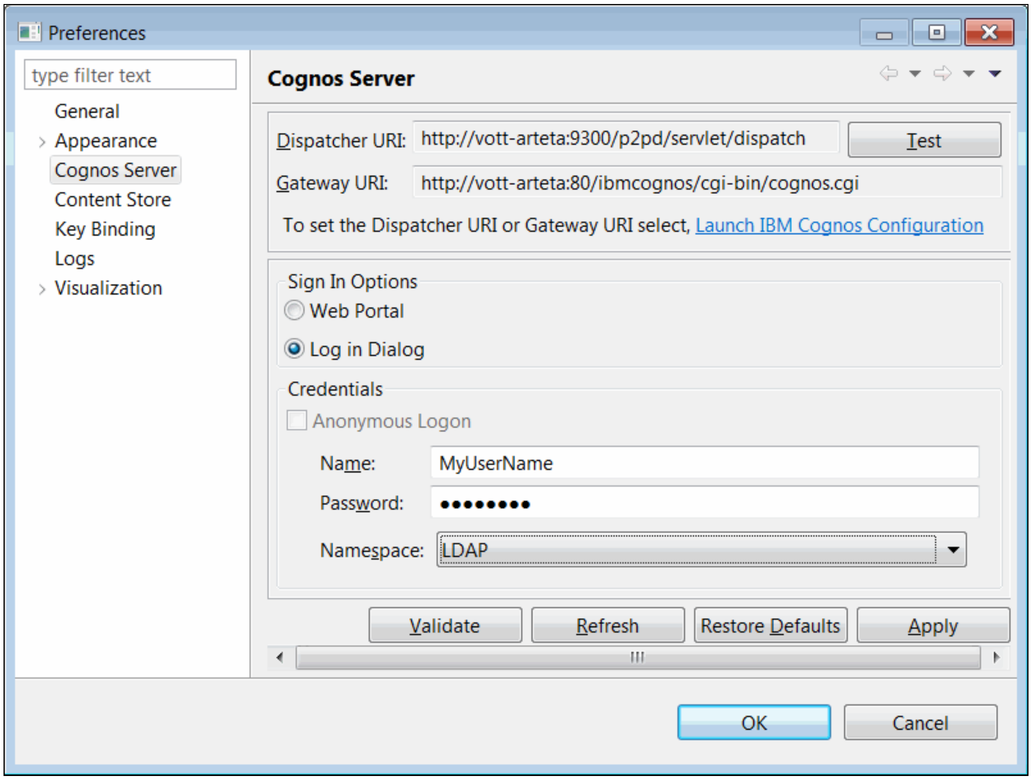


Figure 11-3 Set Cognos server in Preferences

Considerations before running the Aggregate Advisor

The Aggregate Advisor can be run without having any workload information. Not having workload information might be the case during new cube development when a representative query workload is not yet available or determined. In this scenario, the Aggregate Advisor uses only its theoretical, model-based evaluation logic to produce recommendations for the `gosldw_sales` dynamic cube.

For better aggregate recommendations that are relevant to actual user queries, a representative workload should be captured and provided to the Aggregate Advisor to do workload-based analysis to complement the model-based analysis.

Enabling workload logging

Enable workload logging for `gosldw_sales` by setting the properties of the dynamic cube:

1. Go to IBM Cognos Administration.
2. Select **Status.**
3. Select **Dynamic Cubes.**
4. Select **All Server Groups**.
5. Select the server group the server is in.
6. Select the dispatcher.
7. Select the cube.
8. In the **QueryService** menu, select **Set properties.**

9. Select **Enable workload logging** in the Value column (Figure 11-4).

10. Click **OK.**

You do not have to restart the cube before any workload is captured. It takes approximately one minute for the workload logging property setting to propagate to the running cube to take effect. Wait one minute before running a representative query workload to ensure the capture of workload logging.

> **Note:** In Cognos Business Intelligence versions before 10.2.1 Fixpack 3, the cube must be restarted for updates to the `Enable workload logging` property to take effect.

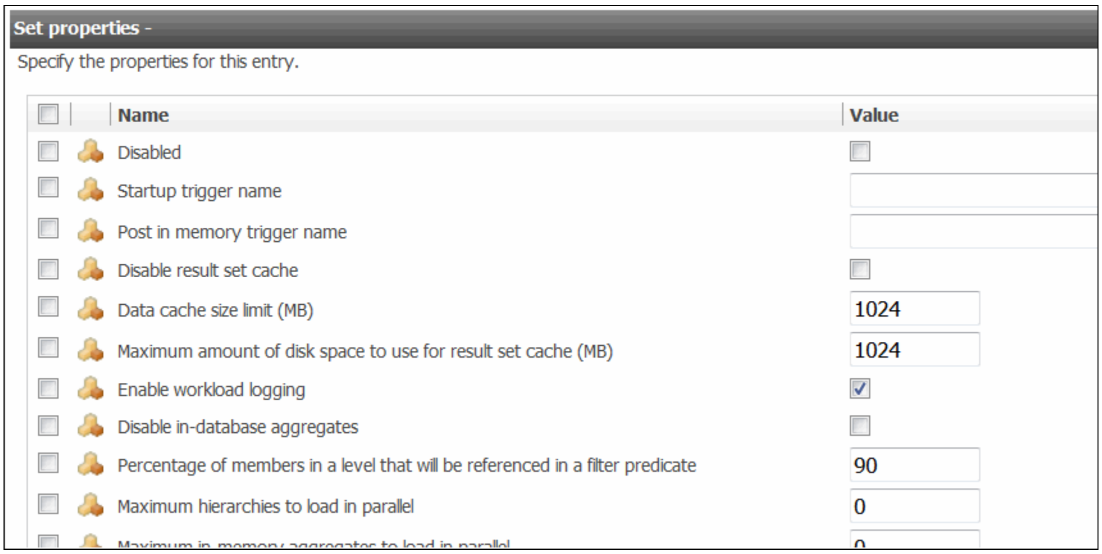


Figure 11-4 Enabling workload logging in the cube properties

Running a representative query workload

For a cube with workload logging enabled, any report run, drill behavior, or any user query action against the cube is captured in the workload log of the cube.

Workload log file

When workload logging is enabled, user query actions are appended to the existing workload log file. If none yet exists, one is created.

The workload log file for Cognos Dynamic Cubes is separate from the Cognos Dynamic Cubes server trace log files. The workload log is stored locally on the server that processes the query in a single file under a directory with the name of the cube. Relative to the server installation directory, the workload log file is in the following location:

```
logs\XQE\ROLAPCubes\gosldw_sales\workload_log.xml
```

The only consumer of the workload log file is the Aggregate Advisor. Although the file can be opened in a text editor to verify that workload entries are being populated, it is not intended to be human-readable.

Clearing the workload log

Workload logs can be cleared by selecting the **Clear workload log** menu option on the cube that is listed under the QueryService in IBM Cognos Administration. This administrative action clears the workload log file found in the location described in “Workload log file” for cubes that do not have the cube configuration property `Automatic optimization of in-memory aggregates` enabled.

The workload logs for cubes that do have the `Automatic optimization of in-memory aggregates` configuration property enabled are found in a different location on the server and are not affected by the **`Clear workload log`** command.

Figure 11-5 shows how to manually clear a workload log for a cube.

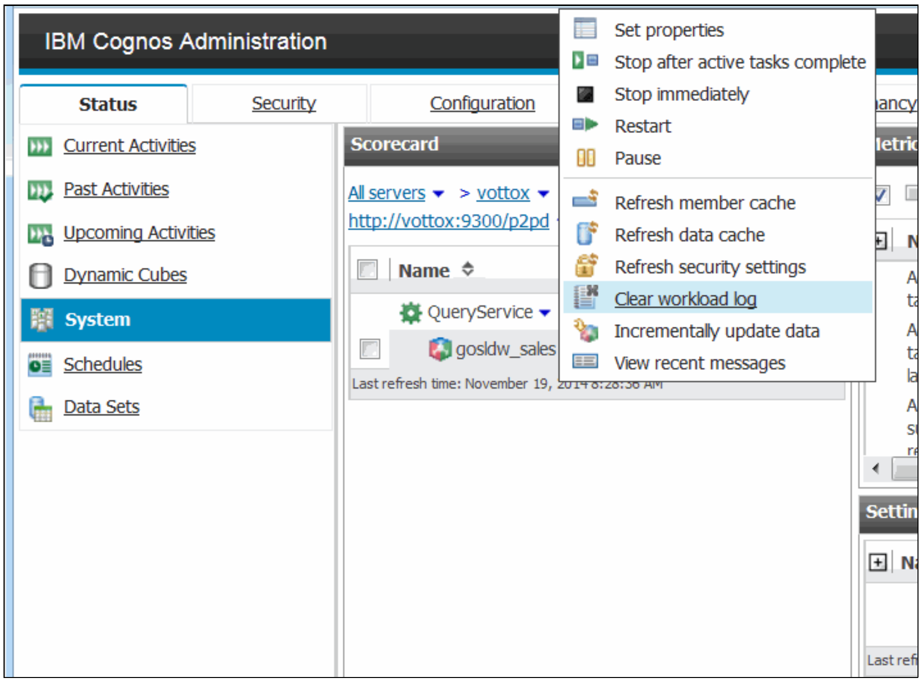


Figure 11-5 Manually clearing workload log for a cube

Clearing of workload logs can also be scheduled as a query service administration task by completing these steps:

1. Go to IBM Cognos Administration.
2. Select **Configuration**.
3. Select **Content Administration**.
4. Select the **New Query service administration task** action on the toolbar.
5. Select **Dynamic cube** from the menu.
6. Specify a name for the task. Click **Next**.
7. In the **Operation** field, select **Clear workload log**.
8. In the Server Group field, select the server group that the QueryService is in, such as **Default Server Group**.
9. In the Dispatcher field, select the dispatcher.
10. Select the cubes for which the operation is to be performed, such as `gosldw_sales`. Click **Next**.
11. Select the action frequency and complete the wizard by clicking **Finish**.

Managing different workloads

When the Aggregate Advisor considers workload information, it picks up the workload log entries from its specific workload log location. If there are different workload characteristics at different times of the year, for example at month-end and quarter-end, then one approach to actively manage server performance is to capture these various workloads in separate workload log files.

Before running the Aggregate Advisor, ensure that the specific workload for month-end is in place so that the recommended aggregates are weighted toward the month-end workload. Repeat the procedure, replacing with the quarter-end workload, and run the Aggregate Advisor to obtain a different set of aggregate recommendations.

The history of these results is stored in the DQA Advisor Results view, and can be used to apply different sets of aggregates when needed.

If replacing the workload log files on the query service cannot be done easily, then an alterative approach is to use the workload filter options in the Aggregate Advisor. Switching between workload log files can be difficult to do if the user who is running the Aggregate Advisor does not have access to the Cognos server to replace the workload log file, or if the Cognos server is already running with workload logging enabled and has a hold on the workload log file. In this approach, the workload log file contains both the month-end and quarter-end workload, and the Filter Workload Information window can be used to specify which queries the Aggregate Advisor should consider.

Disabling workload logging

After a representative set of reports and user actions are complete, disable workload logging. Follow similar steps in IBM Cognos Administration to clear the Enable workload logging property for the cube. Wait approximately one minute for the workload logging property setting to propagate to the running cube to take effect.

11.3.4 Running the Aggregate Advisor wizard

> **Note:** Before running the Aggregate Advisor for the first time, configure DQA to point to the dispatcher with the query service where instances of the Aggregate Advisor will be run.
>
> For a secured Cognos server, the specified user should have write access on the Cognos Dynamic Cubes data source.

To run the Aggregate Advisor, click **File** → **Run Aggregate Advisor**. Then complete the steps in the remainder of this section.

Selecting the cube

The first page of the Cognos Dynamic Cubes Aggregate Advisor wizard allows you to select a cube on which to run the Aggregate Advisor. A list of dynamic cubes is retrieved from the content store.

Having the dynamic cube assigned to a query service is not required, although it is likely that it is already assigned and configured as a cube under the query service. Having a dynamic cube that was verified to start and has successfully run reports is a prerequisite for running the Aggregate Advisor. Ensure that reports can successfully run and return correct results before proceeding.

The cube does not need to be started or running at the time the Aggregate Advisor is run. The Aggregate Advisor issues queries to the underlying relational data source during the various phases of analyses to determine factors such as slice cardinalities. The Aggregate Advisor should be run during off-peak, non-critical business hours to minimize impact to users who are running reports and analysis.

The list of dynamic cubes available does not include any virtual cubes. To optimize the virtual cube using aggregates, run the Aggregate Advisor against the constituent base cubes of the virtual cube.

To run the Aggregate Advisor against the `gosldw_sales` cube, select **gosldw_sales** as shown in Figure 11-6, and click **Next**.

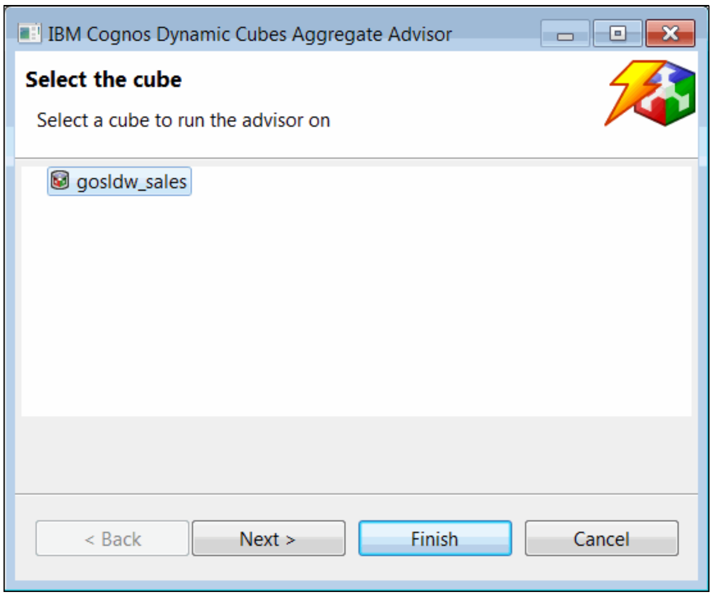


Figure 11-6 Selecting the cube on which to run the Aggregate Advisor

Specifying general options

In the next window, select general options such as whether to include workload information, how much memory to allot for in-memory and in-database aggregates, and maximum time limit for the run.

Figure 11-7 shows the Specify General Options window without using the query workload information in cases where you do not want to use a workload.

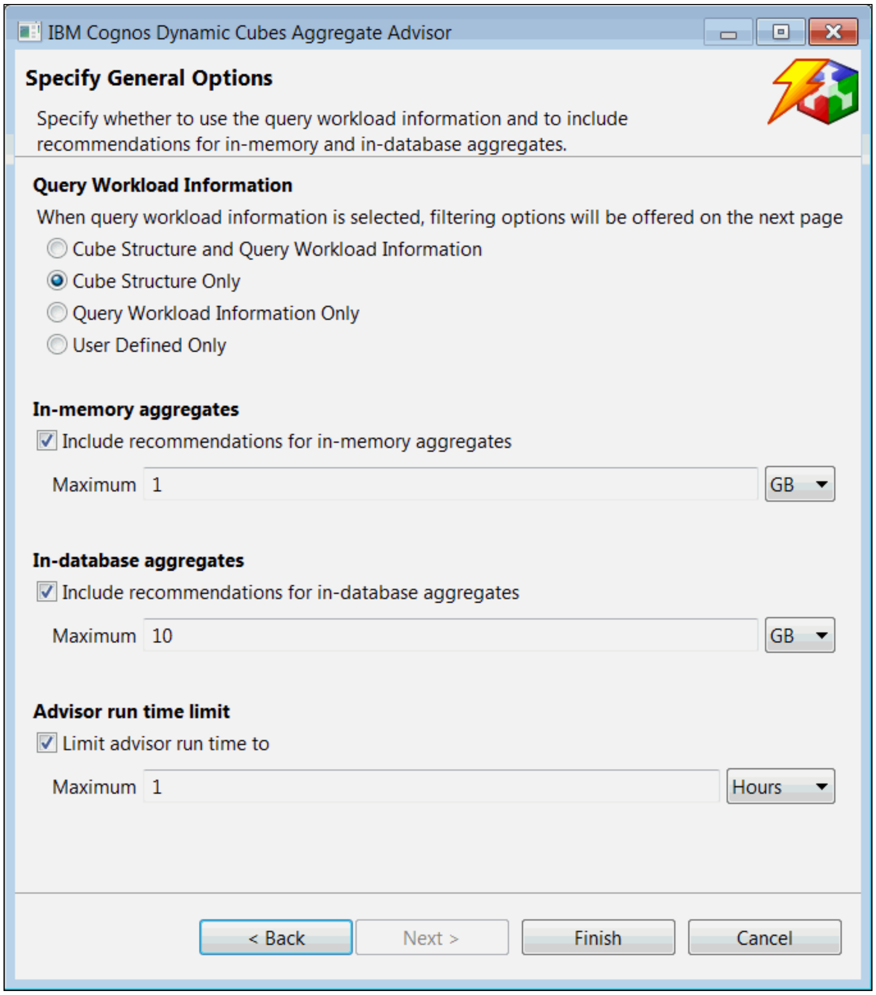


Figure 11-7 Specifying general options without using query workload information

The following fields are shown in Figure 11-7:

- **Query Workload Information**

  There are four options:

  - **Cube Structure and Query Workload Information**. Uses both model base analysis and the query workload. If there is no workload, the Aggregate Advisor runs and uses only model-based analysis. This option is the default option.
  - **Cube Structure Only**. Uses only model-based analysis.
  - **Query Workload Information Only.** Uses only the query workload.
  - **User Defined Only.** Uses only the user-defined in-memory aggregates assigned in the model.

- **In-memory aggregates**

  The default value for the maximum amount of memory to allot for in-memory aggregates is 1 GB.

- **In-database aggregates**

  The default value for the maximum amount of memory for in-database aggregates is 10 GB.

  For guidelines about how to determine the maximum values for in-memory aggregates and in-database aggregates, see *IBM Business Analytics Proven Practices: Dynamic Cubes Hardware Sizing Recommendations* at:

  http://ibm.biz/DynamicCubesHWSizingRec

  These values can be also calculated using the hardware sizing calculator in IBM Cognos Cube Designer by selecting the cube that you are sizing in the model used to create it as shown in Figure 11-8.

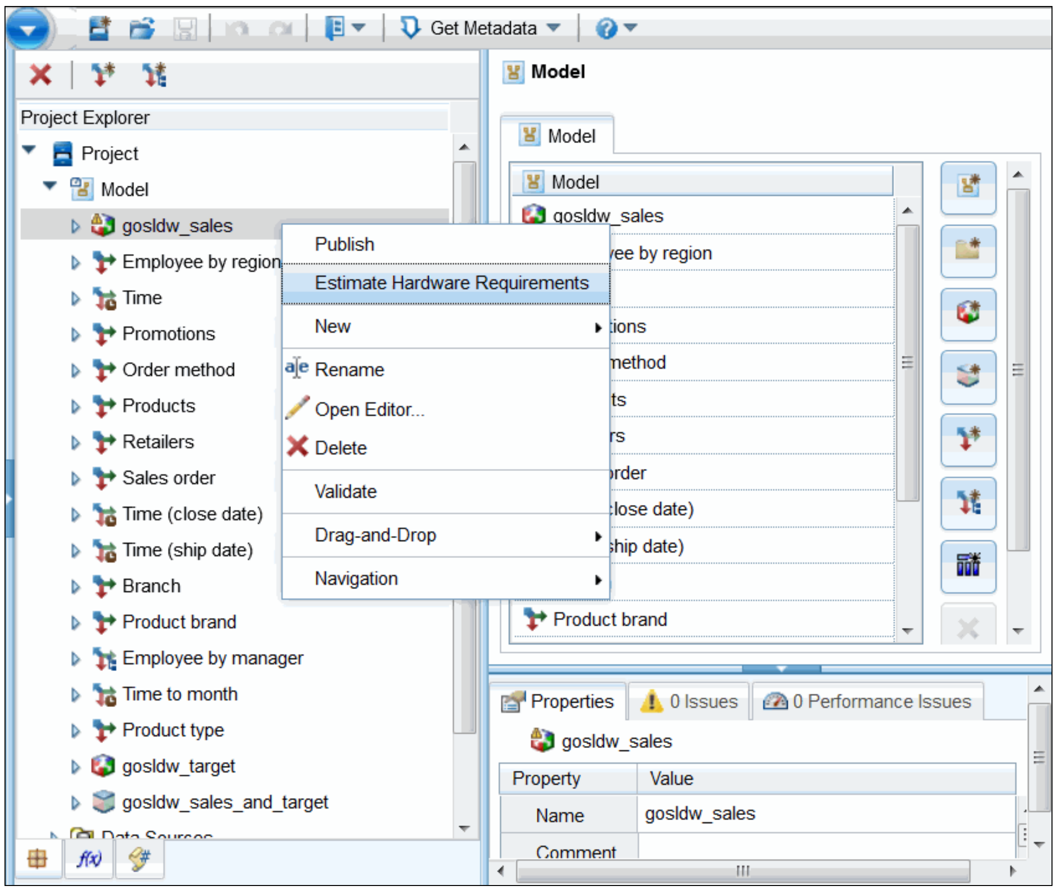


Figure 11-8 Estimate Hardware Requirements calculator in Cognos Cube Designer

For the Great Outdoors warehouse, these defaults are more than sufficient.

Specifying a very large amount of memory to use for in-memory and in-database aggregates in the wizard does not guarantee that the Aggregate Advisor recommends as many aggregates as it can to fill the space.

The Aggregate Advisor only recommends aggregates that it confidently determines are of the greatest benefit. Specifying workload information is a large factor in that confidence because it is better to have aggregates in-memory that are useful and will be used. In a scenario that has no workload information, confidence is factored by dimensional characteristics.

- **Advisor runtime limit**

  This option specifies how long to allow the Aggregate Advisor to run. The default value is one hour. For the `Great Outdoors` warehouse, and in most cases with smaller data sets, this is more than enough time. For larger enterprise warehouses, in the terabytes of uncompressed data, run the Aggregate Advisor for several hours or overnight. You can stop the run to get any recommendations it has so far. However, the longer the Aggregate Advisor is allowed to run, the more time it has to complete its iterations of analysis and potentially have better recommendations, especially for those refined from model-based analysis.

If you want to run with a workload, select **Cube Structure and Query Workload Information** or **Query Workload Information Only**, as shown in Figure 11-9.

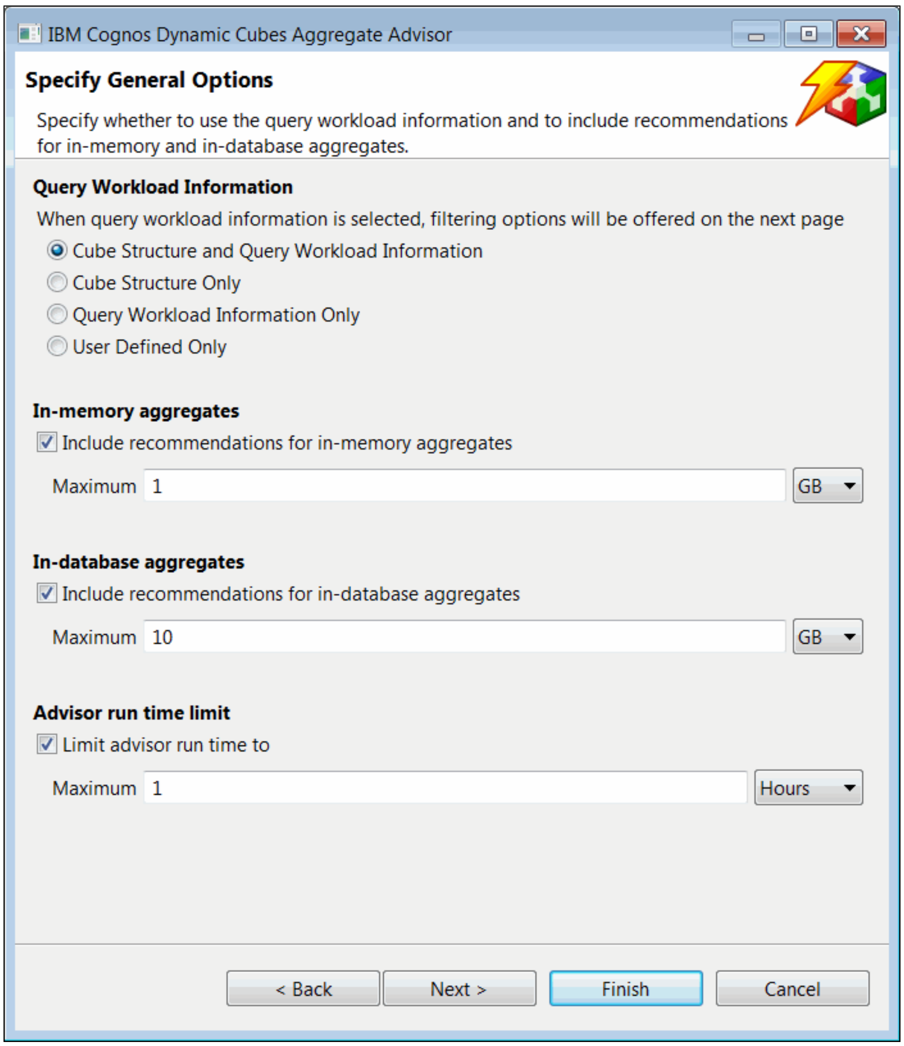


Figure 11-9 Specifying general options including use of available query workload information

To narrow the workload information in the workload log for consideration, click **Next** to specify workload information filters. Otherwise, to run the Aggregate Advisor with all the information in the workload log, click **Finish**.

Filtering workload information

With the **Cube Structure** and **Query Workload Information** or **Query Workload Information Only** option selected on the Specify General Options window, clicking **Next** polls the query service about its workload information.

If no workload information is available on the server, a message is displayed that contains instructions for how to produce workload information. The Aggregate Advisor can still be run, although there is no workload information. It provides recommendations based on only model-based analysis.

If there is workload information available on the server, the Filter Workload Information window options for package names, report names, user names, and time period are populated with the actual values from the workload log. If no filter options are selected when you finish the wizard, then all the information in the workload log is considered by the Aggregate Advisor. If filter options are selected and specified, the Aggregate Advisor considers only those queries that satisfy these conditions from the workload log.

Finishing the wizard and running the Aggregate Advisor

To complete the wizard and run the Aggregate Advisor, click **Finish**.

If an option to use query workload information is selected and there is workload information available, the Aggregate Advisor performs the workload analysis first and heavily weighs the recommendations first, so any initial recommendations available are rough and mostly skewed toward the workload. Then the recommendations are continually refined and supplemented with model-based analysis.

A request is sent from DQA to the dispatcher and query service to initiate a run of the Aggregate Advisor on the Cognos server. When the Aggregate Advisor is running, it does not need to be constantly monitored. The progress monitor window that is displayed has a **Run in background** option. When this option is selected, the progress monitor window is minimized and the recommendation results are returned to DQA when complete.

The progress graph shows time in minutes on the bottom, X-axis, and a unitless fuzzy value for the Y-axis. The value that is plotted is not a concrete value, such as progress complete or data coverage. It is intended to give insight into how far along and how confident the Aggregate Advisor estimates its candidates are at a specific time.

The values should be taken in context, relative to the values of Aggregate Advisor on the same graph. A shallow, gradual slope upward indicates processing of aggregate candidates, and sharp jumps up indicate a significant improvement or iteration of the aggregate candidate analysis. Examples of such analysis iterations are completion of workload analysis, determining cardinality of a candidate that is of acceptable size, or consolidation of one or more candidates. The shape of the progress graphs can vary from one cube model to another, by fact data skew, and the type of workload information available.

The first significant jump in the graph is most likely when the Aggregate Advisor has an initial set of recommendations. At this point, the **End** button on the progress monitor window is enabled. Selecting **End** stops the Aggregate Advisor run, returning any recommendations that it has generated so far. However, the initial sets of recommendations have notably rough estimate sizes. The intent of these initial recommendations is to quickly provide a set of recommendations that are most likely to be used. These recommendations are correlated from the query workload information, if that option is selected to be considered. If no workload information is considered, the initial recommendations are high-level slices from a region of the cube model.

The subsequent iterations of the Aggregate Advisor analysis after determining initial recommendations further refines these candidates by running queries to determine the actual cardinality of the candidates, consolidation of similar candidates, rejecting those that are too large in relation to the fact table, and supplementing with other model-based candidates.

Figure 11-10 shows an example of the progress graph while the Aggregate Advisor runs.

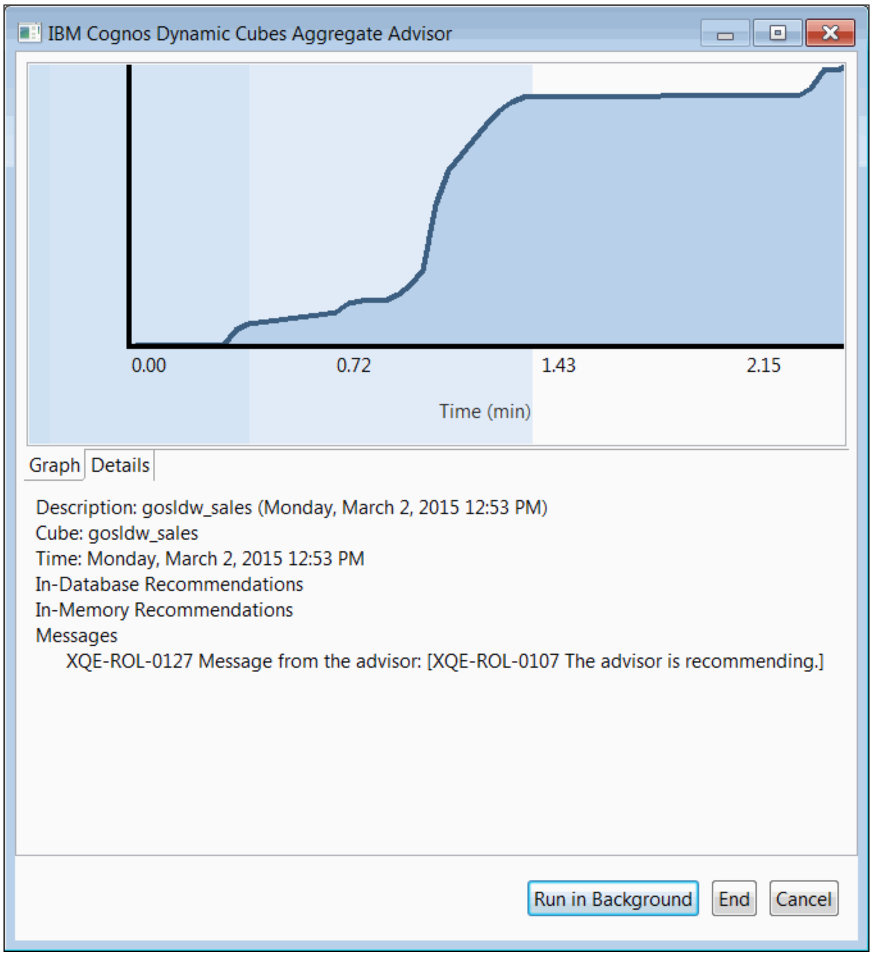


Figure 11-10 Example of the progress graph while the Aggregate Advisor runs

Results populated in the Advisor Results view

When the Aggregate Advisor completes its analysis, results are returned to DQA and displayed as an entry in the Advisor Results window. The Advisor Results window contains the results from the most recent run, and a history of previous run results that were stored on the server.

Figure 11-11 shows the Example Advisor Results view with results from multiple Aggregate Advisor runs.

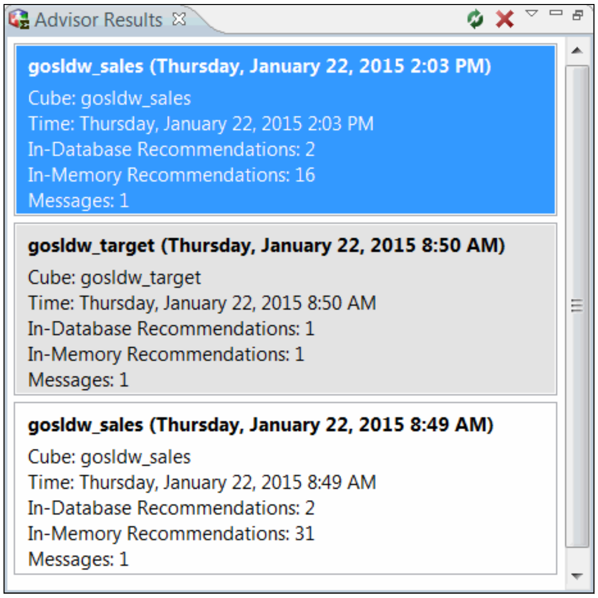


Figure 11-11 Example Advisor Results view with results from multiple Aggregate Advisor runs

11.3.5 Opening Aggregate Advisor results and viewing details

As shown in Figure 11-11, when the Aggregate Advisor completes its analysis, results are returned to DQA and displayed as an entry in the Advisor Results window. The Advisor Results window contains the results from the most recent run, and a history of previous run results.

Opening an advisor result to view details

To open an Aggregate Advisor result and take action, in the Advisor Results view, double-click the entry. The first line of the advisor result entry consists of the cube name and the date and time the run was initiated. A detailed view of the advisor result opens with three tabs: General, In-database, and Options.

General tab

The General tab lists summary and details for both in-memory and in-database recommendations in terms of the aggregate name, estimated size, hierarchy levels, and measures, in a scrollable view. The summary fields enable quick comparison of summary information. Notice that although 1 GB was specified as the maximum for in-memory aggregates, in this run of the Aggregate Advisor, it recommends 16 aggregates, totaling less than an estimated 35 MB.

Figure 11-12 shows an example of the content of the General tab for the Aggregate Advisor results.

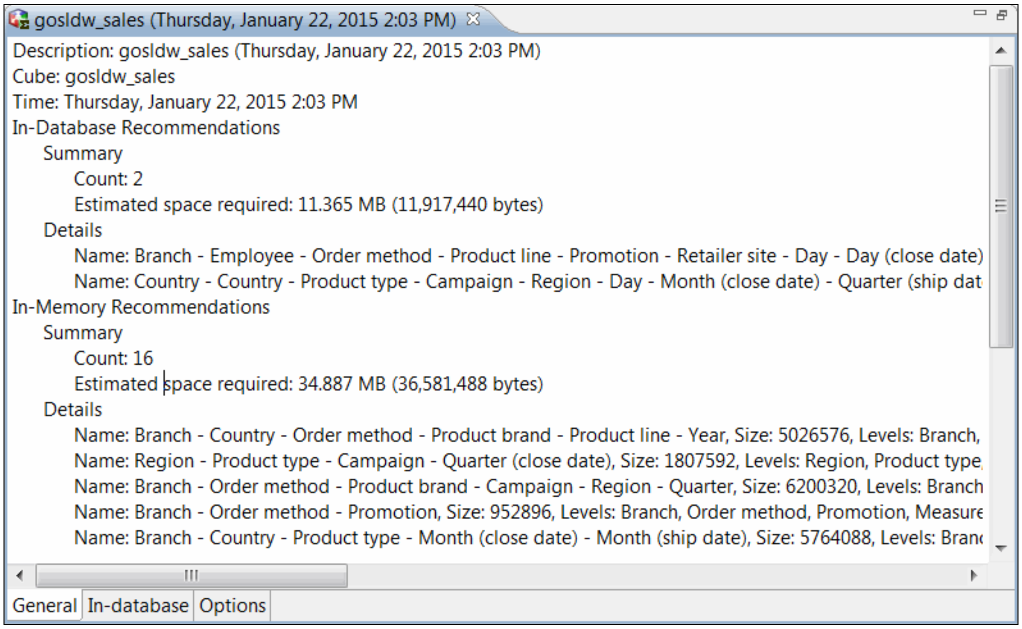


Figure 11-12 Example of Aggregate Advisor results - General tab contents

In-database tab

To view the detailed output for in-database aggregate recommendations, open the In-database tab.

The In-database tab contains information that describes the logical aggregate, such as the measures and hierarchy levels, and instructions for next steps. For each logical aggregate, there is an example SQL scriptthat illustrates how to aggregate the data. However, this information is not an executable SQL script to create database aggregate tables. These recommendations are of database aggregates that a database administrator (DBA) can take and then choose which aggregates to create.

Figure 11-13 shows an example of the content of the In-database tab for the Aggregate Advisor results.

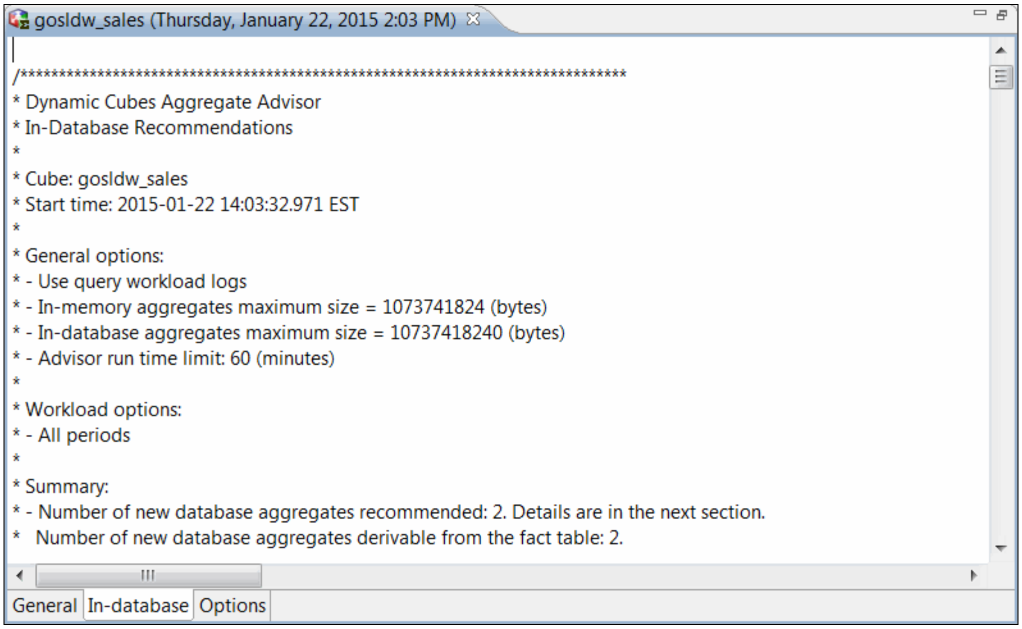


Figure 11-13 Example Aggregate Advisor results - In-database tab contents

Options tab

To view the options that are specified in the Aggregate Advisor wizard corresponding to the Aggregate Advisor run, select the Options tab.

Figure 11-14 shows Aggregate Advisor results in the Options tab.

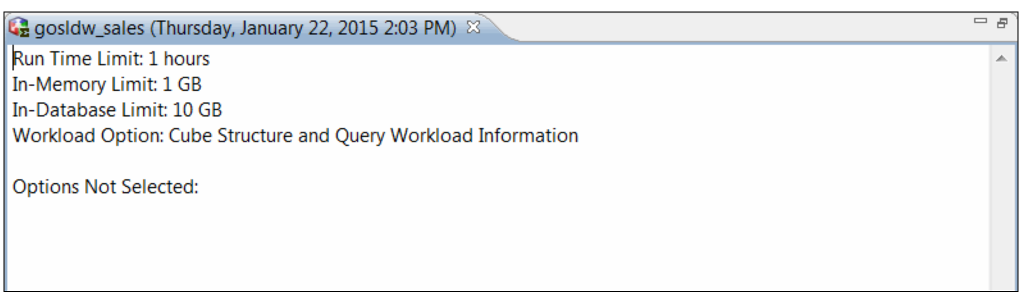


Figure 11-14 Example of Aggregate Advisor results - Options tab contents

Saving aggregate recommendations and next steps

To save aggregate recommendations, open the Advisor Results detailed window (see Figure 11-12 on page 358) that contains the recommendations to apply. With the Advisor Results detailed window open, three actions are now available under the **File** menu, as shown in Figure 11-15 on page 360:

- Save In-Database Recommendations
- Apply Selected In-Memory Recommendations
- Clear In-Memory Recommendations

Figure 11-15 shows the **File** menu options for Aggregate Advisor recommendations.

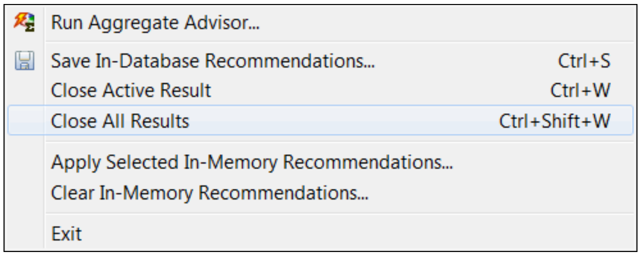


Figure 11-15 File menu options for Aggregate Advisor recommendations

Saving in-database recommendations to file

To save the database aggregate recommendations shown in the in-database tab to an output file, click **File** → **Save In-Database Recommendations**.

The output text file contains the same information as what is available in the In-database tab. This file can be given to the DBA and modeler as the basis for creating the database aggregates and modeling their support in the cube.

This in-database recommendations file describes the logical aggregates, and provides an example SQL statement that illustrates how to aggregate the data. However, this information is not an executable SQL script to create database aggregate tables. These recommendations are of database aggregates that the DBA can take and then choose which aggregates to create. The DBA might even choose to modify the SQL statement to subset the aggregate to only a certain subset of members instead of including the entire level, such as specific years from the `Time` dimension. See 11.5, "Database aggregates" on page 385 for more information about the contents of this file and other considerations when working with the DBA to create database aggregates.

After the database aggregates are created by the DBA, the modeler can then take the relevant logical aggregate information shown in this tab to model aggregate cubes in IBM Cognos Cube Designer. See 11.6, "Modeling in-database aggregates" on page 394 for a description of how to include database aggregates in the model to enable the cube to be aware of the database aggregate.

> **Important:** By considering the in-database recommendations, the DBA can choose which aggregates to create and how to create them. The DBA must communicate any differences from what is described in the in-database recommendations file to the modeler.

Applying in-memory recommendations to the cube

To apply the set of in-memory aggregate recommendations in the advisor result to the dynamic cube, click **File** → **Apply Selected In-Memory Recommendations**.

The set of in-memory aggregate recommendations are saved to the content store and are associated with the dynamic cube data source. Because in-memory aggregates are meant to be a turn-key aggregate management solution, all the in-memory aggregates from a specific Aggregate Advisor run result should be stored together as a set. However, you can choose to select or click to clear individual in-memory aggregates from the set, and add other individual in-memory aggregates from other advisor run results into the applied set.

Figure 11-16 shows the Select the aggregates to apply window with all in-memory aggregates selected to be applied.

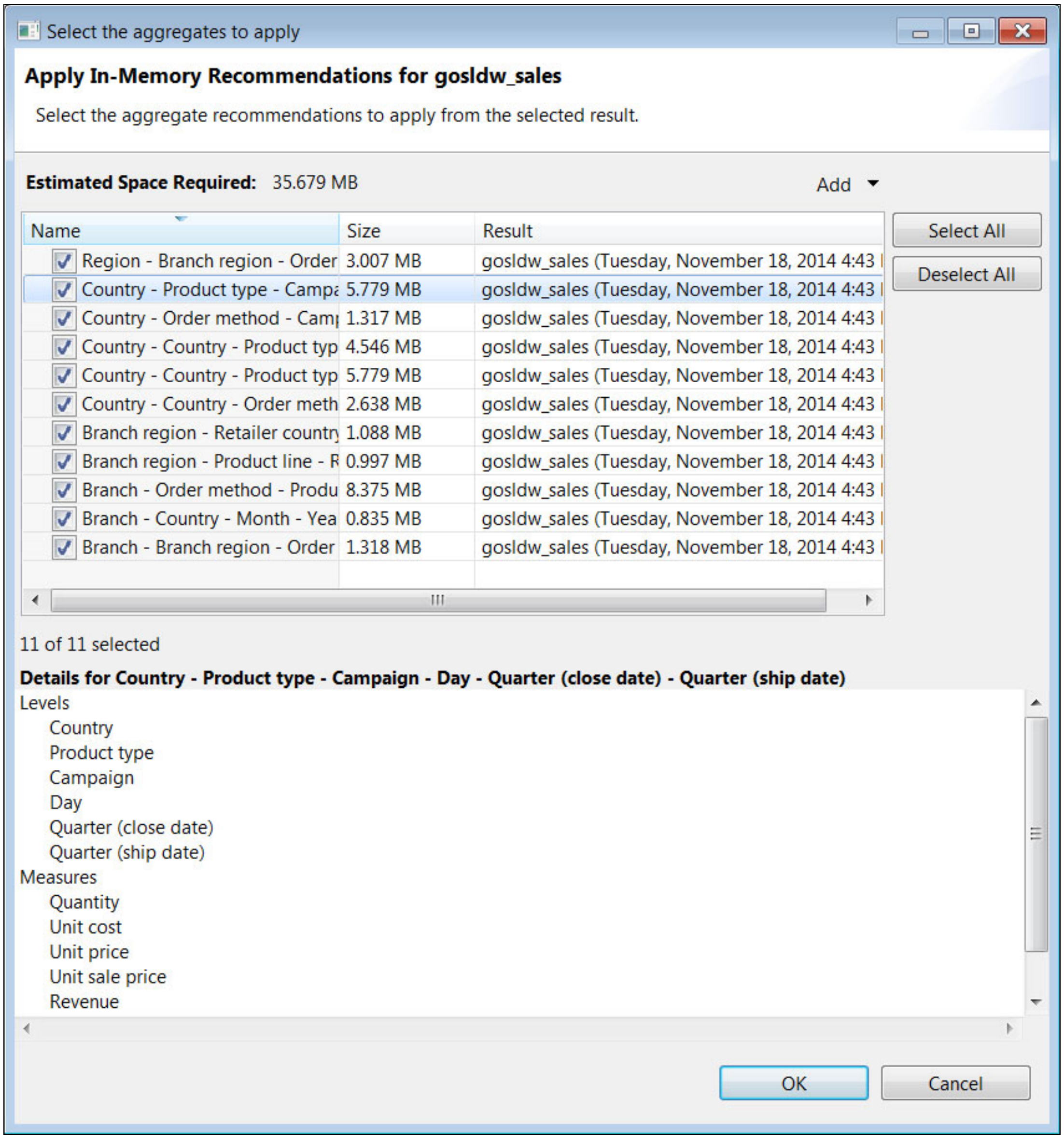


Figure 11-16 Select the aggregates to apply window

When storing the in-memory aggregate definitions, DQA determines whether the cache size limit of the runtime aggregate is sufficient to load and hold the aggregates. The runtime aggregate cache size limit is the value for the `Maximum space for in-memory aggregates` property in cube configuration in the query service. A warning message is displayed if the cube configuration property value is less than the estimated size. Select **Accept** to proceed with saving the aggregate definitions to the content store.

Figure 11-17 shows the warning message that might be displayed when in-memory aggregates are saved.

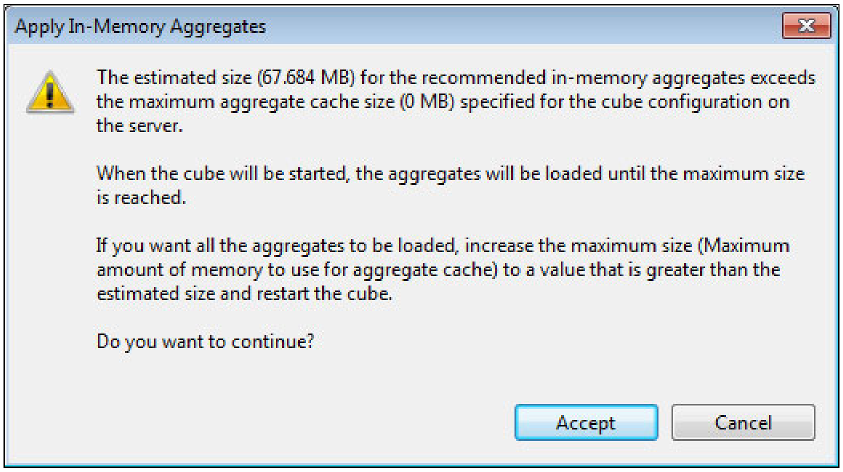


Figure 11-17 Warning message displayed when saving in-memory aggregates

After the in-memory recommendations are saved to the content store and the aggregate cache is enabled, the next time the cube is started, the in-memory aggregates are loaded. The maximum space for in-memory aggregates value can be set in the properties of the cube (see Figure 11-18). The aggregate cache is enabled if this value is greater than zero.

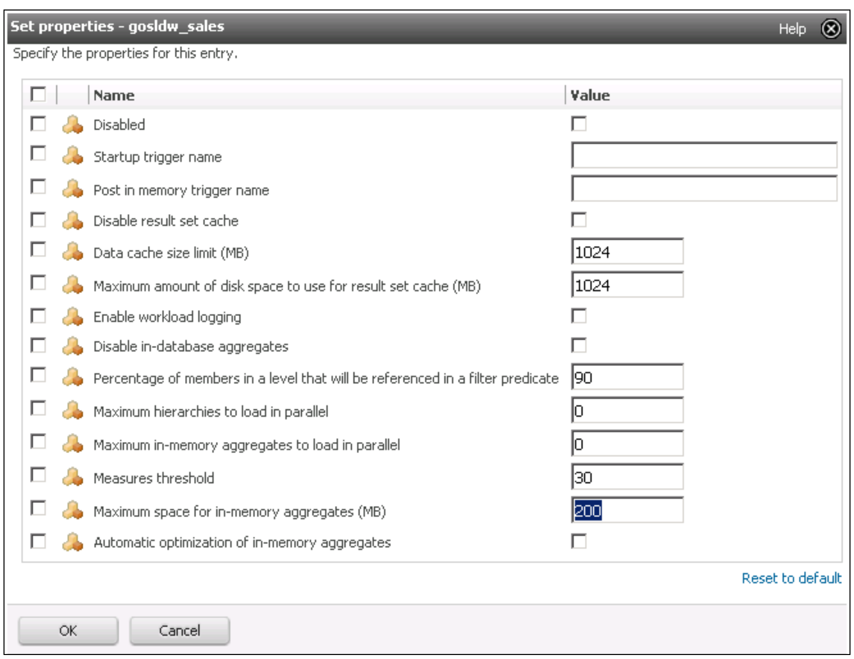


Figure 11-18 Maximum Space for In Memory Aggregates property

See 11.4.4, "User-defined in-memory aggregates" on page 373 for more information about the loading and processing of the aggregate cache.

Note: To have the newly saved in-memory aggregates loaded in the cube, the cube must have the aggregate cache enabled and be started or restarted.

Opening other advisor results and saving other sets of in-memory aggregate definitions from the other advisor results will overwrite any existing definitions that are associated with the cube.

Consider, for example, two advisor results in the Advisor Results view, such as one corresponding to a run with workload for month-end, and the other one corresponding to a run with workload for quarter-end.

To apply all in-memory aggregate recommendations that correspond to the month-end run to the content store, open the result from the month-end advisor run and select to apply in-memory recommendations to the cube.

Then, to remove all month-end recommendations and store the quarter-end recommendations in their place, open the results from the quarter-end Aggregate Advisor run and select to apply in-memory recommendations.

Because aggregate cache definitions are retrieved and loaded at cube start, the quarter-end recommendations are the set of in-memory recommendations that are used when the cube is restarted.

Note: Saving in-memory recommendations associates all of the selected in-memory aggregates from the open advisor-run result together as a set to the cube. To overwrite any existing in-memory aggregate definitions with other sets of definitions, open other advisor run results.

Clearing in-memory recommendations from the cube

To clear all in-memory aggregate definitions previously applied to the cube, open any advisor result for that cube and click **File** → **Clear In-Memory Recommendations**.

Figure 11-19 shows selecting a cube to clear its in-memory aggregates.

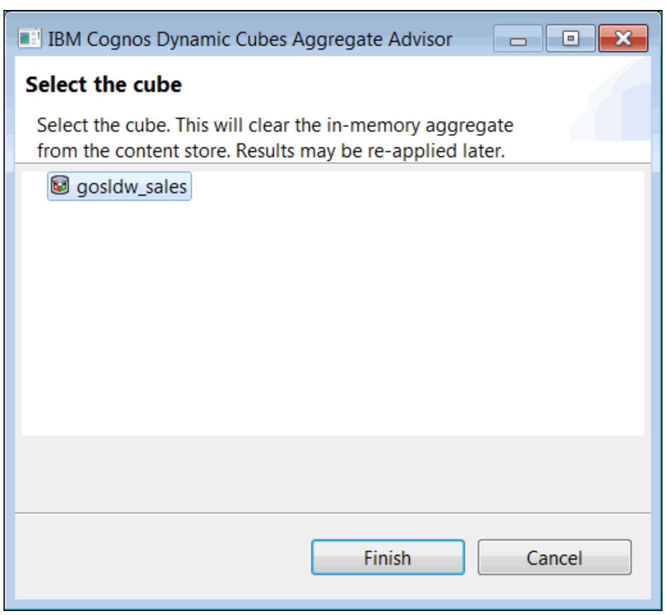


Figure 11-19 Clear in-memory aggregates for the cube gosldw_sales

An example of when to clear the in-memory aggregate recommendations that are associated with the cube is before creating a deployment that includes the Cognos Dynamic Cubes data source and the set of in-memory aggregates are not applicable to the target system.

Deployments that contain Cognos Dynamic Cubes data sources include all associated in-memory aggregate definitions in the content store. If moving a deployment from one scale factor of the data warehouse to another, the same aggregate recommendation might not be applicable. Instead, rerun the Aggregate Advisor against the differently scaled data warehouse to get better recommendations. See 13.3, “Effective practices for deploying between environments” on page 456for more information.

Managing entries in the Advisor Results view

The Advisor Results view contains the results from multiple Aggregate Advisor runs.

Sorting entries in the Advisor Results view

The list of advisor results can be sorted by cube name, by the time stamp, or by the description of the advisor run. By default, the sort order is by time stamp in descending order, such that the most recent advisor result is at the top of the list.

To sort, select the **View** menu of the Advisor Results window, and select **Sort**.

Figure 11-20 shows the Advisor Results **View** menu options.

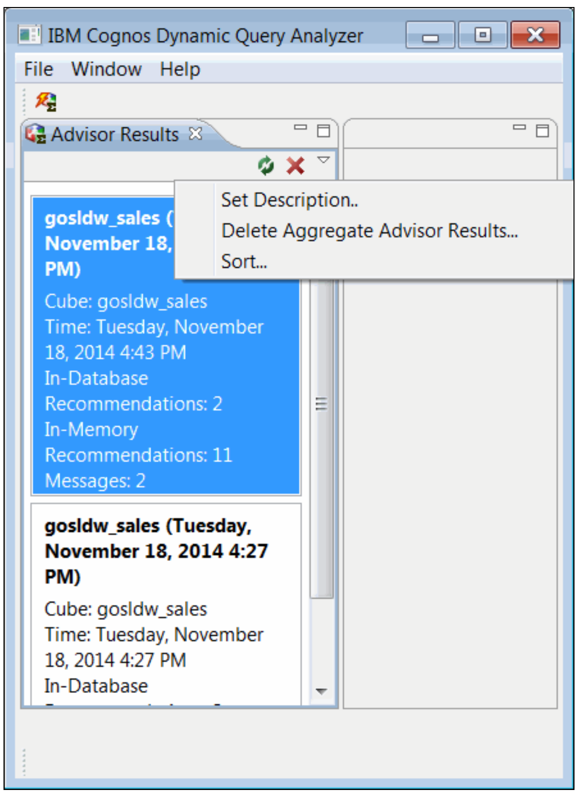


Figure 11-20 Advisor Results View menu options

The Sort the Advisor Results window (Figure 11-21) allows sorting by two criteria and two directions.

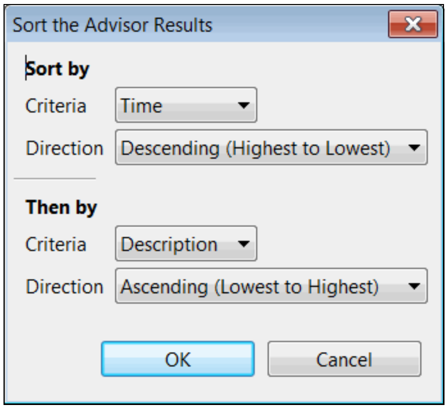


Figure 11-21 Sort the Advisor Results window

Clearing entries from the Advisor Results

Advisor result entries can be cleared from the server by clicking the **X** button. Because these entries are stored on the server, removing them removes them for all users of the specified dispatcher.

You will be asked to confirm the deletion (Figure 11-22).

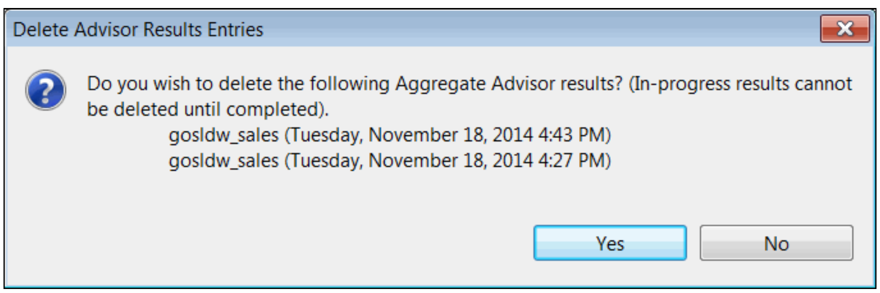


Figure 11-22 Delete confirmation dialog

Setting the description of an advisor result

Advisor result entries can be given a new description by selecting **Set Description** from the drop-down menu. Because these entries are shared with all other users on the server, it can be useful to have them labeled in a more meaningful way (Figure 11-23).

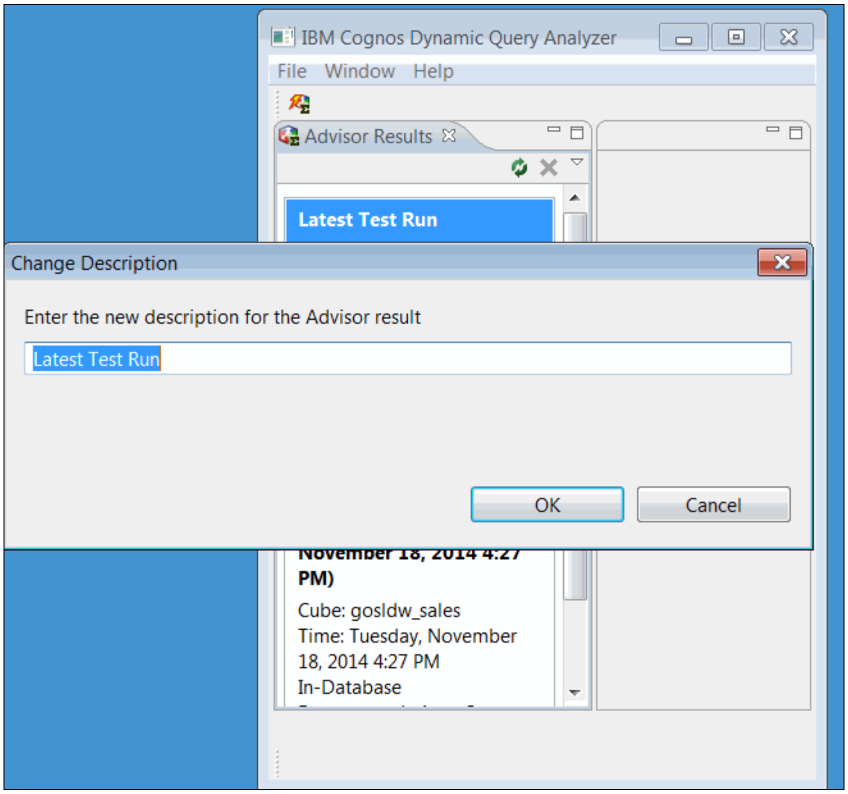


Figure 11-23 Setting the description for an Advisor result

11.3.6 Rerunning the Aggregate Advisor

Note: The Aggregate Advisor can be run multiple times. Use the Advisor Results view to manage various aggregate recommendations based on various run characteristics, criteria, and workloads.

Rerun the Aggregate Advisor if any of the following situations exist:

- If non-trivial changes are made to the cube model or underlying star schema, such as in the following examples:
  - Removal of a dimension, hierarchy, or level that is included in an aggregate that is in use.
  - Addition of a dimension, hierarchy, or level that will be queried against and you want it to be included in aggregates.

– Addition of a measure that will be queried against and you want it to be included in aggregates. Note the following information:
  - Updates for a new measure to in-database aggregates can be made without rerunning the Aggregate Advisor. The DBA can modify the database aggregate creation and refresh scripts to include the measure. Then, using Cognos Cube Designer, the modeler can update the in-database aggregate definition and publish the cube.
  - Updates to in-memory aggregates require rerunning the Aggregate Advisor and saving a different set of in-memory aggregates from the new advisor results.

- Significant data skew changes are made, such as moving from a small-scale development system to a larger quality assurance (QA) or production system.
- Query performance becomes unsatisfactory.
- Workload characteristic changes significantly. In this case, recapture workload logs and rerun the Aggregate Advisor to get more relevant recommendations.

11.4 In-memory aggregates

As described in 2.3, “Cognos Dynamic Cubes caching” on page 48, Cognos Dynamic Cubes supports two types of precomputed aggregate values: Those stored in database tables and those stored in its in-memory aggregate cache.

This section describes how the set of in-memory aggregates that were recommended by the Aggregate Advisor and saved to the content store with the cube in 11.3, “Overview of the Aggregate Advisor” on page 343 are loaded into the aggregate cache and used by the query engine.

11.4.1 Applying in-memory aggregates

In “Results populated in the Advisor Results view” on page 356, the Aggregate Advisor recommended 16 in-memory aggregates with an estimated total size of less than 35 MB. After opening and reviewing the advisor result, the set of in-memory aggregate recommendations were saved to the content store and were associated with the `gosldw_sales` cube.

With the in-memory aggregate definitions saved and associated with the cube, the next time `gosldw_sales` is started or restarted, the in-memory aggregates are loaded if the aggregate cache is not disabled. An aggregate cache size of zero disables the aggregate cache.

11.4.2 Loading in-memory aggregates into the aggregate cache

This section describes how to enable and load the aggregate cache.

Enabling the aggregate cache

The estimated size of the in-memory aggregates by the Aggregate Advisor is an estimate done at the time of the Aggregate Advisor run. The actual amount of memory that is consumed might differ because the Aggregate Advisor is making estimates of the size, and as the data warehouse grows, the amount of memory that is needed to hold the in-memory aggregates increases.

The size of the aggregate cache that is specified in the properties of a dynamic cube is a maximum. The aggregate cache, though, is loaded on a first-come basis. If an aggregate result will not fit into the cache, it is discarded.

Setting the value to a large number does not increase or waste memory. Only enough memory required to hold the defined aggregates is used. For example, if it takes 35 MB to hold the aggregates for `gosldw_sales`, and the aggregate cache size is set to 1 GB, only 35 MB of memory is used. Over time, if the underlying fact tables grow, the aggregates are allowed to grow to the specified maximum of 1 GB.

> **Note:** Set the `Maximum space for in-memory aggregates` cube property to a value that is greater than the advisor-estimated size so that all the in-memory aggregates can be loaded. Remember that, as the data warehouse grows, the amount of memory needed to hold the in-memory aggregates also grows based on a sliding scale that is relative to the size of the member cache.

To verify that the aggregate cache is enabled and has enough memory to load all the in-memory aggregates, set the `Maximum space for in-memory aggregates` value in the cube properties:

1. Go to IBM Cognos Administration.
2. Select **Status**.
3. Select **Dynamic Cubes**.
4. Select **All Server Groups**.
5. Select the server group the server is in.
6. Select the dispatcher.
7. Select the cube.
8. In the **QueryService** menu, select **Set properties**.

9. Enter a value for **Maximum space for in-memory aggregates (MB)**. Specify a value greater than the advisor estimated size so that all the in-memory aggregates can be loaded. For `gosldw_sales`, specifying a value of `200` is more than sufficient for now. See Figure 11-24.

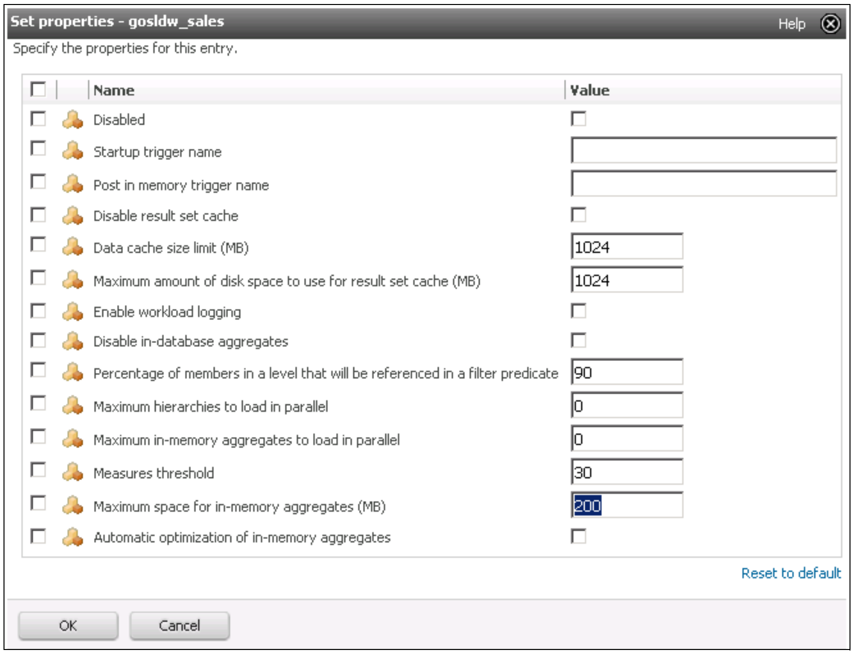


Figure 11-24 Set Maximum space for in-memory aggregates in the cube properties

10. Click **OK**.

Wait several minutes for the updated cube configuration settings to refresh and take effect in the query service. Then, start or restart the cube to initiate loading of the in-memory aggregates.

Loading the aggregate cache

The following cube administrative actions result in the loading of in-memory aggregates into the aggregate cache:

- Cube start or cube restart
- Refresh data cache
- Refresh member cache

After the cube is started and available, the in-memory aggregates are loaded. The queries to load the aggregates are run concurrently and asynchronously. As each individual aggregate finishes loading, it becomes available for use by the query engine.

Underlying database considerations

The DBA should be aware of the aggregate cache-load activities and consider the impact to the underlying relational database. The queries that are run to populate the aggregate cache go against the underlying relational database, and the retrieval and processing of many cells can be resource-intensive for the underlying relational database.

The aggregate load automatically begins after the cube is running and available. The cube is currently open to user queries. User queries can still be processed during the aggregate load. However, until each in-memory aggregate completes its loading, the aggregate cannot be used and query performance will not be optimal. Because the in-memory aggregates can take some time, the cube should be started during, or immediately after, the underlying database extract, transform, and load (ETL) process to allow enough time for the load and provide optimal query performance to users.

These queries are planned by Cognos Dynamic Cubes in the same manner as data queries. Therefore, if there are any in-database aggregate tables available, the engine can take advantage of these database aggregates and can route to them. The engine can also take advantage of any already loaded, existing in-memory aggregates and route to them without requiring a query to the database. See “Using previously loaded in-memory aggregates” on page 371 for more information.

An approach to increase the speed of the aggregate cache load is to have supporting database aggregates. These database aggregates can be created by the DBA, based on specific needs.

To help determine which supporting database aggregates can be used to increase the speed of aggregate cache load, view the information in the database recommendation output. If the Aggregate Advisor was run to include both in-memory and in-database recommendations, information in the in-database recommendation textual output indicates how many corresponding in-memory aggregate recommendations can be satisfied by the database aggregate recommendation. For example, the header section for a logical database aggregate might contain a line similar to the following line:

```
Number of recommended in-memory aggregates that this aggregate can cover: 5
```

Note: The DBA should be aware of the impact to the underlying relational database resources that the aggregate cache load activities can have. That is, the DBA should be aware of the cube administration tasks that result in an aggregate cache load, such as cube start, restart, data cache refresh, and member cache refresh.

Controlling the number of aggregates loaded in parallel

By default, the number of aggregates that are loaded concurrently is determined as being twice the number of processors of the query service system. For example, if the query service system has four processors, then there will be eight threads to load the aggregates in parallel.

The number of queries posed concurrently to populate the in-memory aggregate cache can be controlled by a cube property of Cognos Dynamic Cubes to ensure that the underlying relational database is not saturated with concurrent requests computing summary values.

To reduce the number of aggregate load queries against the underlying database at one time, reduce the value for the `Maximum in-memory aggregates to load in parallel` cube property. Fewer threads require more overall time to load the aggregates.

To set the value for the `Maximum in-memory aggregates to load in parallel` cube property:

1. Go to IBM Cognos Administration.
2. Select **Status**.
3. Select **Dynamic Cubes**.
4. Select **Data Stores (All)**.
5. Select the cube.
6. Select the server group.
7. Select **QueryService - <server URL>**.
8. In the **QueryService** menu, select **Set properties**.

9. Enter a value for the `Maximum in-memory aggregates to load in parallel` property. The default value of 0 means twice the number of processors to the QueryService system.
10. Click **OK**.

If the cube is deployed to multiple server groups and multiple instances of the query service, then each cube instance has its own set of cube properties.

In IBM Cognos Business Intelligence versions before 10.2.2, the query service advanced setting `qsMaxAggregateLoadThreads` was used to control the number of aggregate load threads. This setting is not supported in versions 10.2.2 and later. Use the `Maximum in-memory aggregates to load in parallel` cube property instead.

Using previously loaded in-memory aggregates

In IBM Cognos Business Intelligence 10.2.2 Fix Pack 1, during the loading of in-memory aggregates, whenever possible, Cognos Dynamic Cubes loads an in-memory aggregate from previously loaded in-memory aggregates. This is only supported for measures with associative aggregation rules (SUM, MAX, MIN, and COUNT).

This feature helps to reduce the number of queries posed to the underlying relational database, reducing the load on the database and in most cases, reducing the time required to load a set of in-memory aggregates.

This feature is enabled by default, but can be disabled by adding the advanced property `UseStackedAggregates` to the query service and assigning it a value of `false`.

Monitoring the aggregate cache load

Cube metrics available in Cognos Administration can be used to monitor and indicate when aggregates have completed loading. Select the cube and view the following metrics:

- Loaded/defined in-memory aggregates metric, which shows the number of completely loaded aggregates to the number of defined in-memory aggregates.
- Time spent loading in-memory aggregates metric.
- Used/defined size of in-memory aggregate cache metric, which shows the amount of memory used for loaded in-memory aggregates to the amount of space defined.

Figure 11-25 shows an example of the `gosldw_sales` metrics, related to aggregate cache load.

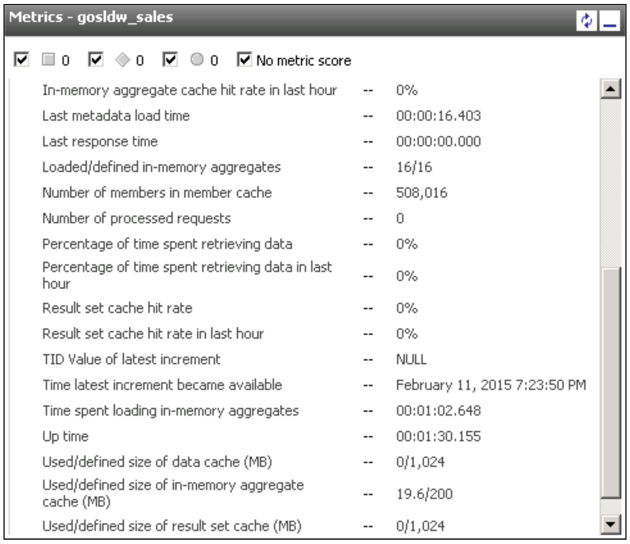


Figure 11-25 Example of the gosldw_sales metrics related to aggregate cache load

Similar information is also available if you select **View recent messages** from the cube menu. The message about the number of aggregates loaded is available after the entire loading process is complete. This message can be used to indicate that the aggregate loading process is completed. For in-progress load information, see the cube Metrics window in Cognos Administration.

Figure 11-26 shows the View recent messages list for `gosldw_sales`, indicating the number of successfully loaded aggregates.

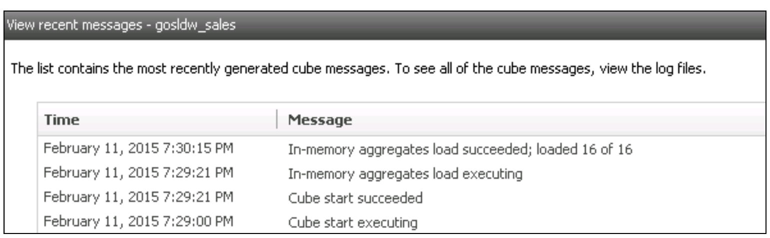


Figure 11-26 Recent messages for gosldw_sales showing number of successfully loaded aggregates

Note: The speed of aggregate cache-loading can be increased by having available supporting database aggregates that can satisfy these aggregate cache-load data queries.

11.4.3 Monitoring aggregate cache hits

Cube metrics that are available in Cognos Administration can be used to monitor the aggregate cache hit rate, along with the hit rates of the result set cache, data cache, and database aggregate tables. Select the cube so that you can see the In-memory aggregate cache hit rate value.

Figure 11-27 is an example of the `gosldw_sales` metrics that shows in-memory aggregate cache hit rate.

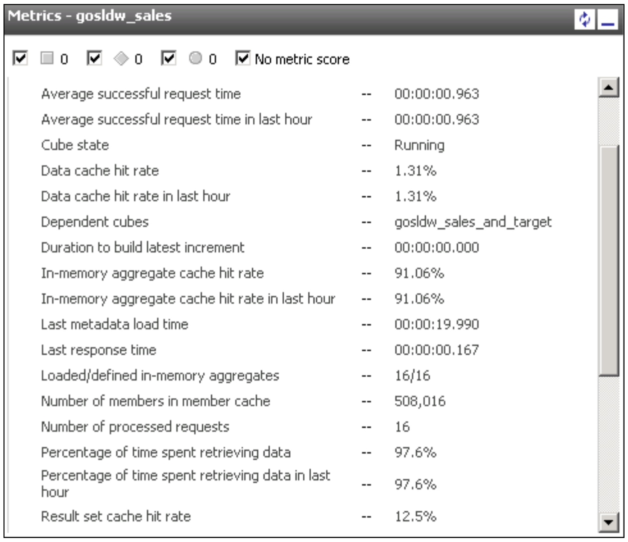


Figure 11-27 Example of the gosldw_sales metrics that shows in-memory aggregate cache hit rate

11.4.4 User-defined in-memory aggregates

User-defined in-memory aggregates provide advanced dynamic cube modelers with the ability to suggest in-memory aggregates to be included in the Aggregate Advisor recommendations.

Use this workflow to load user-defined in-memory aggregates into the cube:

1. Identify measures and aggregation levels that you want to include in an in-memory aggregate.
   - You might be in the modeling phase of your cube development and you know up-front some of the in-memory aggregates you want to have in your cube before the optimization phase. The optimization phase is when you evaluate performance and iteratively refine aggregates by capturing user workload, running the Aggregate Advisor, and applying aggregates recommendations.
   - You might be in the optimization phase of your cube development and have run the Aggregate Advisor and applied those Aggregate Advisor-recommended in-memory aggregates. Now, you want to supplement that set of in-memory aggregates with specific in-memory aggregates to cover additional areas of the cube that the Aggregate Advisor did not recommend. Typically, the Aggregate Advisor does not recommend in-memory aggregates that are large in relation to the size of the fact table.
2. In Cognos Cube Designer, create the user-defined in-memory aggregate definitions in the cube model. See 11.7, “Modeling user-defined in-memory aggregates” on page 407 for information about how to model user-defined in-memory aggregates.
3. Publish the cube to the content store.
4. Run the Aggregate Advisor to generate a recommendation with the user-defined in-memory aggregates.

   You might choose to select the **User Defined Only** option under **Query Workload Information** in the Aggregate Advisor wizard to get recommendations that consists of only user-defined in-memory aggregates assigned in the model. Selecting the other **Query**

Workload Information options results in recommendations that also include user-defined in-memory aggregates, in addition to the system-recommended aggregates.

5. Evaluate the estimated sizes for the user-defined in-memory aggregates and select a set of in-memory aggregates to apply to the dynamic cube for usage. See "Applying in-memory recommendations to the cube" on page 360 for details about this step.
6. Adjust the cube property for `Maximum space for in-memory aggregates` in Cognos Administration, if necessary.
7. Restart the cube to load the new set of in-memory aggregates.

User-defined in-memory aggregates might reduce the time to optimize dynamic cubes. However, the modelers should understand how these aggregates affect the performance and memory utilization of a dynamic cube. Therefore, the workflow to implement user-defined in-memory aggregates still requires running the Aggregate Advisor to obtain size estimates for aggregate recommendations before selecting to apply in-memory aggregates to the cube.

User-defined in-memory aggregates do not have a limit on the size of the dimensional space that they can encompass, so they can span any dimensional space. Unlike system-generated in-memory aggregate candidates that can be rejected based on estimated size during the Aggregate Advisor run, the Aggregate Advisor always includes user-defined in-memory aggregates in the set of recommendations. Therefore, consider the additional memory that these additional user-defined in-memory aggregates will use as part of the cache settings for the cube and the total JVM heap size of the query service.

User-defined in-memory aggregates support all measures except the semi-aggregates measures. Aggregates with additive measures, such as SUM and COUNT, can be used to satisfy queries of the higher level. For example, an aggregate with a SUM measure and Time levels to Quarter can be used to satisfy a query at the Year level because the values from the Quarter level can be added, or rolled up, to get the Year value with this aggregate.

Non-additive measures, such as AVG and STDDEV, can also be included in the user-defined in-memory aggregates. However, these types of measures can only be used if the query is an exact match of the aggregate and the rolling up cannot be done for higher-level queries.

The process to load in-memory aggregates is the same whether they originated as user-defined in the model or system-recommended in the Aggregate Advisor. Therefore, the same considerations when loading in-memory aggregates to query service system resources and underlying relational database system apply:

- Impact to the query service system resources

  By default, the number of aggregates that are loaded concurrently is twice the number of processors of the query service system. If user-defined in-memory aggregates are particularly large, this might result in more large aggregates loading at the same time.

  For more information, see "Controlling the number of aggregates loaded in parallel" on page 370 and "Using previously loaded in-memory aggregates" on page 371.

- Impact to the underlying relational database system

  To alleviate the stress on the database system, one approach is to have supporting database aggregates to aid in loading of in-memory aggregates. The DBA can create database aggregate tables, as needed. The Aggregate Advisor can be used to recommend some database aggregates if the user-defined in-memory aggregate is not too large in relation to the size of the fact table.

  For more information, see "Underlying database considerations" on page 369 and "Using previously loaded in-memory aggregates" on page 371.

- Total elapsed time to load in-memory aggregates

  Loading all in-memory aggregates might require more time due to extra in-memory aggregates and large user-defined in-memory aggregates. Due to fixed ETL window time constraints, it might be necessary to create a subset of these aggregates and take advantage of whatever means necessary to allow the creation of as many aggregates as needed.

  For more information, see “Monitoring the aggregate cache load” on page 371 and “ETL window impacts” on page 423.

Consider these additional important points when using user-defined in-memory aggregates:

- Continue running Aggregate Advisor to obtain other recommendations for defining regular in-memory aggregates.
- Define supplementary user-defined in-memory aggregates to cover the intermediate aggregation levels if user-defined in-memory aggregates are particularly deep in terms of the slice of levels in the hierarchies.

11.4.5 Automatic optimization of in-memory aggregates

The automatic optimization of in-memory aggregates is a feature that enables the system to continually analyze the workload activity and automatically optimize the set of in-memory aggregates in response to the report queries over time. The automatic optimization of in-memory aggregates is a useful feature, particularly in production environments, for minimizing the number of manual Aggregate Advisor runs and reducing the need to purposely generate comprehensive workload logs.

Automatic optimization and adjustment of in-memory aggregates

When you enable this feature on a cube, the query service captures workload activity automatically, and the Aggregate Advisor runs automatically in the background. The Aggregate Advisor quickly analyzes the workload, sometimes recommends new and more effective in-memory aggregates, and applies them to the content store. The query service then automatically adjusts the in-memory aggregates in the running instance of the cube, one in-memory aggregate at a time.

This approach helps to minimize the impact on the live system by loading any newly recommended in-memory aggregates serially, one at a time, as opposed to concurrently as with the in-memory aggregate load at cube start and cube refresh. The live system includes the query service system resources and the underlying database server.

From run to run of the automatic Aggregate Advisor, the automatic optimization tries to recommend fewer numbers of new in-memory aggregates. It does not reload any of the in-memory aggregates that are already loaded and that still exist in the latest recommendation. This approach also helps to reduce the impact of new in-memory aggregates on the live system.

There is no need to manually enable workload logging and capture a comprehensive workload ahead of time. The capturing of workload activity is done automatically when enabling the automatic optimization of in-memory aggregates feature. Be aware of the following differences between the workload log captured as a result of manually enabling workload logging as described in “Enabling workload logging” on page 347 and “Running a representative query workload” on page 348, and the workload log automatically captured as the result of enabling the automatic optimization of in-memory aggregates feature:

- When manually running the Aggregate Advisor against a cube enabled for automatic optimization of in-memory aggregates, the automatic optimization workload log is not subject to the workload log filters in the Aggregate Advisor wizard in DQA. In this case, the Aggregate Advisor will still use all of the information in the automatic optimization workload log, but you will not be able to specify workload filtering options such as report name, time, or users.

- The cube administration settings and actions regarding workload logs in Cognos Administration do not apply to the automatic optimization workload log. For example, the `Clear workload log` cube action does not affect the automatic optimization workload log. The `Clear workload log` cube action only applies to the manually enabled workload log.

The automatic optimization adjusts the set of in-memory aggregates over time, in a conservative manner. For example, the system creates additional in-memory aggregates if it estimates that there is enough memory space. If the space might be exceeded, the system tries to make an intelligent compromise between the previously and newly recommended in-memory aggregates.

The system is especially cautious when it recommends removing aggregates. This approach results in minimal changes to the set of in-memory aggregates, and when the in-memory aggregates are loaded one at a time, minimizes the impact on the system.

The automatic optimization of in-memory aggregates feature is particularly appropriate in the following situations:

- In-database aggregates are either not required in the model, or are stable and do not include slicers. Because this feature optimizes in-memory aggregates only, you need to be aware of the in-memory aggregate load considerations, including approaches that benefit from having in-database aggregates. This feature does not recommend any in-database aggregates. Manually run the Aggregate Advisor for any in-database aggregate recommendations.
- Additive measures are used in the model. Additive measures can roll up values for higher levels of aggregation. Non-additive measures cannot roll up from aggregates and can result in a number of in-memory aggregates to provide direct match for queries.
- There is a prominent ad hoc self-service queries type of activity that requires regular adjustment of in-memory aggregates. This feature can be used as an approach to improve report performance by adjusting the set of in-memory aggregates over time to better match query activity.

To enable automatic optimization of in-memory aggregates, turn on the dynamic cube property `Enable automatic optimization of in-memory aggregates`, and set the dynamic cube property `Maximum space for in-memory aggregates` to a value greater than 0.

Additional considerations when enabling automatic optimization of in-memory aggregates:

- If you are using multiple dispatchers for the query service, only the server with the dynamic cube property enabled has its in-memory aggregates automatically and continually optimized. The cube on other dispatchers synchronizes with the content store and loads new in-memory aggregates only when it restarts.
- If you have configured reports to run based on the post in-memory trigger, as described in 11.8.3, “Scheduling priming reports to run by trigger” on page 411, this trigger will not be invoked after the incremental in-memory aggregate load of automatic optimization. This post in-memory trigger will be invoked only after the complete set of in-memory aggregate loads for cube start or refresh.

Automatic optimization and referential integrity

As described in “Incorrect data values” on page 381, problems with referential integrity can affect the quality of the data in summary tables, in-memory aggregates, and data caches. For more information about how to run a series of SQL queries to identify any referential integrity issues that could cause data discrepancies in aggregates, see 6.6, “Data quality and integrity” on page 185.

When using in-memory aggregates, it is common practice to run the Aggregate Advisor, apply the in-memory aggregate recommendations to the content store, and then test the overall application for performance and data validity. Without running the previously mentioned set of SQL queries to ensure the validity of the data in the data warehouse, a newly introduced in-memory aggregate could manifest such a discrepancy by containing summarized data values that are less than the aggregate of all the data in the fact table.

The in-memory aggregates from the automatic optimization behave the same as regular in-memory aggregates and are affected in the same manner by referential integrity errors. However, because the in-memory aggregates are created automatically while a cube is running, there is no opportunity for validation before their use by a dynamic cube. Hence, when using the automatic optimization of in-memory aggregates feature, ensure the referential integrity of the data warehouse before enabling the feature.

Another approach to ensure the consistency of the data without doing a referential integrity check is to create, for each dimension, a dimension filter that expresses the equality between the key of a dimension and its corresponding key in the fact table. The downside of this approach is that it forces a join between each dimension and the fact table, thus affecting the performance of all queries that are posed to the database.

This guidance is valid as well for both in-database aggregate tables and regular in-memory aggregates, though in these cases there is an opportunity for validation before employing the aggregates in a production environment.

Advanced settings to configure automatic optimization

There are situations where configuring automatic optimization of in-memory aggregates can help you meet specific requirements:

- Reduce time and impact of loading in-memory aggregates on the underlying database

  By default, Aggregate Advisor recommends in-memory aggregates that are based only on the workload, which means that the in-memory aggregates are loaded either from the in-database aggregates or from the fact table. Loading an in-memory aggregate that is based on a large fact table takes a long time.

  Load performance is improved if the in-memory aggregates load from in-database aggregates that are smaller than the fact table. To ensure that your in-memory aggregates load from in-database aggregates, use the query service advanced setting, `qsAutomaticAggregateOptimizationMatchInDatabaseAggregates` and set it to `True`.

  As a result, Aggregate Advisor recommends only in-memory aggregates that match the in-database aggregates. Because the in-memory aggregates do not have slicers, Aggregate Advisor ignores any in-database aggregates with slicers when assessing whether an in-memory aggregate would match. Note that even with this advanced setting, if you have any user-defined in-memory aggregates that are not supported by in-database aggregates, the Aggregate Advisor will still recommend them because user-defined in-memory aggregates are always included in the set.

  This advanced setting is best used for larger cubes where the impact of loading in-memory aggregates that do not have supporting database aggregates is much greater than for smaller cubes.

- Control when the automatic optimization of in-memory aggregates for cubes occur

  By default, the system determines when to run Aggregate Advisor and load in-memory aggregates. Use the query service advanced setting, qsAutomaticAggregateOptimizationStartTime, if you prefer to start this activity at a specific time. The value is based on a 24-hour clock, where valid values are 00:00 to 23:59. For example, if you specify 23:00, the automatic optimization of the dynamic cubes on the server occurs nightly, starting at 11:00 PM.

By default, the system performs automatic optimization of one cube at a time. For example, if there are three cubes on a server that is enabled for automatic optimization of in-memory aggregates, the system automatically runs Aggregate Advisor and loads any recommended aggregates for the first cube. After it is complete, this action is repeated for the second cube, and then for the third cube. This type of processing minimizes the load on the query service and database servers.

You can change this behaviorby using the query service advanced setting, qsAutomaticAggregateOptimizationMaxConcurrentCubeTasks, to specify the number of cubes to be optimized concurrently. Use a positive integer starting with 1 for the value. Note that this advanced setting does not require a query service restart, whereas the others mentioned in this section do require a query service restart to take effect.

This setting is typically used when the cubes are also configured for automatic optimization at a specified time (that is, the qsAutomaticAggregateOptimizationStartTime advanced setting is configured to a time), preferably during a maintenance window when the system is lightly used.

However, if optimization occurs throughout the day, which is the default behavior (that is, the qsAutomaticAggregateOptimizationStartTime setting is configured to use the default value or empty string), you must be cautious about changing the qsAutomaticAggregateOptimizationMaxConcurrentCubeTasks advanced setting.

Troubleshooting automatic optimization of in-memory aggregates

For general in-memory aggregate troubleshooting, see 11.4.6, "In-memory aggregate tips and troubleshooting" on page 379.

For information about issues with incorrect data values, see "Automatic optimization and referential integrity" on page 376.

As a first step to troubleshooting automatic optimization issues, see the automatic aggregate optimization log file, located in the same directory as the query service log file, relative to the server installation directory: logs\XQE\automatic_aggregate_optimization_log-<timestamp>.xml. This trace log is enabled by default and provides high level information about what the automatic run of the Aggregate Advisor has done, what occurred on the system, and when.

Although it can seem that with the automatic optimization of in-memory aggregates feature enabled you do not need DQA installed, there are several cases where you can still require using DQA and even manually run the Aggregate Advisor.

The automatic optimization is for in-memory aggregates only. If the in-memory aggregates take too long to load, one approach is to have supporting database aggregates. Run the Aggregate Advisor in DQA to get some in-database aggregate recommendations.

To see the current set of in-memory aggregates that are applied to the automatically optimized cube, use DQA to retrieve and view history of runs in the Advisor Results list view. All types of Aggregate Advisor results, whether they are from a manual or automatic run, are stored on the server.

Only the latest automatic run result per cube is kept. Be sure to configure DQA to refer to the server that has the running instance of the cube with automatic optimization of in-memory aggregates enabled. See "Setting preferences to Cognos server" on page 345 for more information about how to configure DQA. Figure 11-28 shows an example of what an automatically applied advisor result looks like in the Advisor Results list view.

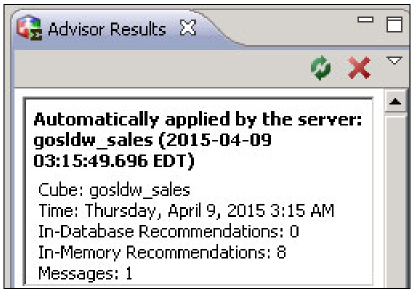


Figure 11-28 Advisor Results showing an entry for an automatically applied advisor result

11.4.6 In-memory aggregate tips and troubleshooting

This section provides tips and troubleshooting information for in-memory aggregates.

System considerations

You must consider several cube and system settings when you use in-memory aggregates.

Aggregate cache size

You can find hardware sizing and guidelines for the amount of memory to use for the in-memory aggregate cache of a cube in *IBM Business Analytics Proven Practices: Dynamic Cubes Hardware Sizing Recommendations*, which you can find in the IBM Business Analytics Proven Practices website at:

http://ibm.biz/DynamicCubesHWSizingRec

Alternatively, see 17.2, “Using the hardware sizing calculator in IBM Cognos Cube Designer” on page 534.

The size of the aggregate cache that is specified in the properties of a dynamic cube is simply a maximum. Setting the value to a large number does not increase or waste memory. Only the amount of memory that is required to hold the defined aggregates is used. For example, if 90 MB can hold the aggregates for `gosldw_sales`, and the aggregate cache size is set to 1 GB, only 90 MB of memory is used. Over time, if the underlying fact tables grow, the aggregates are allowed to grow to the specified maximum of 1 GB.

You should not use more than 30 GB for the aggregate cache. Although the query service system might have enough physical memory, the number of objects that are created by the large in-memory aggregates can affect the system resources and result in overall degraded performance.

Data cache size

The data cache is still used, even with the presence of in-memory aggregates. If a query is at a higher level of aggregation than the in-memory aggregate, the values in the in-memory aggregate are used to calculate the requested value. After the calculation is done by using the in-memory aggregate values, it is then stored in the data cache for future use. Therefore, you should provide a sufficient amount of memory for the data cache, even when using the in-memory aggregate cache.

JVM considerations

The default query service JVM settings for the JVM garbage collection policy, initial JVM nursery size, and JVM nursery size limit allow the system to determine settings based on the

JVM heap size limit. In most cases, these default settings generally allow acceptable performance. If you are a Java tuning expert and choose to adjust any of these settings, be aware of the following JVM considerations concerning Cognos Dynamic Cubes usage of memory:

- The Java heap is split into two areas, a new (or nursery) area and an old (or tenured) area. Objects are created in the nursery area first, and if the objects continue to be referenced for a period of time, they are considered long-lived and are moved into the tenured area.
- By default, the query service JVM garbage collection policy is `Generational`. A generational garbage collection policy is suited for applications with usage of many short-lived objects in memory.
- The memory allocation pattern of Cognos Dynamic Cubes changes over time. Cognos Dynamic Cubes preload the member and aggregate caches when the cube is started. These caches contain large, long-lived objects. So, during the load or refresh phase of a cube, it is preferable to have a smaller nursery to reduce the likelihood of long nursery collection pauses. Then, after the cube is loaded and user report queries are run, it is preferable to have a larger nursery space because the objects created during a query are short-lived.
- The initial JVM nursery size is zero, by default, allowing the system to calculate the initial nursery size as a proportion of the initial heap size. The calculated proportion is intended for better performance during and immediately after activities that create long-lived objects, such as the loading of cube data into the caches. Generally, keep the default setting.
- The JVM nursery size limit is set to zero, by default, indicating that the system will calculate the maximum nursery size. You can choose to set the JVM nursery size limit to a different value, given your expected amount of caching, reporting style, and user query concurrency requirements. The idea is for all the temporary objects created during report execution to use nursery space so that they are not promoted to tenured space and take advantage of the nursery's collection policy. Adjust the default setting if needed.

  Alternatively, you can set the initial tenured space size and maximum tenured size using the `-Xmos` and `-Xmox` settings in the additional JVM arguments for the query service. If you adjust these settings, be sure to have a tenured space large enough to fit all the long-lived caches, plus a 10-20% buffer.

For more information about IBM SDK Java, Generational Concurrent Garbage Collector, nursery, and tenured areas of the Java heap, see the Diagnostic guide for IBM SDK, Java Technology Edition, Version 6 at:

http://www.ibm.com/support/knowledgecenter/SSYKE2_6.0.0/com.ibm.java.doc.diagnostics.60/diag/understanding/mm_gc_generational.html

Underlying database resources

See 11.4.2, "Loading in-memory aggregates into the aggregate cache" on page 367 for the underlying database considerations that the DBA should be aware of relative to database resources when in-memory aggregates are loaded.

Troubleshooting

This section describes several common in-memory aggregates issues that you might encounter, how to identify the cause, and how to resolve the issue.

In-memory aggregates are not loading

The most common reason why in-memory aggregates are not loaded is that the cube configuration property for the amount of memory to use for the aggregate cache is set to zero, which disables the aggregate cache. In this case, a message that describes the situation will be available in the View recent messages window for the cube.

Figure 11-29 shows the dialog and why in-memory aggregates were not loaded.

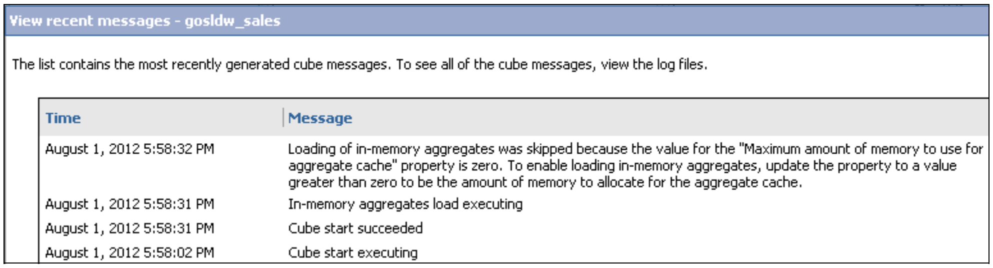


Figure 11-29 View recent messages for gosldw_sales - In-memory aggregates were not loaded

Another common reason why the aggregates are not loading is that the cube configuration was updated or the in-memory aggregates were saved immediately before beginning a cube start or restart. In this case, wait a few minutes for the updates to the Content Store (that is, the cube configuration and the in-memory aggregate definitions) to be refreshed and available to the query service before attempting to start the cube.

Incorrect data values

The first step in troubleshooting a potential in-memory aggregate query problem is to determine whether the problem is because of the aggregates. Disable the aggregate cache and rerun the scenario without the in-memory aggregates to see if the correct data values are returned. To disable the aggregate cache, set the `Maximum space for in-memory aggregates` property in the cube configuration to `0` (zero). Restart the cube so that the updated setting takes effect. Although in-memory aggregates can be defined and associated with the cube, they will not be loaded.

If the values that are calculated from the in-memory aggregates are smaller than the aggregation of the actual underlying database values, it might be a sign of referential integrity problems in the data. For more information, see 6.6, “Data quality and integrity” on page 185 or see the *Cognos Dynamic Cubes User Guide,* v10.2.2 at the following website:

http://www-01.ibm.com/support/knowledgecenter/SSEP7J_10.2.2/com.ibm.swg.ba.cognos.ug_cog_rlp.10.2.2.doc/c_cubingconcepts.html?lang=en

In the *Cognos Dynamic Cubes User Guide*, see the section about Referential integrity in data warehouses in the Cognos Dynamic Cubes overview topic. It describes a set of queries to run to identify when referential integrity issues occur.

Using the query service log to understand in-memory aggregate processing

This section describes advanced troubleshooting techniques by using more verbose logging levels and examining the query servicelog.

To generate some insight about what in-memory aggregates are selected and what happens when processing a query against an in-memory aggregate, enable the logging event group specifically for the aggregate cache-related activity:

1. On the query service system, edit the DQM logging configuration, which is located relative to the server installation directory:

   ```
   configuration\xqe.diagnosticlogging.xml
   ```

2. Edit the aggregate cache event group to the log level of `info`:

   ```
   <eventGroup name="ROLAPCubes.AggregateCache" level="info"/>
   ```

3. Edit the query performance event group to the log level of `info`:

   ```
   <eventGroup name="ROLAPQuery.Performance" level="info"/>
   ```

4. Save the `xqe.diagnosticlogging.xml` file.
5. You do not need to restart the query service for the logging level changes to take effect. Wait for the logging refresh interval, default of which is 30 seconds.
6. Start the cube.
7. After the in-memory aggregates complete loading, run the problem query.
8. By default, the query servicelog is written to a file, which is located relative to the server installation directory:

   ```
   logs\XQE\xqelog-<timestamp>.xml
   ```

   When troubleshooting by using the query service log is complete, revert the changes to the log levels in the `xqe.diagnosticlogging.xml` file.

Open the query service log to verify that the following steps occur when processing a query in the aggregate cache:

- Score in-memory aggregates and select one.

  The requested members of the query are inspected to see whether there is an in-memory aggregate that can be used to solve the query. Each aggregate is internally scored based on how close to the user-requested query it is, and the aggregate with the best score is selected. The closer to the user-requested query the aggregate is, the better the score.

  For example, if the query is for `[Retailers].[Americas]`, then an aggregate on `[Retailers].[Retailer Country]` will score better than an aggregate on `[Retailers].[Retailer name]`. This log entry is shown in Example 11-1.

  Example 11-1 Log entry

  ```
  <event ...><![CDATA[Aggregate [aggregate_memory_1] selected  with cost
  85596.0]]></event>
  ```

- Determine whether the query is a direct hit for the selected aggregate.

  If the levels in the query match all the levels in the aggregate, it is considered a direct hit and it is routed directly to the aggregate cache. A direct hit has a score cost of 1.0.

- Fetch values from the selected aggregate for rollups if not a direct hit.

  If the query is not a direct hit, then rollups are needed and processing continues. The query is inspected to see what values are needed from the aggregate. A portion of the query is issued against the aggregate to fetch the needed values.

  For example, if the query is for `[Retailers].[Americas]` and the aggregate is on `[Retailers].[Retailer Country]`, then the aggregate query will request values for every country in the Americas. The fetch time is also recorded because many values might be needed.

  Example 11-2 shows the log entry.

  Example 11-2 Log entry

  ```
  <event ...><![CDATA[Fetched 24,088 tuples in 549ms.]]></event>
  ```

- Process the aggregate values in multiple threads.

  After the values are fetched from the aggregate, they are inspected to determine how many values there are and how much work processing might be. This step determines whether the processing is split into multiple subtasks that will execute in parallel. The number of available threads is also accounted for, so a busy system will result in splitting the processing into fewer subtasks.

Also, a calculation modifier value accounts for how many locations each aggregate cell belongs to. Consider an example query that requests `[Retailers].[Americas]`, `[Retailers].[Americas].[United States]` and `[Retailers].[Americas].[United States].[Outdoor Gear Co-op]`. The value for `[Outdoor Gear Co-op]` can be in, or contribute to, three locations: itself, as part of `[United States]`, and part of `[Americas]`. More locations mean more work and a larger calculation modifier.

Example 11-3 shows the log entry.

Example 11-3 Log entry

```
<event ...><![CDATA[Using 4 tasks to process aggregate.  Total number of
aggregate cells accessed: 24088 with a calculation modifier of 4.0]]></event>
```

- Combine values as each processing thread completes.

  Each subtask works on its set of values independently. Then, as these subtasks finish processing, their results are combined to form the final result set.

  For example, if the query is for `[Retailers].[Americas]` and the aggregate on `[Retailers].[Retailer name]`, there can be a subtask for each country, rolling up the values for all retailers from each country independently. Then, the original task rolls up the country values together.

  An entry for each subtask is shown in the log entry in Example 11-4.

Example 11-4 Log entry

```
<event ...><![CDATA[Aggregate calc thread finished.  Processed 6,022 cells.
Rollup time : 162 ms into 72 cells with 0 cells in the overfetchResultSet.
Result combination time 2ms, representing 1 calc threads.   Cell cache hit
rate: 99% 99% 99% 99% 99% 99% 99% 99%]]></event>
```

- Query processing is complete.

  After all the subtask results are combined, the final result set is returned. The log record in Example 11-5 shows the end of this processing and captures some metrics.

Example 11-5 Log record

```
<event ...><![CDATA[Finished execution of Query Strategy for report unknown
   Requested 72 tuples
   Found 0 tuples in data cache
   Found 0 tuples in aggregate cubes
   Fetched 0 tuples from database
   0 of the tuples were from pushdown queries
   Created 72 tuples from the aggregate cache
      Total time processing aggregates : 774 (ms) using 4 tasks
            24088 aggregate cells rolled up into 8 separate locations
```

If the query was not a direct hit and involved rolling up values, then those rolled up values are put into the data cache asynchronously. Direct hit values that did not require rolling up are not put into the data cache to prevent duplicating values in the caches and save memory usage.

Using the query service log to troubleshoot incorrect data values from in-memory aggregate processing

This section describes advanced troubleshooting techniques using trace logging levels and examining the query service log.

First, see the following sections to determine whether the incorrect values are the result of the in-memory aggregates:

- "Incorrect data values" on page 381
- "Using the query service log to understand in-memory aggregate processing" on page 381

Next, if they are, use informational log entries to determine whether the appropriate aggregate was selected.

Finally, if more information is needed to assess the issue, enable the logging event group specifically for the aggregate cache-related activity and increase the log level to trace:

1. On the query service system, edit the DQM logging configuration, which is located relative to the server installation directory:

   ```
   configuration\xqe.diagnosticlogging.xml
   ```

2. Edit the aggregate cache event group to the log level of `trace`:

   ```
   <eventGroup name="ROLAPCubes.AggregateCache" level="trace"/>
   ```

3. Edit the query performance event group to the log level of `info`:

   ```
   <eventGroup name="ROLAPQuery.Performance" level="info"/>
   ```

4. Save the `xqe.diagnosticlogging.xml` file.
5. It is not required to restart the query service for the logging level changes to take effect. Wait for the logging refresh interval, which by default is 30 seconds.
6. Start the cube.
7. After the in-memory aggregates complete loading, run the problem query.
8. By default, the query service log is written to a file that is in a location relative to the server installation directory:

   ```
   logs\XQE\xqelog-<timestamp>.xml
   ```

After the problem query has been captured, revert the changes to the log levels in the `xqe.diagnosticlogging.xml` file.

Keeping the aggregate cache trace-level logging on for longer than what is required can negatively affect performance. Because the trace log level will generate a lot of data to the query service log, it should be enabled only for the problem query, and troubleshooting be done when there is little to no other user activity.

The trace log level for the aggregate cache will dump the intermediate calculations from each of the subtask threads to the log. This information can be used to determine the source of the incorrect values.

> **Note:** Do not keep aggregate cache trace-level logging enabled longer than what is required for troubleshooting. The trace-level log will generate a lot of log information and can negatively affect performance.

11.5 Database aggregates

As described in 11.2, “Aggregate awareness in Cognos Dynamic Cubes” on page 342, database aggregates can either be built-in relational database system constructs, or be regular tables that hold aggregated data. Both types can be modeled into the cube, and Cognos Dynamic Cubes can ensure routing to them.

For warehouses that do not yet have aggregates, or want to supplement existing database aggregates with in-memory and other in-database aggregates, run the Aggregate Advisor as described in 11.3, “Overview of the Aggregate Advisor” on page 343 to get aggregate recommendations.

This section describes taking the in-database recommendations from the Aggregate Advisor, giving them to the DBA for creation, and other considerations.

11.5.1 In-database aggregate recommendations

In 11.3, “Overview of the Aggregate Advisor” on page 343, the Aggregate Advisor recommended two in-database aggregates for the `gosldw_sales` cube. After opening and reviewing the advisor result, the in-database output text file was saved.

The in-database recommendations output file describes each aggregate in terms of what it logically contains, and other relevant information in a header section. This file can be given to the DBA and modeler as the basis for creating the database aggregates and modeling their support in the cube. The remainder of this section describes and has an example of the header section that is available for each aggregate.

Aggregate name

Each aggregate is given a name based on the levels included:

```
* Aggregate: Branch - Order method - Promotion
```

List of dimensions, hierarchies, and level of aggregation

Example 11-6 shows the dimension hierarchies with the level at which they are aggregated.

Example 11-6 Dimensions, hierarchies, and level of aggregation

```
* Dimension             Hierarchy             Level
* ---------------       ---------------       ---------------
* Branch                Branch                Branch
* Employee by region    Employee by region    [All]
* Order method          Order method          Order method
* Product brand         Product brand         [All]
* Products              Products              [All]
* Promotions            Promotions            Promotion
* Retailers             Retailers             [All]
* Time                  Time                  [All]
* Time (close date)     Time (close date)     [All]
* Time (ship date)      Time (ship date)      [All]
```

List of measures

Example 11-7 shows the list of measures.

Example 11-7 Measures

```
* Measures:
* ---------------
* Quantity
* Revenue
* Gross profit
```

Description of the columns that must be created for the aggregate table

Example 11-8 shows the columns and data types for the aggregate table.

Example 11-8 Description of columns

```
* Column              Data Type
* ---------------     ---------------
* Region_code         INTEGER
* Country_code        INTEGER
* Branch_key          INTEGER
* Order_method_key    INTEGER
* Campaign_code       INTEGER
* Promotion_key       INTEGER
* Quantity            BIGINT
* Revenue             DECIMAL(38,2)
* Gross_profit        DECIMAL(38,2)
```

If the data type of the column is unknown, see the definition of the column in the example of the SQL(“SQL example to illustrate how to aggregate the data” on page 387) and the documentation for the database to determine the precise data type.

List of other database aggregates from which the aggregate can be derived

This information can be used by the DBA to create stacked aggregates, or aggregates that can be derived from another recommended database aggregate:

```
* This aggregate can be derived from any of the following other aggregates for the database:
* Branch - Order method - Product name - Product - Promotion
* Branch - Order method - Product name - Product - Promotion - Retailer Site
```

The example SQL described in “SQL example to illustrate how to aggregate the data” on page 387 illustrates how to aggregate the data. It is based off the derivable database aggregate, not the underlying warehouse tables.

Number of recommended in-memory aggregates from the same, corresponding advisor result that the aggregate can cover

This information can be used to identify database aggregates that can accelerate the loading of in-memory aggregates because they can satisfy this number of in-memory aggregates, if the in-memory aggregates from the same advisor result are used.

Database aggregates that cover zero in-memory aggregates can still provide performance benefits because queries against aggregate tables are faster than those against the underlying warehouse tables:

```
* Number of recommended in-memory aggregates that this aggregate can cover: 0
```

SQL example to illustrate how to aggregate the data

Example 11-9 is *not* an executable SQL script to create database aggregate tables. It is an SQL example that can aggregate the data and be used as guidance. The DBA can take this information and choose which aggregates to actually create. The DBA might even decide to modify the SQL to create a subset of the aggregate to contain only a certain subset of members instead of including the entire level, such as specific years from the Time dimension.

Example 11-9 SQL that illustrates how to aggregate the data

```
SELECT
    "GO_REGION_DIM"."REGION_CODE" AS "Region_code",
    "GO_BRANCH_DIM"."COUNTRY_CODE" AS "Country_code",
    "GO_BRANCH_DIM"."BRANCH_KEY" AS "Branch_key",
    "SLS_ORDER_METHOD_DIM"."ORDER_METHOD_KEY" AS
        "Order_method_key",
    "MRK_CAMPAIGN_LOOKUP"."CAMPAIGN_CODE" AS "Campaign_code",
    "MRK_PROMOTION_DIM"."PROMOTION_KEY" AS "Promotion_key",
    SUM("SLS_SALES_FACT"."QUANTITY") AS "Quantity",
    SUM("SLS_SALES_FACT"."SALE_TOTAL") AS "Revenue",
    SUM("SLS_SALES_FACT"."GROSS_PROFIT") AS "Gross_profit"
FROM
    "GOSALESDW"."EMP_EMPLOYEE_DIM" "EMP_EMPLOYEE_DIM"
        INNER JOIN "GOSALESDW"."GO_BRANCH_DIM" "GO_BRANCH_DIM"
        ON "EMP_EMPLOYEE_DIM"."BRANCH_CODE" = "GO_BRANCH_DIM"."BRANCH_CODE"
            INNER JOIN "GOSALESDW"."SLS_SALES_FACT" "SLS_SALES_FACT"
            ON "EMP_EMPLOYEE_DIM"."EMPLOYEE_KEY" = "SLS_SALES_FACT"."EMPLOYEE_KEY"
                INNER JOIN "GOSALESDW"."GO_REGION_DIM" "GO_REGION_DIM"
                ON "GO_REGION_DIM"."COUNTRY_CODE" = "GO_BRANCH_DIM"."COUNTRY_CODE"
                    INNER JOIN "GOSALESDW"."MRK_PROMOTION_DIM" "MRK_PROMOTION_DIM"
                    ON "MRK_PROMOTION_DIM"."PROMOTION_KEY" =
"SLS_SALES_FACT"."PROMOTION_KEY"
                        INNER JOIN "GOSALESDW"."SLS_ORDER_METHOD_DIM"
"SLS_ORDER_METHOD_DIM"
                        ON "SLS_ORDER_METHOD_DIM"."ORDER_METHOD_KEY" =
"SLS_SALES_FACT"."ORDER_METHOD_KEY"
                            INNER JOIN "GOSALESDW"."MRK_CAMPAIGN_LOOKUP"
"MRK_CAMPAIGN_LOOKUP"
                            ON "MRK_PROMOTION_DIM"."CAMPAIGN_CODE" =
"MRK_CAMPAIGN_LOOKUP"."CAMPAIGN_CODE"
GROUP BY
    "GO_REGION_DIM"."REGION_CODE",
    "GO_BRANCH_DIM"."COUNTRY_CODE",
    "GO_BRANCH_DIM"."BRANCH_KEY",
    "SLS_ORDER_METHOD_DIM"."ORDER_METHOD_KEY",
    "MRK_CAMPAIGN_LOOKUP"."CAMPAIGN_CODE",
    "MRK_PROMOTION_DIM"."PROMOTION_KEY"
```

Example 11-10 is an example of an SQL statement showing how to aggregate data if the database aggregate can be derived from another recommended database aggregate. Notice that the example SQL statement references the aliased column names of the derivable recommended database aggregate. It does not reference the underlying warehouse tables and columns. If the database aggregate can be derived from more than one recommended database aggregate, then the one with the smaller estimated row count is referenced in the example SQL statement. When the DBAs implement this database aggregate, they update the placeholder in the FROM clause and any other corresponding changes, to reflect the actual derivable database aggregate table and column names.

Example 11-10 SQL that aggregates the data based on another database aggregate

```
SELECT
    "Region_code",
    "Country_code",
    "Branch_key",
    "Order_method_key",
    "Campaign_code",
    "Promotion_key",
    SUM("Quantity") AS "Quantity",
    SUM("Revenue") AS "Revenue",
    SUM("Gross_profit") AS "Gross_profit"
FROM
    [Branch - Order method - Product name - Product - Promotion]
GROUP BY
    "Region_code",
    "Country_code",
    "Branch_key",
    "Order_method_key",
    "Campaign_code",
    "Promotion_key"
```

11.5.2 Creating recommended database aggregates

The DBA has the flexibility to choose which in-database aggregate recommendations to use and how to implement them. The recommendations are intended to be easy to use and have relevant guidance based on the Aggregate Advisor workload and cube model analysis. Any changes that the DBA makes to the actual database aggregates that are created should be communicated to the modeler so that the database aggregates are properly modeled into the cube. For more information about how to use Cognos Cube Designer to include database aggregates in the model to enable the cube so it is aware of the database aggregate, see 11.6, “Modeling in-database aggregates” on page 394.

Many approaches to create aggregate tables exist. Each differs across various relational database vendors. Example 11-11 on page 389 shows how, in a simple manner, the information and example SQL from the recommendation file can be used to create an aggregate table and initially populate a database aggregate table.

This example does not describe any of the relational database considerations that are involved in creating and inserting into aggregate tables, which a DBA will likely do. These considerations include tablespaces, database transaction log management, isolation levels, indexes, use of built-in database optimizer, and aggregate support. It might be necessary to split the insert load query into multiple queries that are partitioned based on one or more dimensions to avoid temporary table, buffer, or spool space exhaustion.

Example 11-11 Aggregate table

```
CREATE TABLE "GOSALESDW"."AGGR_BRANCH_ORDERMETHOD_PROMO" (
  Region_code         INTEGER,
  Country_code        INTEGER,
  Branch_key          INTEGER,
  Order_method_key    INTEGER,
  Campaign_code       INTEGER,
  Promotion_key       INTEGER,
  Quantity            BIGINT,
  Revenue             DECIMAL(38,2),
  Gross_profit        DECIMAL(38,2)
);
INSERT INTO "GOSALESDW"."AGGR_BRANCH_ORDERMETHOD_PROMO"
SELECT
    "GO_REGION_DIM"."REGION_CODE" AS "Region_code",
    "GO_BRANCH_DIM"."COUNTRY_CODE" AS "Country_code",
    "GO_BRANCH_DIM"."BRANCH_KEY" AS "Branch_key",
    "SLS_ORDER_METHOD_DIM"."ORDER_METHOD_KEY" AS
        "Order_method_key",
    "MRK_CAMPAIGN_LOOKUP"."CAMPAIGN_CODE" AS "Campaign_code",
    "MRK_PROMOTION_DIM"."PROMOTION_KEY" AS "Promotion_key",
    SUM("SLS_SALES_FACT"."QUANTITY") AS "Quantity",
    SUM("SLS_SALES_FACT"."SALE_TOTAL") AS "Revenue",
    SUM("SLS_SALES_FACT"."GROSS_PROFIT") AS "Gross_profit"
FROM
    "GOSALESDW"."EMP_EMPLOYEE_DIM" "EMP_EMPLOYEE_DIM"
        INNER JOIN "GOSALESDW"."GO_BRANCH_DIM" "GO_BRANCH_DIM"
        ON "EMP_EMPLOYEE_DIM"."BRANCH_CODE" = "GO_BRANCH_DIM"."BRANCH_CODE"
            INNER JOIN "GOSALESDW"."SLS_SALES_FACT" "SLS_SALES_FACT"
            ON "EMP_EMPLOYEE_DIM"."EMPLOYEE_KEY" = "SLS_SALES_FACT"."EMPLOYEE_KEY"
                INNER JOIN "GOSALESDW"."GO_REGION_DIM" "GO_REGION_DIM"
                ON "GO_REGION_DIM"."COUNTRY_CODE" = "GO_BRANCH_DIM"."COUNTRY_CODE"
                    INNER JOIN "GOSALESDW"."MRK_PROMOTION_DIM" "MRK_PROMOTION_DIM"
                    ON "MRK_PROMOTION_DIM"."PROMOTION_KEY" =
"SLS_SALES_FACT"."PROMOTION_KEY"
                        INNER JOIN "GOSALESDW"."SLS_ORDER_METHOD_DIM"
"SLS_ORDER_METHOD_DIM"
                        ON "SLS_ORDER_METHOD_DIM"."ORDER_METHOD_KEY" =
"SLS_SALES_FACT"."ORDER_METHOD_KEY"
                            INNER JOIN "GOSALESDW"."MRK_CAMPAIGN_LOOKUP"
"MRK_CAMPAIGN_LOOKUP"
                            ON "MRK_PROMOTION_DIM"."CAMPAIGN_CODE" =
"MRK_CAMPAIGN_LOOKUP"."CAMPAIGN_CODE"
GROUP BY
    "GO_REGION_DIM"."REGION_CODE",
    "GO_BRANCH_DIM"."COUNTRY_CODE",
    "GO_BRANCH_DIM"."BRANCH_KEY",
    "SLS_ORDER_METHOD_DIM"."ORDER_METHOD_KEY",
    "MRK_CAMPAIGN_LOOKUP"."CAMPAIGN_CODE",
    "MRK_PROMOTION_DIM"."PROMOTION_KEY";
```

11.5.3 Maintaining database aggregates

Just as a DBA has the flexibility to choose which database aggregates to create and how to create them, the DBA is responsible for the maintenance of these database aggregate tables. Database aggregates are usually built during a predefined ETL processing window when the underlying warehouse data is also updated. The DBA must consider the time and processing order that are related to the maintenance of any database aggregates that can be derived from one another. The DBA must also consider when the cube should be started, because in-memory aggregates are loaded at that time and can make use of the data in the database.

11.5.4 Monitoring database aggregate table hits

Cube metrics that are available in Cognos Administration can be used to monitor the database aggregate table hit rate, along with the hit rates of the result set cache, data cache, and aggregate cache. Select the cube to see the *Aggregate table hit rate* value.

Figure 11-30 is an example of the `gosldw_sales` metrics showing the database aggregate table hit rate.

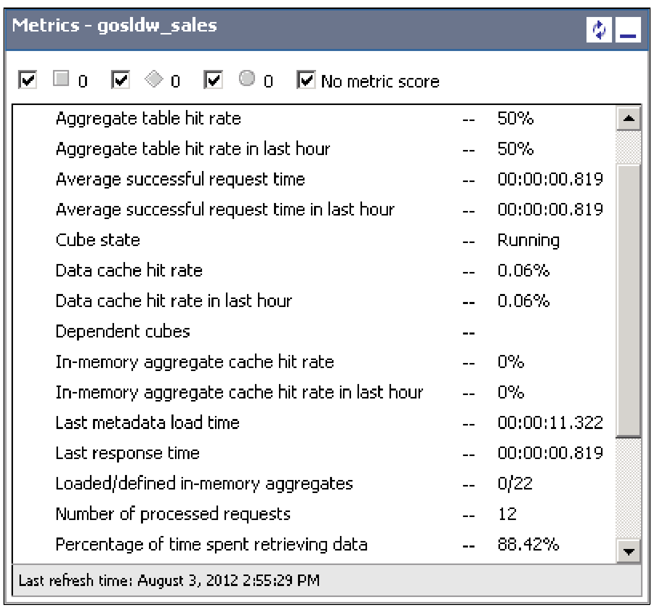


Figure 11-30 Example of the gosldw_sales metrics showing database aggregate table hit rate

11.5.5 Database aggregates tips and troubleshooting

This section provides troubleshooting information for database aggregates.

Suggested practices for database aggregates and Cognos Dynamic Cubes

The following suggested practices are related to database aggregates and Cognos Dynamic Cubes:

- Use the Aggregate Advisor to get recommendations for aggregates.
- Consider stacked aggregates, that is, having aggregates that are derived from one another. This practice can contribute to the following benefits:
  - Higher level aggregates to be derivable from lower aggregates.
  - Improved maintenance time.
  - Deep aggregates to cover more queries, and upper aggregates to give better performance.
- In Cognos Cube Designer, model the measures as additive when possible.

  If the measure is defined by an expression and the expression can be considered as additive, set the regular aggregate type to SUM. This practice enables the aggregate routing logic to use the in-database aggregate and enables the Aggregate Advisor to consider including it in its recommendations.
- For measures, use SUM and COUNT aggregates rather than AVERAGE, whenever possible.
- Denormalize the aggregate table to eliminate the need to join tables.

Troubleshooting Cognos Dynamic Cubes database aggregate routing

This section provides troubleshooting information for database aggregate routing.

Disable external database aggregate routing

The first step in troubleshooting a potential in-database aggregate query problem is to determine whether the problem is the result of the database aggregates. Disable the external aggregate support and rerun the scenario without the database aggregate awareness to see if the correct data values are returned:

1. Go to IBM Cognos Administration.
2. Select **Status**.
3. Select **System**.
4. Select the server.
5. Select the dispatcher.
6. Select **QueryService**.
7. In the **QueryService** menu, select **Set properties**.
8. Select the Settings tab.
9. In Dynamic cube configurations, select **Edit** in the Value column.
10. Each cube has its own aggregate cache, so select the pencil icon next to `gosldw_sales` to edit the configuration properties of the cube.
11. Select **Disable in-database aggregates**.
12. Click **OK**.

Restart the cube for the updated setting to take effect. Although in-database aggregates can be defined in the cube model, disabling this setting will not consider any of them for routing.

Using the query service log to understand database aggregate routing

This section describes advanced troubleshooting techniques by using more verbose logging levels and examining the query service log.

To generate some insight about which in-database aggregates are considered and why they are selected, enable the logging event group specifically for the database aggregate routing-related activity:

1. On the query service system, edit the DQM logging configuration, which is located relative to the server installation directory:

   `configuration\xqe.diagnosticlogging.xml`

2. Edit the aggregate cache event group to the log level of `trace`:

   ```
   <eventGroup name=" ROLAPQuery.AggregateStrategy" level="trace"/>
   ```

3. Save the `xqe.diagnosticlogging.xml` file.
4. It is not required to restart the query service for the diagnostic logging changes to take effect. Wait for the logging refresh interval, which has a default of 30 seconds.
5. Start the cube.
6. Run the problem query.
7. By default, the query service log is written to a file that is located relative to the server installation directory:

   `logs\XQE\xqelog-<timestamp>.xml`

8. When troubleshooting is complete, revert the log level change made in step 2 to disable logging of the aggregate cache event group.

As queries are processed in Cognos Dynamic Cubes, the analysis of which in-database aggregates are considered, which ones are rejected, and which ones are selected, is written to the query service log. A single user query can be decomposed into smaller queries so that some values that can benefit from database aggregates are routed to an aggregate table and other values are routed to the underlying warehouse tables.

The analyses for each of the smaller queries are found in log entries for the database aggregate routing event group denoted by the `<aggregateAnalysis>` element. The aggregate analysis element consists of the following sections:

- Original query: Lists the measures, levels, and the dimension and hierarchy of each level from the input query for consideration.
- Aggregates considered: Lists all of the matching aggregates that are qualified for routing the input query for consideration. There might be more than one matching in-database aggregate that is qualified and considered as a match.
- Aggregates selected: Describes the final aggregate that was selected from one or more qualified aggregates for consideration. It also describes the reason for choosing this aggregate.
- Aggregates not matching: All the aggregates that do not qualify for routing are listed here. The reason for not qualifying is also described for each non-matching aggregate. Typically, reasons are as follows:
  - Measure mismatch: The input query measure aggregation is not listed as part of the in-database aggregate, and therefore cannot be used.
  - Level mismatch: The levels at which the in-database aggregate is defined do not match with the level of aggregate of the input query.

– Aggregate slice mismatch: The slice coverage of the in-database aggregate is not enough to satisfy the input query even though the measure and levels are matching. For example, if the in-database aggregate is defined for slice of Time Year 2015, and the input query is asking for measures with Time Year 2014 and 2015, the in-database aggregate with only 2015 slice cannot be used to satisfy the input query.

– Aggregate cannot be rolled up for non-additive measures: Aggregates with additive measures, such as SUM and COUNT, can be used to satisfy queries of the higher level. For example, an aggregate with a SUM measure and Time levels to Quarter can be used to satisfy a query at the Year level because the values from the Quarter level can be added, or rolled up, to get the Year value with this aggregate. For non-additive measures, such as AVG and STDDEV, the roll up cannot be done for higher level queries.

Consider the cross-tab report shown in Figure 11-31 against the `gosldw_sales` cube for `Time Year` by the measures `Gross profit`, `Quantity`, `Revenue`, `Unit cost`, `Unit price`, and `Unit sale price`. There is an in-database aggregate, `gosldw_sales2`, that maps to the AGGR_TIME_PROD_OM_FACT database aggregate table, which aggregates measures `Quantity` and `Revenue` at the `Time Quarter` and `Products Product type` levels. In this example scenario, there are no in-memory aggregates enabled, only the database aggregates.

	<#Gross profit#>	<#Quantity#>	<#Revenue#>	<#Unit cost#>	<#Unit price#>	<#Unit sale price#>
<#Year#>	<#1234#>	<#1234#>	<#1234#>	<#1234#>	<#1234#>	<#1234#>
<#Year#>	<#1234#>	<#1234#>	<#1234#>	<#1234#>	<#1234#>	<#1234#>

Figure 11-31 Cross tab report of Time Year by gosldw_sales measures

The log entries in Example 11-12 show a cross-tab report user query, decomposed into two smaller queries and their corresponding aggregate analysis. One query does route to the `gosldw_sales2` aggregate because the measures and levels match.

Example 11-12 Cross-tab report user query (1)

```
<event component="XQE" group="ROLAPQuery.AggregateStrategy"...>
<![CDATA[<aggregateAnalysis>
<v5Report name="unknown"/>
<originalQuery>
<measures>
<measure name="Quantity"/>
<measure name="Revenue"/>
</measures>
<levels>
<level name="Year" dimension="Time" hierarchy="Time"/>
</levels>
</originalQuery>
<aggregatesConsidered>
<aggregateCube name="gosldw_sales2" ordinal="1"/>
</aggregatesConsidered>
<aggregateSelected>
<aggregateCube name="gosldw_sales2" reasonForChoosing="Only matching aggregate
cube found." SQLExecTimeInMS="-1"/>
</aggregateSelected>
<aggregatesNotMatching/>
</aggregateAnalysis>
]]></event>
```

The other query does not route because the measures do not match, as shown in Example 11-13.

Example 11-13 Cross-tab report user query (2)

```
<event component="XQE" group="ROLAPQuery.AggregateStrategy"...>
<![CDATA[<aggregateAnalysis>
<v5Report name="unknown"/>
<originalQuery>
<measures>
<measure name="Unit sale price"/>
<measure name="Unit price"/>
<measure name="Unit cost"/>
<measure name="Gross profit"/>
</measures>
<levels>
<level name="Year" dimension="Time" hierarchy="Time"/>
</levels>
</originalQuery>
<aggregatesConsidered/>
<aggregatesNotMatching>
<aggregateCube name="gosldw_sales2">
<ordinal value="1"/>
<reason value="measureNotMatching">
<measure name="Unit sale price"/>
</reason>
</aggregateCube>
</aggregatesNotMatching>
</aggregateAnalysis>
]]></event>
```

In this example, because one of the two decomposed queries was routed to an in-database aggregate, the aggregate table hit rate is 50%.

11.6 Modeling in-database aggregates

This section describes modeling considerations and how to incorporate in-database aggregates into the model using IBM Cognos Cube Designer.

11.6.1 Identifying in-database aggregates in the model

The objective of modeling in-database aggregates is to establish rules by which the dynamic cube can know when it can route a query to an aggregate table.

To perform this task, complete these steps:

1. Specify the measures and dimensions included in the aggregate table.
2. Identify the levels in the hierarchies that the aggregate table covers.
3. Specify a mapping between the keys in the dimensions and measures in the cube to columns in the aggregate table, and, if necessary, its related tables, such as rollup dimension tables.

In most cases, a query can be routed to the aggregate if all the measures and dimension objects of the query exist in the aggregate definition. Not all of the dimension objects or measures of the aggregate have to be in the query. If the query contains distributive (that is, additive) measures, further aggregation of records can take place to match the query. However, if the query contains non-distributive measures, the aggregate needs to be an exact match in order for the query to route to it.

For example, assume an in-database aggregate that consists of additive and non-additive measures is aggregated at the following levels: Month - Order Method - Products (see Table 11-1). The column *Route to in-database aggregate* in Table 11-1 indicates if the corresponding combination of query levels and measure types results in routing (Yes or No) and provides information explaining why or why not.

Table 11-1 Combination of query levels and measure types that route to an in-database aggregate

Query levels	Measure type	Route to in-database aggregate?
Year - Order Method - Products	Additive	Yes, can roll up Month to Year values for additive measures.
Month - Products	Additive	Yes, can roll up Order Method values for additive measures
Year - Products	Additive	Yes, can roll up Month to Year and Order Method values for additive measure
Month - Order Method - Products	Non-additive	Yes, it is an exact match for all levels.
Year - Order Method - Products	Non-additive	No, cannot roll up from Month to Year for a non-additive measure.
Month - Products	Non-additive	No, cannot roll up Order Method for a non-additive measure

Cognos Cube Designer allows you flexibility to work with many types of aggregate table scenarios.

The Aggregate Advisor in IBM Cognos Dynamic Query Analyzer generates aggregate tables that contain the fact and dimension information in one table in its recommendations. This type is referred to as degenerate dimension, and is the type of aggregate table you will use in most cases.

In Cognos Cube Designer, you can also model aggregates where the aggregate tables need relationships to be defined to the dimensions of the in-database aggregate, rollup dimensions, and slicers to support in-database aggregate tables that are already in your data warehouse.

In addition to the degenerate dimension type of aggregate table, other types of aggregate tales can already be in your data warehouse, including:

- Aggregate tables with relationships to the dimension tables
- Aggregate tables with relationships to rollup dimension tables
- Aggregate tables with slicers

Cognos Cube Designer also supports the modeling of these other types of aggregate tables, thereby allowing you to take advantage of your investment in any existing database aggregates.

You can use the samples to explore and learn about in-database aggregate modeling. The sample database GOSLDW contains one aggregate table. The name of the aggregate table is AGGR_TIME_PROD_OM_FACT. The type of aggregate table is a degenerate dimension, meaning that the fact and dimension information is contained in one table. The sample Cognos Cube Designer model contains an in-database aggregate named `gosldw_sales2`, which is stored in the `gosldw_sales` cube. For more information, see 11.6.10, “Exploring the in-database aggregate definition in the samples” on page 405.

11.6.2 Identifying aggregate table scenarios

The modeler needs to be aware of the nature of the aggregate table. This information determines what you need to do to model the in-database aggregate. The primary aggregate table scenarios are degenerate dimensions, rollup dimensions, parent-child dimensions, custom aggregation, and slicers. The following sections describe the different aggregate table scenarios.

Degenerate dimensions

If the aggregate table is a degenerate dimension, you have to know the nature of the key in the aggregate table. The nature of the key determines your modeling actions. It is possible that the key allows you to map the in-database aggregate to the level keys of dimensions in the cube. Conversely, the key might require you to create a relationship between the aggregate table and the dimension.

The recommendations of the Aggregate Advisor will produce degenerate dimensions with level keys of dimensions in the cube, thereby enabling straightforward level key mapping if the aggregate tables are implemented with the table structure as recommended.

If a key in the aggregate table matches the key of a level, then the modeling action is to map the appropriate levels to the aggregate table. For example, if the keys in the aggregate table contain the level keys of a Time dimension’s Year and Quarter levels, then you can choose to map the Time dimension to the in-database aggregate. A query that used Year or Quarter is routed to use the aggregate table.

The sample table AGGR_TIME_PROD_OM_FACT is this type of aggregate table. The result, in relational query terms, is that the aggregate table is generated in the relational query structure as a stand-alone table without the need for relationships to be defined in the relational model.

Figure 11-32 shows an example of a relational query model of an in-database aggregate of this type. As you can see, the fact table, SLS_SALES_FACT, has its dimension tables revolving around it. Out of the picture are dimensions that are also in the cube but are not in the in-database aggregate definition. The aggregate table is shown at the top of the figure without the need for join relationships to dimension tables.

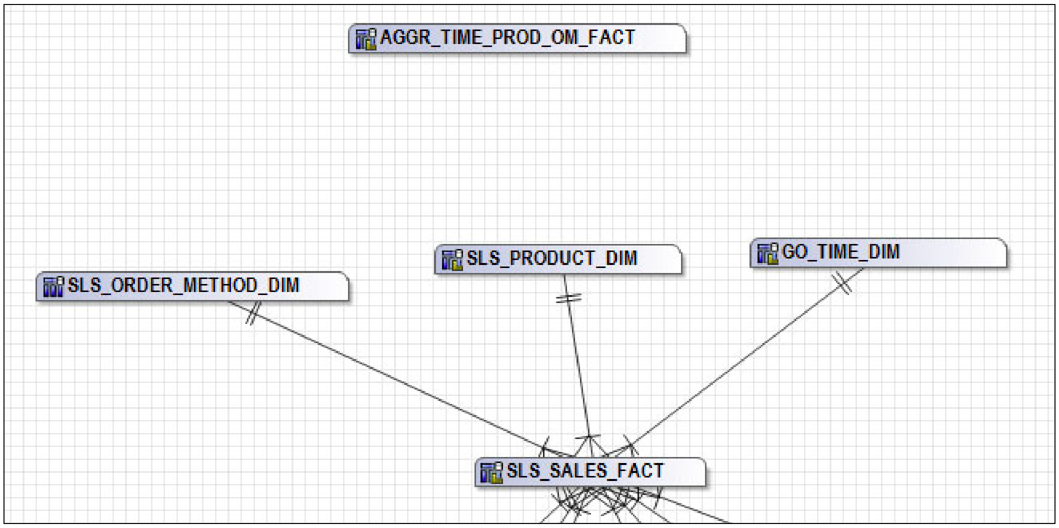


Figure 11-32 Relational table relationships for degenerate dimension case

Join relationships to dimensions

Alternatively, you might have a situation in which the aggregate table needs to have relationships to the dimensions. If a key in the aggregate table matches the key in a level such that the key can be used to identify a relationship between the dimension and the in-database aggregate, you can define those relationships between the aggregate table and the dimension table objects. The aggregate table becomes an alternate fact table with relationships defined between it and its participating dimensions. Figure 11-33 provides a conceptual picture. As you can see, the aggregate table has relationships between it and the dimensions that are in the in-database aggregate table.

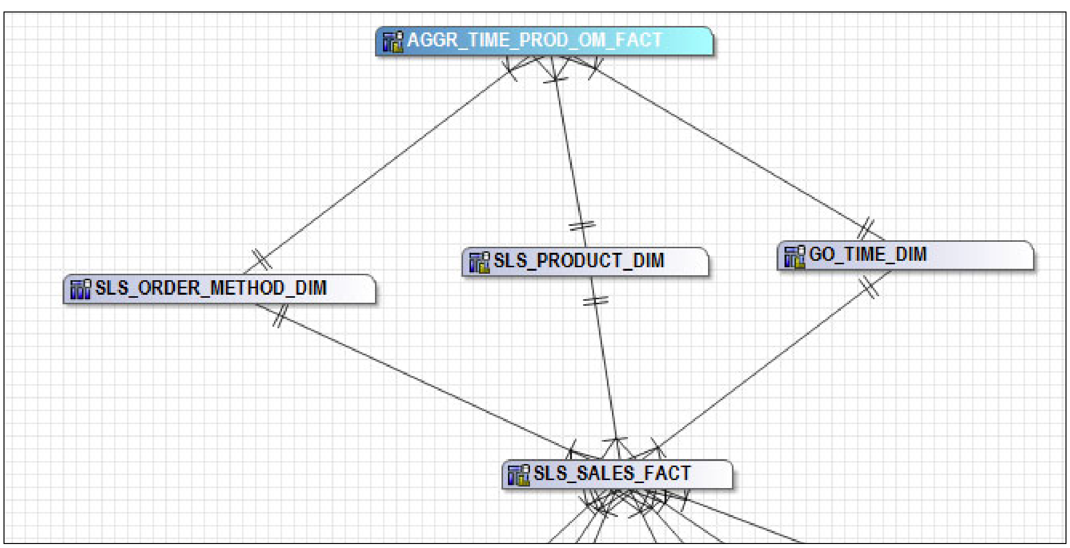


Figure 11-33 Relational table relationships showing joins to dimensions

Rollup dimension tables

It is possible that the aggregate table has associated rollup dimension tables. You then need to model the in-database aggregate to include them. The aggregate table and its rollup dimensions will be a relational schema separate from the schema of the detail fact table and its relational schema.

Parent-child dimension

It is possible that one dimension in the in-database aggregate is a parent-child dimension. Parent-child dimensions do not have defined levels. The data of the parent and child attributes determine how the members are generated and assembled into a member tree. The aggregate table has a record for each member. The mapping allows the query to route to the aggregate table.

Custom aggregation

You need to know if a measure in the cube has custom aggregation. If it does, then your aggregate table has to support it as well. Custom aggregation is a method to predefine the aggregation of member values outside of the aggregation functionality of Cognos Dynamic Cubes. For more information about custom aggregation, see *Regular aggregates in the Dynamic Cubes User Guide 10.2.2* at:

http://www.ibm.com/support/knowledgecenter/SSEP7J_10.2.2/com.ibm.swg.ba.cognos.ug_cog_rlp.10.2.2.doc/c_cmdregagg.html

Slicers and partitioned aggregates

Aggregate tables that contain a subset, or a partition, of the data can be identified in the model with in-database aggregate slicers. Slicers indicate that the measures contained in the in-database aggregate are captured for a subset of the original measure dimension. For example, an in-database aggregate can contain data for one or two particular years, as compared to the underlying fact table that contains data for 10 years.

What to do for each case

For any particular aggregate table scenario, you have to do several tasks to correctly model the in-database aggregate. Some are common to all cases. Some are specific to one case. The following sections describe each of these modeling tasks.

11.6.3 Degenerate dimensions with matching level keys

For the case of degenerate dimensions that have keys that match the level keys of their analogue dimensions, perform the following tasks:

- Specify which measures participate in the in-database aggregate.
- Specify which dimensions participate in the in-database aggregate.
- Map the appropriate facts in the aggregate table to those measures in the measure dimension.
- Set the level grain for those hierarchies in each dimension that is involved in the aggregate table.
- Map columns in the aggregate table to the level keys.

The Aggregate Advisor recommends aggregate tables with this type of structure.

The exercise in 11.6.10, “Exploring the in-database aggregate definition in the samples” on page 405 guides you through the in-database aggregate of the sample model, which is of this type.

Figure 11-34 shows the measure mapping page of the in-database aggregate editor. The tooltip shows the mapping column value qualified with the table name. In this case it is the SALE_TOTAL column of the AGGR_TIME_PROD_OM_FACT aggregate table.

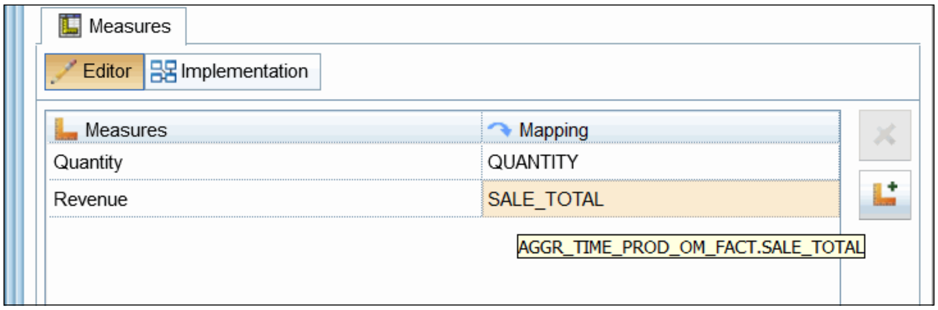


Figure 11-34 Measure mappings to the aggregate table columns - Tooltip shows table and column names

Figure 11-35 shows the definition of the in-database aggregate in the sample model. The `Products` dimension is included in the aggregate. The aggregate grain is set at the `Product type` level.

Setting the aggregate grain at the `Product type` level makes it possible for the query to route to the aggregate table if the query for a distributive measure comes from an object from either the `Product line` level or `Product type` level.

A query that involves an object from the Product level cannot route to the aggregate, but it can route to the detail fact table.

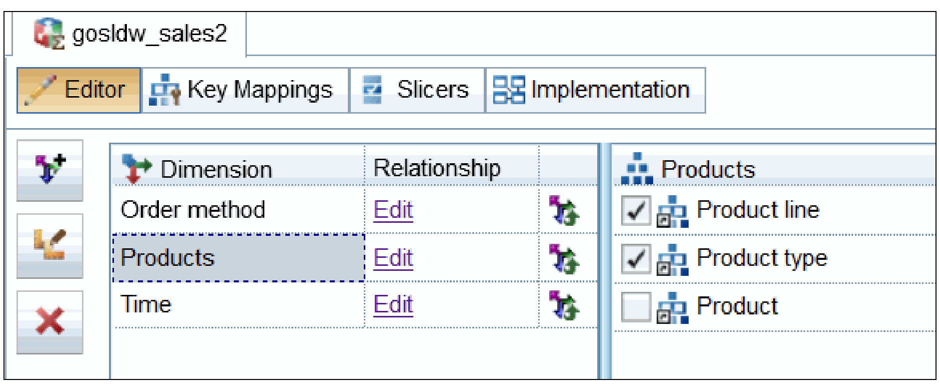


Figure 11-35 In-database level grain specification

You do not need to specify a relationship between the dimensions and the aggregate table because the level keys of the dimensions match those of the dimension key columns in the aggregate table,.

If more than one key is designated as part of the level unique key in a level, all those keys must be mapped to the keys of the aggregate table.

Figure 11-36 shows a dimension which does not have its aggregate grain set. The mapped dimension icon is not displayed. After you set the level grain, or, if you are working with a matching join key scenario, you have set the relationship between the dimension and the aggregate table, the mapped dimension icon is displayed.

The next step is to select the aggregate grain for the Product brand hierarchy. In this case, select the box for the Product brand level.

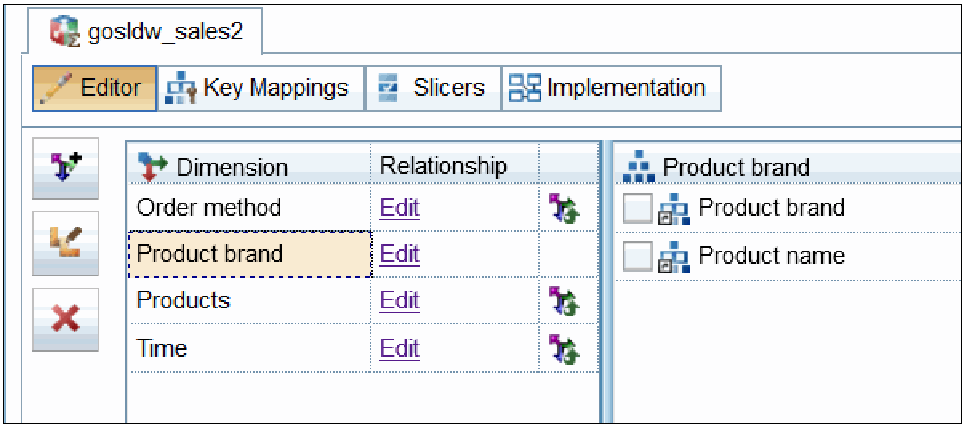


Figure 11-36 Unset aggregate grain level

Figure 11-37 on page 401 shows the Key Mappings window of the in-database aggregate. The aggregate grain for Product brand is set to the Product brand level. The level key of that level is the `Product brand` key. The key is not mapped to an object in the aggregate table yet. The next step is to map the `Product brand` key to an aggregate table column.

You might recall from examining the in-database aggregate of the sample model that the aggregate grain level of the dimension for the sample in-database aggregate is set to the `Product type` level.

Keep in mind that in Figure 11-35 on page 399, the check boxes were selected for the `Product type` level and the `Product line` level above it.

Notice in Figure 11-37 that the level unique keys for both the `Product line` and the `Product type` levels are listed in the Key Mappings page. Both keys must be mapped.

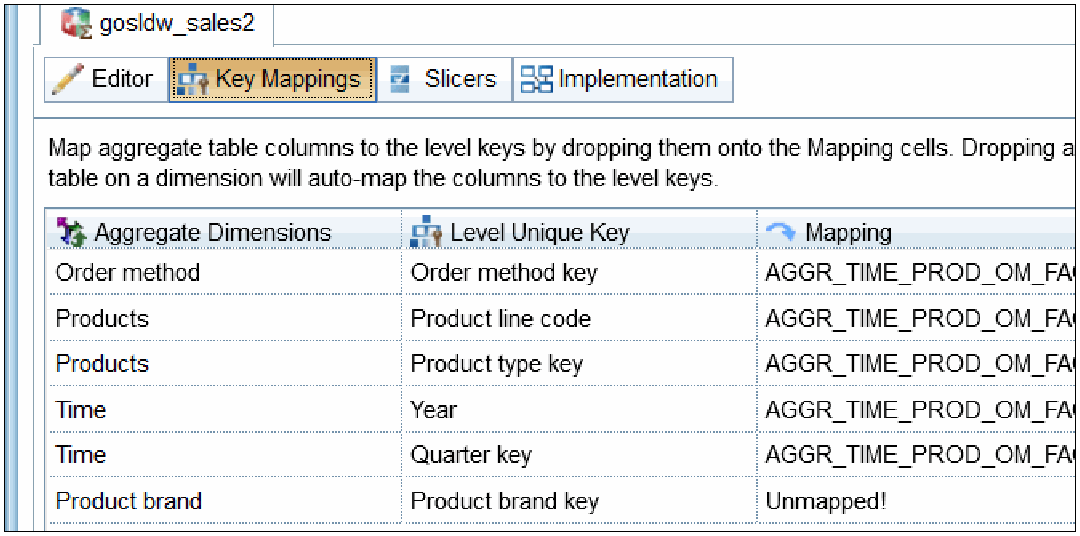


Figure 11-37 Key Mappings page showing mapped and unmapped level keys.

11.6.4 Aggregate table with matching join keys

In the case of an aggregate table that has keys that match the join keys, perform the following tasks:

- Specify which measures participate in the in-database aggregate.
- Specify which dimensions participate in the in-database aggregate.
- Map the appropriate facts in the aggregate table to those measures in the measure dimension.
- Create relationships between the aggregate table and the dimensions.

Note that if you choose to create relationships between the aggregate table and the dimension and you do not specify the level grain, the lowest level is implied. Figure 11-38 shows an example of the `Time` dimension with no levels of the `Time` hierarchy selected. When you create a relationship to the dimension, the lowest level, `Day`, is implied as the level of the aggregate table.

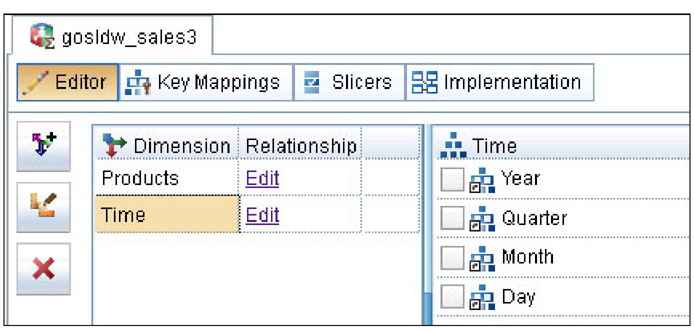


Figure 11-38 Time hierarchy with no levels selected

To create a relationship between the aggregate table and the dimension, click the **Edit** link in the Relationship column for the corresponding dimension. Figure 11-39 shows an example of the relationship between the `Time` dimension and the aggregate table. Note on the left side that the `Time dimension` key, DAY_KEY, is from the GO_TIME_DIM dimension table, and is correlated to the DAY_KEY of the aggregate table, shown on the right side.

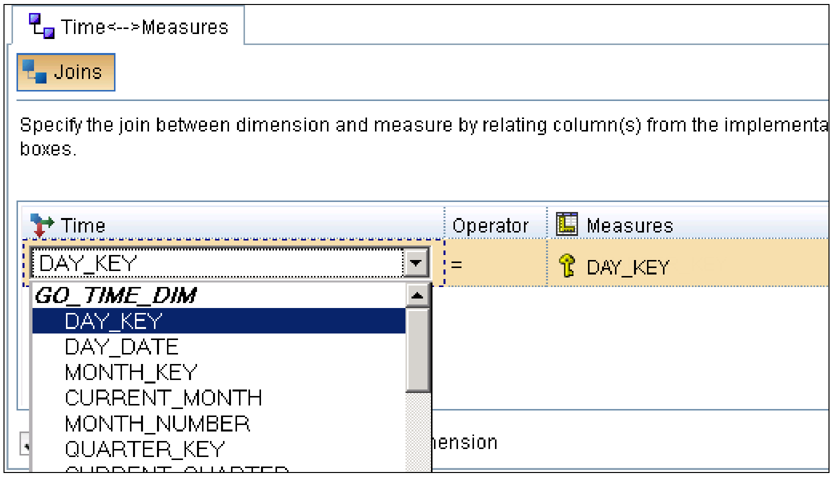


Figure 11-39 Relationship editor for aggregate table join key definitions

11.6.5 Rollup dimensions

For rollup dimensions, perform the following tasks:

- Specify which measures participate in the in-database aggregate.
- Specify which dimensions participate in the in-database aggregate.
- Map the appropriate facts in the aggregate table to those measures in the measure dimension.
- Set the level grain for those hierarchies in each dimension that is involved in the aggregate table.
- Map the key mappings to the keys of the rollup dimensions.
- Define the relationship between the rollup dimension tables and aggregate fact table.

The relationships are created between the dimensions participating in the in-database aggregate and the rollup dimension tables. The Implementation tab shows the rollup dimensions, the aggregate table, and the relationships between them.

11.6.6 Parent-child dimensions

Parent-child dimensions are treated differently than level-based dimensions. If you choose to remap the dimension, the members will be mapped to a row in the aggregate table if the key value that is specified in the child key value matches a value for a record in the table. In this way, you map the dimension to a table with custom aggregation. You must specify a key mapping between the parent-child dimension and the aggregate table. The dimension role that you use is the child role.

Figure 11-40 shows an in-database aggregate with a parent-child dimension. Cognos Cube Designer detects that the dimension is a parent-child dimension and presents this option for you. It shows that the modeler has chosen to map the key.

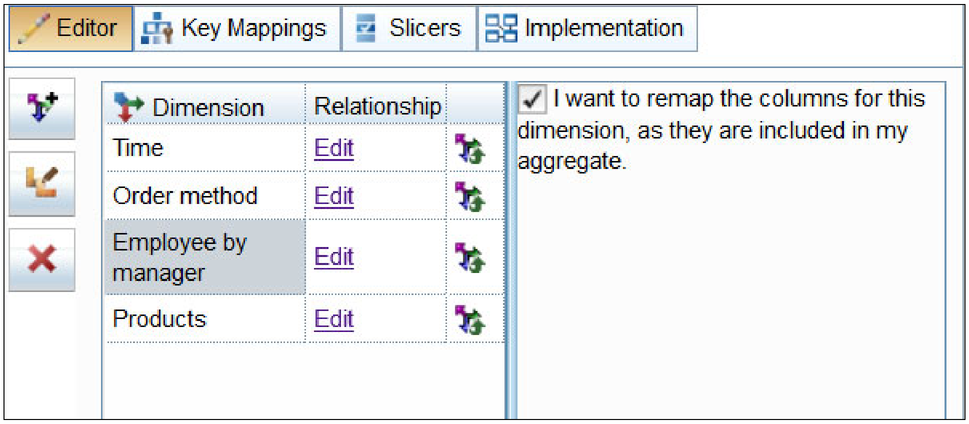


Figure 11-40 In-database Aggregate editor for a Parent-child dimension

Figure 11-41 shows the Key Mappings tab of the in-database aggregate. The child role attribute is shown before being mapped to the matching column in the aggregate table. The next step is to map the aggregate table child role column.

If you do not choose to remap the dimension, you must define a relationship between the dimension and the aggregate table.

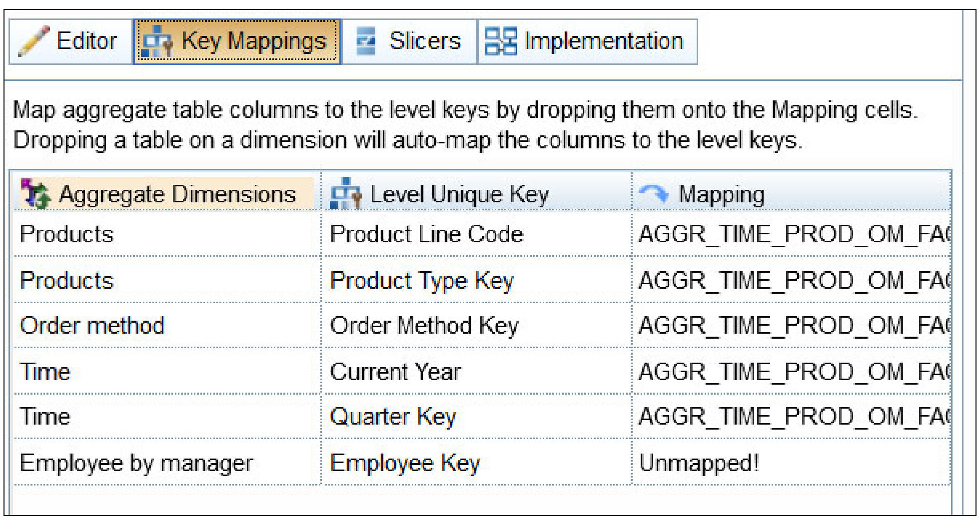


Figure 11-41 Key Mappings editor with an unmapped level key

11.6.7 Custom aggregation

This scenario involves measures with regular aggregate of type Custom. The modeling of the measures and dimension is no different than the scenarios previously described. You have to specify a mapping so that the members in the query can derive their associated measure value for a measure that has custom aggregation from the appropriate records in the aggregate table. It is worth noting that if a query with a custom aggregate type measure cannot be routed to the aggregate table because it is not an exact match to the levels of the aggregate, it is not routed to the underlying fact table and returns a null value.

11.6.8 Slicers and partitioned aggregates

You must identify the members that are slicers and add them to the in-database aggregate definition by adding them into the slicers tab of the in-database aggregate editor.

Each in-database aggregate definition corresponds to a single aggregate table. If you have multiple aggregate tables, each partitioned to aggregate a different subset of the data, then you will model two in-database aggregate definitions in Cube Designer. For example, if you have one aggregate table for year 2013 and another aggregate table for 2014, then the two separate in-database aggregate definitions will have the same selected measures, dimensions, and level grains. However, the members in the slicers will be different.

Figure 11-42 shows an example of one of the in-database aggregate definitions with the year 2013 member in the Slicers tab.

Figure 11-42 In-database aggregate with Time Year 2013 in the slicer

Figure 11-43 shows an example of the other in-database aggregate definition with the year 2014 member in the Slicers tab.

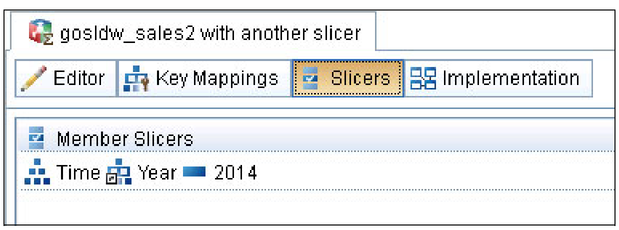


Figure 11-43 In-database aggregate with Time Year 2014 in the slicer

If you have a query for years 2012, 2013, and 2014 against a cube with these two in-database aggregates, each with a slicer, then you will have at least three SQL queries:

- One query for 2013 data to the aggregate table with the 2013 slicer
- Another query for 2014 data to the aggregate table with the 2014 slicer
- A query to the underlying fact table for 2012 data

If relative time is enabled on the Time dimension, because it is in the `gosldw_sales` cube, you can use relative time members in the slicer for the in-database aggregate definitions.

Figure 11-44 shows an example of an in-database aggregate definition with the relative time member, Current Year, in the Slicers tab. Note that the relative time member caption as seen in Cognos Cube Designer has an example year in parentheses. However, having a time year in parentheses does not necessarily correspond to the actual current period. The actual current period is evaluated at cube start time.

Figure 11-44 In-database aggregate with relative time member, Current Year, in the slicer

If you use this approach, that is, using a relative member in the slicers, you only need to ensure that the aggregate tables and the relative time values are synchronized so that the data correlates with each other. For this approach to work correctly, the relative time hierarchies must all have a single root member, which is typically the case. A benefit of this approach is that you do not have to edit the model whenever the time period changes for the aggregate tables, for example, when the current year is no longer 2014 but a new year.

11.6.9 Other modeling considerations

The objective is to select the aggregate grain that matches the grain of the dimensions that are contained in the aggregate table. If you choose to map the aggregate grain of a dimension to a level above the fact grain of the aggregate table, the query will still route to the in-database aggregate, but some aggregation must still be performed by the query engine. For example, AGGR_TIME_PROD_OM_FACT has two columns that map to level keys of the time dimension. They map to the level keys of the year and quarter levels of that dimension. Although you could set the aggregate grain of the time dimension to be years, doing so might not be the best approach because it precludes some queries from routing to an in-database aggregate, defeating the purpose of the in-database aggregate.

If you set the aggregate grain to below the grain of the aggregate table, you need to map level keys to objects that do not exist in the aggregate table, which is quite difficult to do. If your application must map that grain, the aggregate table must be modified to include that grain.

Be sure to understand the concept of level keys. The keys of all the levels that you set as included in the aggregate grain must be defined in the aggregate table. That is why AGGR_TIME_PROD_OM_FACT, which is the aggregate table that is included in the Cognos sample relational data base GOSLDW, does not have only the quarter level keys but also the year level keys. If your level key needs to include keys from higher levels to uniquely identify the members in the level, there is no need to map the key twice. Although the object is used more than once, there will be only one reference of the object to map.

11.6.10 Exploring the in-database aggregate definition in the samples

Use the following steps to explore the in-database aggregate in the samples:

1. In Cognos Cube Designer, select the `gosldw_sales` cube.
2. Double-click the Project Explorer tree node for the cube or right-click **Open Editor** and select the Aggregates tab**.** You see an in-database aggregate named `gosldw_sales2`.

3. Right-click `gosldw_sales2` and select **Open Editor**. You see three dimensions: `Time`, `Order method`, and `Products`.
4. Select **Time**.

   To the side, you see a list of hierarchies and levels. In the case of Time, only one hierarchy is listed, because it has only one hierarchy. If there were multiple hierarchies, they would also be listed. Several levels show check boxes that are selected: **Year** and **Quarter** levels. This selection indicates that the aggregate level grain for the Time dimension is the Quarter level.
5. Deselect the **Year** check box.

   The **Quarter** check box is cleared also, which indicates that you must include the keys of higher dimension grains in the in-database aggregate to correctly route the query.
6. Select **Quarter**. The **Year** check box also becomes selected.
7. Examine the other two dimensions.

   Notice that Order Method has only one level. Because the aggregate table does contain values at the Order Method level, this single level must be selected to accurately identify the levels covered in the aggregate.
8. Click **Measures**.

   You see two measures, **Quantity** and **Revenue**. In the mapping column, you see the columns in the aggregate table that map to these measures. They are QUANTITY and SALE_TOTAL.
9. Hover the mouse over one of the cells in the mapping column.

   The aggregate table name and the column name appear in a tooltip. This step enables you to trace the object mapping back to the source. For example, the tooltip for the mapping for the measure **Quantity** displays the following text: `AGGR_TIME_PROD_OM_FACT.QUANTITY`.
10. Return to the in-database aggregate editor and click the Key Mappings tab. You see a list of level unique keys, their source dimensions, and the aggregate table mapping.
11. Hover the mouse on one of the cells in the mapping column.

    The aggregate table name and the column name appear in a tool tip. This step enables you to trace the object mapping back to the source. Because AGGR_TIME_PROD_OM_FACT is a degenerate dimension with level keys, it is sufficient to map the cube to `gosldw_sales2` with the level key mapping.
12. Notice that the Year and Quarter key rows show as unmapped. The reason is because in step 5 on page 406 and step 6 on page 406, you deselected and selected different grain levels for the Time dimension.
13. Return to the Aggregates tab.
14. Click the relationship edit links for any of the dimensions. The Relationship Editor opens. A message indicates that no joins are necessary. Return to the Aggregates tab.
15. Click **Undo** to restore the level grain or manually click the level grain that was being used.
16. Click **New Dimension**. A window opens that lists the dimensions that exist in the cube but do not yet exist in the in-database aggregate.
17. Select any of the dimensions in the list and click **OK**. The dimension is added to the in-database aggregate.
18. Select the new dimension and click **Delete**. The dimension is removed from the in-database aggregate.

19. Select the **Slicers** tab.

 If you needed to add slicers, expand the member browser for the dimension, select them, and add them to the slicer list.

20. Select the **Implementation** tab.

The Implementation tab shows a diagram representation of the in-database aggregate. The diagram is fairly simple, there is only one table in the in-database aggregate because AGGR_TIME_PROD_OM_FACT is a degenerate dimension. If the in-database aggregate contained rollup dimensions, they are presented in the implementation diagram as well.

11.6.11 Automatic aggregate creation

You can use the in-database aggregate auto-matching functionality to create an in-database aggregate automatically. You can do this by dragging the aggregate table onto the aggregates field of the aggregates tab of the cube that you are working on. Where matching measures and dimensions are found in the cube, Cognos Cube Designer maps each of these items to the aggregate table. Where possible, it also attempts to identify the highest level of aggregation required and roll up dimensions. Inspect the in-database aggregate and refine it to ensure that it is modeled as you need it to be.

The ability to automatically map depends on how the aggregate tables are set up. A naming convention for in-database aggregate columns that conforms them to their analogue source objects is helpful. Complete the following steps:

1. Select the `gosldw_sales` cube.
2. Double-click the **Project Explorer** tree node for the cube or right-click **Open Editor** and click the Aggregates tab.

 You see an in-database aggregate called `gosldw_sales2`.

3. Expand the metadata tree in the Data Source Explorer and select AGGR_TIME_PROD_OM_FACT.
4. Drag AGGR_TIME_PROD_OM_FACT onto the in-database aggregate Editor.
5. A new in-database aggregate is created. Examine the dimensions, measures, levels, and level key mappings that are created in the in-database aggregate.

Cognos Dynamic Query Analyzer has an Aggregate Advisor, which can identify, from the model or from log files, possible candidates for the creation of aggregate tables. For more information about the Aggregate Advisor, see 11.3, “Overview of the Aggregate Advisor” on page 343.

11.7 Modeling user-defined in-memory aggregates

Modeling a user-defined in-memory aggregate is similar to modeling an in-database aggregate, in that you must first select the measures and dimensions that participate in the aggregate, and then specify the level grain for those hierarchies in each dimension that is involved in the aggregate.

Because there is no aggregate table to map to, there is no need to specify key mappings or join relationships.

For the sake of example, assume that you want to create a user-defined in-memory aggregate for the measures `Quantity` and `Revenue` at the `Product type` level and `Quarter` level.

Perform the following steps:

1. Click **New User-Defined In-Memory Aggregate** to create a new user-defined in-memory aggregate. Figure 11-45 shows an example of the New User-Defined In-Memory Aggregate window with measures and dimensions selected.

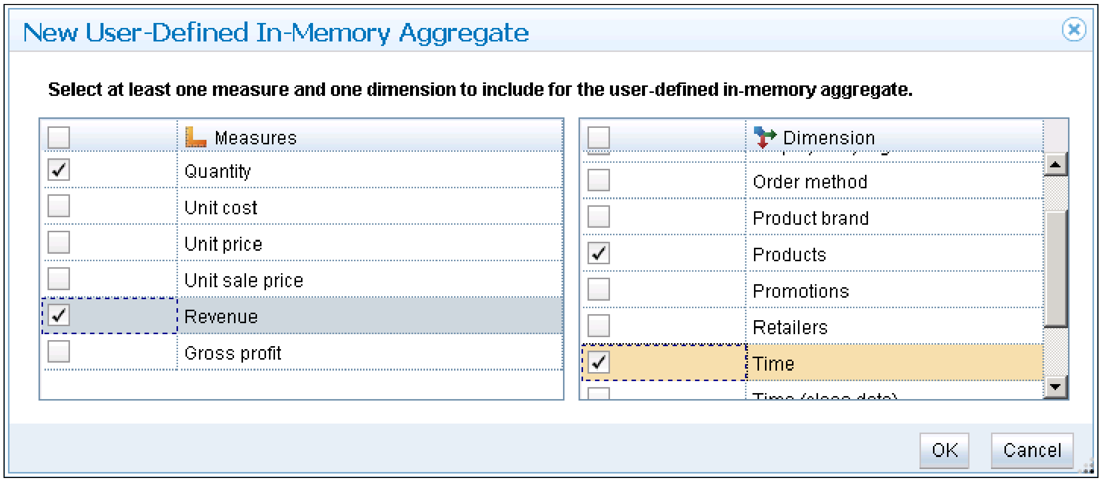


Figure 11-45 New User-Defined In-Memory Aggregate window

2. You can give the user-defined in-memory aggregate a more meaningful name, as it will be the name of the in-memory aggregate displayed by the Aggregate Advisor.
3. Double-click the new user-defined in-memory aggregate definition created to open it in an editor with two tabs: Dimensions and Measures.
4. In the Dimensions tab, select each dimension and identify aggregate level grain for the hierarchy, and add more dimensions to the definition as needed.
5. The Measures tab lists the selected measures. You can choose to add more measures on this tab.

Figure 11-46 shows an example of the Dimensions tab with `Product type` level of the `Products` dimension selected as the level in this user-defined in-memory aggregate.

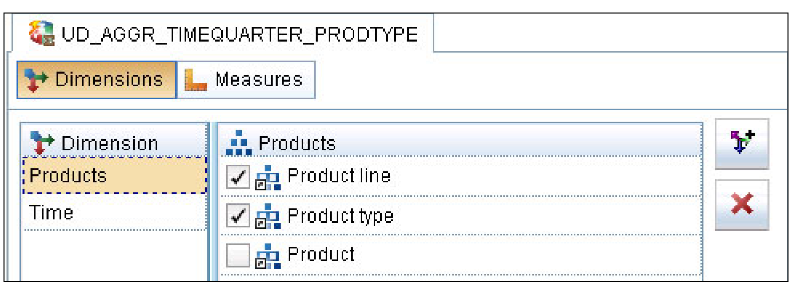


Figure 11-46 Dimensions tab of a user-defined in-memory aggregate showing Products dimension

Figure 11-47 shows an example of the `Dimensions` tab with the `Quarter` level of the `Time` dimension selected as the level in this user-defined in-memory aggregate.

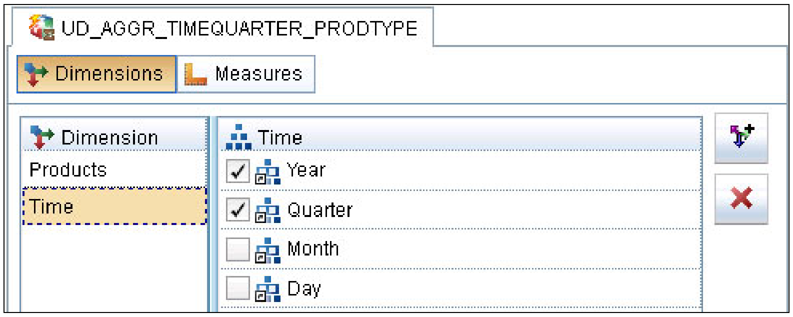


Figure 11-47 Dimensions tab of a user-defined in-memory aggregate showing Time dimension levels

Figure 11-48 shows an example of the `Measures` tab with a list of the measures in this user-defined in-memory aggregate.

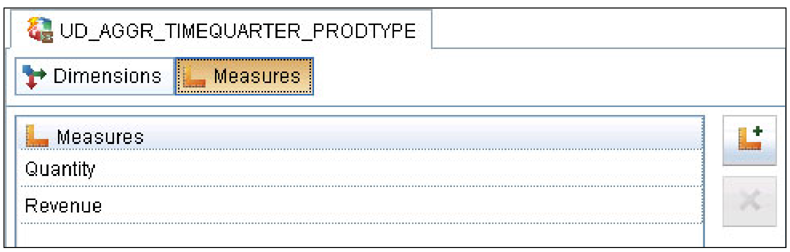


Figure 11-48 Measures tab of a user-defined in-memory aggregate

Unlike in-database aggregates, user-defined in-memory aggregates do *not* support these types:

- Slicers
- Parent-child dimensions
- Measures with aggregate rules applied to them

Non-distributive measures, such as Average, can be included in the aggregate. However, these types of measures can only be used if the query is an exact match of the aggregate.

Note the warning text in the empty user-defined in-memory aggregates window:

```
User-defined in-memory aggregates can affect the performance and memory utilization
of a dynamic cube. Use the Aggregate Advisor to incorporate the user-defined
in-memory aggregates to a dynamic cube.
```

This is the reason why after identifying the user-defined in-memory aggregates and publishing the cube model, you have to run the Aggregate Advisor to estimate the space required for the in-memory aggregates and obtain actionable in-memory aggregate recommendations to apply to the cube.

11.8 Cache Priming Techniques

This section describes two approaches for ensuring that relevant data is in the Cognos Dynamic Cubes caches to accelerate query performance: *preloading* and *priming*.

11.8.1 Preloading the caches

Preloaded caches are loaded with data that is based on metadata definitions associated with the cube model data source. They are loaded at cube start, cube restart, and cache refreshes. In Cognos Dynamic Cubes, the member cache and the aggregate cache are both considered preloaded caches. After they are built and loaded from definition, the member cache and the aggregate cache are not affected by and do not grow based on query usage. Only changes to the cube model or changes to the set of in-memory aggregates that are saved for the cube will affect these caches at the next cube start.

The member cache is always loaded. The size of the member cache is a key factor in determining overall JVM heap size and needs for the cube. For more information about JVM heap size and cache size considerations, see *IBM Business Analytics Proven Practices: Dynamic Cubes Hardware Sizing Recommendations* at:

http://ibm.biz/DynamicCubesHWSizingRec

To use the aggregate cache, run the Aggregate Advisor to get and save recommendations for in-memory aggregates, and enable the aggregate cache by specifying enough memory for the corresponding cube setting. For more information, see 11.4.2, “Loading in-memory aggregates into the aggregate cache” on page 367. At the core of the Aggregate Advisor logic is a model-based evaluation that uses heuristics to analyze potential slices of the cube to recommend aggregates that balances coverage with effective size. Running the Aggregate Advisor to consider workload information in the workload log will weigh the measures and slices that are actually hit by users so that the aggregate recommendations are more reflective of query usage of the data.

As described in 11.4.2, “Loading in-memory aggregates into the aggregate cache” on page 367, in-memory aggregates can take some time to load, so make sure that the cube start is initiated during, or immediately after, the underlying database ETL process to allow enough time for the load and to provide optimal query performance to users.

11.8.2 Priming the caches

Cognos Dynamic Cubes cache-priming involves running a set of reports to populate the result set cache, expression cache, and data cache to accelerate query performance for users. This technique can be used to optimize specific reports, especially if they compute lots of values or it is more efficient to cache certain information in a targeted way rather than relying on large aggregates.

The reports that are used for cache-priming can be from a known set of dashboards and reports that a majority of users will use, or those that process large volumes of data to cache upfront for reuse by multiple users. For example, if there are a set of dashboards or reports that most users use as a starting point, these are good candidates for priming so that all users can benefit from the quick performance that cached values can provide.

The result set cache, expression cache, and data cache are populated in response to incoming queries. Therefore, cube administrative commands such as Refresh Data Cache and Refresh Member Cache will clear these caches, and incoming queries will miss the cache until they are repopulated by queries. These caches are gradually populated as user queries are run and can be used after the values are stored. However, query performance is not as good as when the caches have the relevant data available. The reason is that there can be longer response times as a result of data queries to the underlying data source or the processing of large volumes of data.

After a cache refresh, cache repopulation can occur naturally in response to user queries, or proactively by scheduling priming reports to run immediately after cache refresh. Note that refreshing the data cache also refreshes the aggregate cache. The aggregate cache starts its preloading automatically. Depending on the nature of the priming reports, running these priming reports after the aggregate cache completes loading might be better so that these reports can also use the aggregate cache and not compete for the same underlying resources.

11.8.3 Scheduling priming reports to run by trigger

Depending on the nature of the priming reports, running these priming reports after the aggregate cache completes loading might be better so that these reports can also use the aggregate cache and not compete for the same underlying resources.

You can schedule reports automatically based on a trigger so that running these reports primes the caches, in addition to having aggregates:

1. Go to IBM Cognos Connection.
2. Select the **Schedule** icon action for a report to be scheduled.
3. Select the By Trigger tab under **Frequency**.
4. For **Trigger name**, specify a name for the trigger.
5. Click **OK**.

Figure 11-49 shows an example of specifying a trigger named `afterload` to signify this trigger is for after in-memory aggregates load.

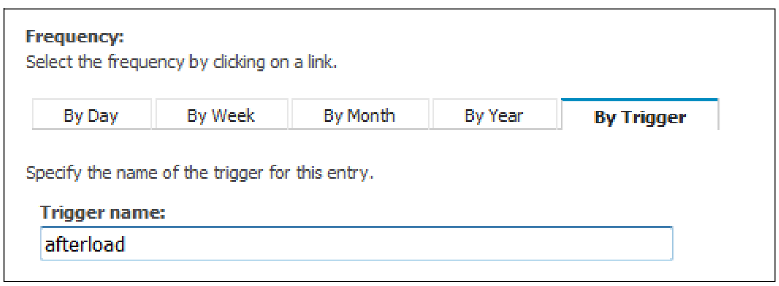


Figure 11-49 Scheduling a report to run by trigger and specifying a trigger name

After the scheduled report trigger is defined, the trigger name can be identified as the trigger to run for a cube after in-memory aggregate load has completed:

1. Go to IBM Cognos Administration.
2. Select **Status**.
3. Select **Dynamic Cubes**.
4. Select **Data Stores (All)**.
5. Select the cube.
6. Select the server group.
7. Select the QueryService.
8. In the **QueryService** menu, select **Set properties**.
9. Enter the trigger name for the `Post in memory trigger name` property.
10. Click **OK**.

If the cube is deployed to multiple server groups and multiple instances of the query service, then each cube instance has its own set of cube properties.

Figure 11-50 shows an example of specifying the post in-memory aggregate trigger named `afterload` for the `gosldw_sales` cube.

Figure 11-50 Specifying the post in-memory trigger name in cube properties

Notice that there is also the `Startup trigger name` property. This property specifies the trigger event that is sent when the cube is started. Keep in mind that in-memory aggregates are loaded in parallel when the cube is started. Therefore, to use the in-memory aggregate cache and avoid unnecessary concurrency of running priming reports while aggregates are loading, consider using the `Post in memory trigger name` property.

Cube actions such as restarting the cube, refreshing the metadata cache, and clearing the data cache, reload the in-memory aggregates. When reloading the in-memory aggregates is complete, the cube invokes the post in-memory trigger to run all the reports scheduled to run by the specified trigger name.

11.8.4 Advanced setting to identify priming-only reports

After a report is run, the report results are stored in the result set cache so that when the same report is run again, it will have the fastest response possible. For priming reports that are used solely to populate the data cache, the results set does not need to be changed because no users actually care for the report results.

Using a QueryService Advanced Setting, you can identify these data-cache-only priming reports and reduce the amount of time and resources to run the specific priming reports.

Use the following steps to identify a report as a data-cache-only priming report:

1. Launch **IBM Cognos Administration**.
2. Under Status, click **System**.
3. In the Scorecard window, click the server link → the dispatcher link → the **Query Service** link.
4. In the **QueryService** menu, select **Set properties**.

Figure 11-51 shows the query service context menu.

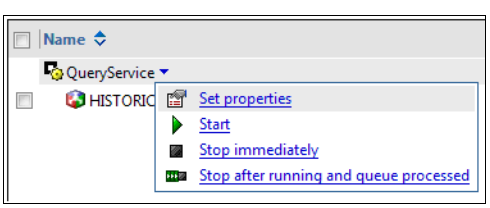


Figure 11-51 Query service context menu

5. Click **Settings** and then **Edit** next to Advanced settings.

   Figure 11-52 shows the query service settings.

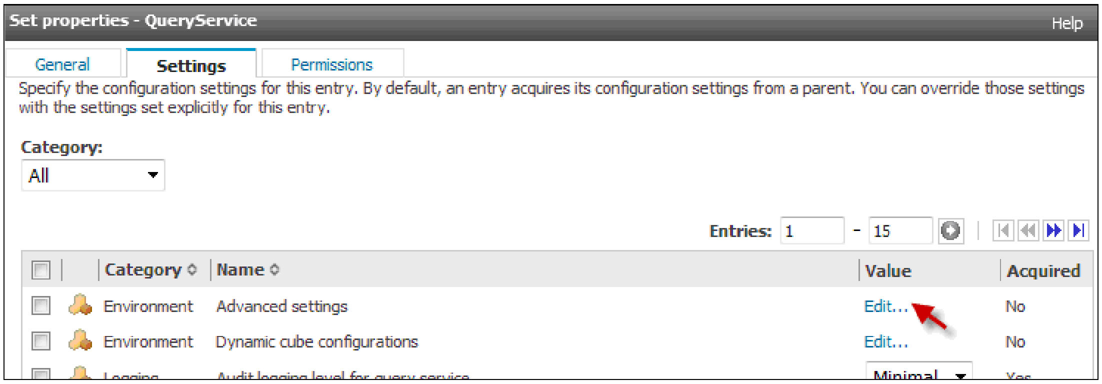


Figure 11-52 Query service settings

6. In an available row, enter `qsPrimingReportNamePrefix` under the **Parameter** field. Under the **Value** field, enter a name prefix for reports used to populate the data cache.

   Figure 11-53 shows the query service advanced settings

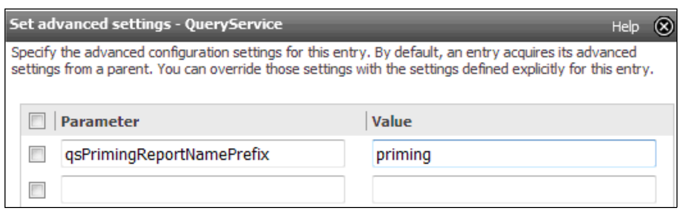


Figure 11-53 Query service advanced settings

7. Click **OK** to close the Set advanced settings window, and **OK** again to close the QueryService properties window. A QueryService restart is required before the changes take effect.

When a report that has a name that starts with the prefix specified by the priming report name prefix property runs, the data cache is populated with all the data needed to satisfy the report, but no data is returned as a result of the run. The result is an empty result set and so nothing is saved in the result set cache. Figure 11-54 shows the empty report that results from running a priming-only report in report.

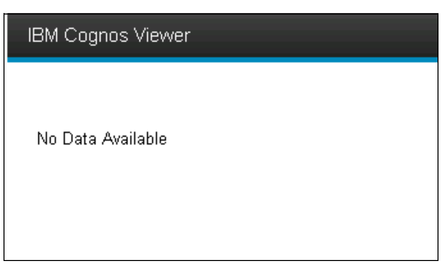


Figure 11-54 Empty report resulting from running a priming report

Note that setting the priming report name prefix property does not schedule reports to run as a priming report. To schedule a report to run to populate the caches, see 11.8.3, “Scheduling priming reports to run by trigger” on page 411.

11.8.5 Optimization of calculated measures

Calculated measures are computed by the MDX engine of the dynamic query mode (DQM) server in each of the individual cells with which a calculated measure intersects. As the MDX engine evaluates a calculated measure expression, it might determine that it requires the value for one or more (non-calculated) measures. The MDX engine retrieves the values from the underlying dynamic cube, which might in turn cause the cube to retrieve the necessary data from the underlying relational database.

The presence of multiple calculated measures can result in a dynamic cube issuing a large number of nearly identical SQL queries to obtain data for each of the cells for each of the calculated measures. The overhead of executing a large number of these SQL queries leads to long report execution times.

In IBM Cognos BI version 10.2.2 Fix Pack 1, Cognos Dynamic Cubes can consolidate these multiple SQL queries into a smaller set of SQL queries that are run ahead of the execution of the MDX query, thus priming the data cache of the cube with the data necessary for the evaluation of the calculated measures. Cognos Dynamic Cubes can perform this optimization for some, but not all, calculated measures, depending upon the various functions and expressions used within the calculated measures.

The impact of this feature on performance can vary and is best ascertained by testing the performance of a particular workload with and without this feature enabled. As a consequence, this feature is disabled by default. The feature is controlled by two query service advanced properties:

- `qsEnablePrimingQuery`

  If set to `true`, the query service evaluates queries issued for execution to determine if they are suitable for this particular optimization. This evaluation incurs an overhead to query planning. If this feature is enabled, the goal is that the overall cost of this overhead is less than the gains obtained from the query optimization.

- `qsPrimingQueryThreshold`

  This advanced property is used to control the queries that are evaluated for the optimization. Its value is a number greater than 0, with a default of 10. If a query contains less calculated measures than the threshold value, the query evaluation is skipped entirely, thus limiting the number of queries that are affected by the query planning overhead of this optimization.

This particular optimization, when enabled, can result in a larger number of data values being placed in the data cache when compared with it being disabled. Consequently, consider allocating additional memory to the data cache when this feature is enabled.

Additional tracing is enabled for this feature. Set the event group `PrimingQuery` to `trace` level in the `configuration/xqe.diagnosticlogging.xml` file. This setting enables to write entries related to this feature to the `logs/XQE/xqelog-<timestamp>.xml` file.

11.9 Cache size effective practices

For more information about suggested practices when estimating cache sizes for a cube, see *IBM Business Analytics Proven Practices: Dynamic Cubes Hardware Sizing Recommendations* at:

http://ibm.biz/DynamicCubesHWSizingRec

The section about detailed sizing recommendations describes how to compute the minimum size requirements for each of the caches and describes other considerations in determining sizes for the caches. In general, having more CPU cores and allotting more memory for the data cache can improve performance of Cognos Dynamic Cubes.

Alternatively, see 17.2, “Using the hardware sizing calculator in IBM Cognos Cube Designer” on page 534.

11.10 Scenarios for performance tuning

Depending on the scenario, performance tuning can be done in areas, such as adjusting caches, leveraging aggregates, optimizing the cube model, and writing efficient reports.

11.10.1 Cases for adjusting cache sizes

The amount of memory per cube is the total of that required for the following items:

- Member cache
- Data cache
- Aggregate cache
- Temporary query execution space
- Additional space to reduce occurrence and cost of JVM garbage collection

This section describes cases in which the amount of memory that is associated with these elements requires adjustment of the cache sizes of the cube and possibly the overall query service JVM heap size.

Actual number of members is significantly larger than the estimated number of members

For information about estimating cache sizes for a cube, see *IBM Business Analytics Proven Practices: Dynamic Cubes Hardware Sizing Recommendations* at:

http://ibm.biz/DynamicCubesHWSizingRec

One approach to estimating cache sizes is to compute based on the size of the two largest dimensions of the cube.

Typically, the two largest dimensions of a cube are larger than all other dimensions by a large margin. If this case does not apply to your situation, use the actual number of members across all the large dimensions or even all the dimensions of the cube to calculate the size of the member cache. Then, use the new member cache value to recompute the other cache values and overall memory for the cube.

Use these same steps to recompute the member cache and other cache sizes if the dimensional space changes or grows significantly.

For more information about specifying the number of members in the tool to help estimate sizes, see 17.2, “Using the hardware sizing calculator in IBM Cognos Cube Designer” on page 534.

Number of users increases

The size of the data and aggregate caches is related to the dimensional space queried and user behavior. If the number of users increases, it might be necessary to account for users who are accessing data that is unique to their session. More of the dimensional space might need to be queried, or the nature of the reports might require processing of different amounts of data. Recompute the data cache size based on the updated number of named users for the user factor.

See 17.2, “Using the hardware sizing calculator in IBM Cognos Cube Designer” on page 534 for information about specifying the number of users in the tool to help estimate sizes.

Nature of reports and usage changes

In most cases, the size of the data cache and aggregate cache can be determined relative to the size of the member cache. The data cache and aggregate cache are dependent on the number of rows in the fact table when, relative to the dimensional space of a cube, there are many rows. Many rows means a dense fact table that requires more space to store non-null values.

The data cache retains data until it is nearly full and then discards data that is least accessed, because the data that users are interested in, or most likely to query, can change over time. If the nature of reports changes in such a way that data that would have been cached is discarded to make room for new data cache values, consider increasing the data cache to accommodate and retain more values.

Presence of the aggregate cache

The data cache size can be reduced by the presence of an aggregate cache. The reason is that an aggregate cache can retain many values that would have been stored in the data cache. If the aggregate cache contains aggregates that are recommended from analyzing query workload information, the aggregate cache should have more data that is relevant to the user. Consider recomputing the data cache for memory efficiency, accounting for the presence of the aggregate cache.

Virtual cubes are directly accessed but base cubes are not

If a cube exists solely for use within a virtual cube, and it is not accessible from any reporting packages, assigning a data cache to the cube might not be necessary because data is cached in the virtual cube that is being accessed. It is only important to cache data for the cubes that are accessible by users. Note that any cube, base or virtual, that is not accessible through a reporting package, might have little need for a data cache.

If a virtual cube is built to combine history and recent data into a single cube, assigning a data cache to the base cubes might make sense because this technique ensures fast query performance when the recent data cube is updated.

Virtual cubes do not have aggregate caches. Consider retaining the aggregate cache for the base cubes so that it can satisfy queries that are ultimately issued to the underlying base cube.

11.10.2 Cases for aggregates

This section describes how aggregates can enhance data warehouse and query performance.

OLAP-style reports that show data aggregated at levels higher than the grain of the fact

The expectation is that most OLAP-style reports and analyses that run against Cognos Dynamic Cubes show data at high levels of hierarchies, for example, report on the Gross

Profit for Retailers Region and Products Product Line. Use of aggregates can significantly improve performance of queries by providing data aggregated at levels higher than the grain of the fact.

By using both in-memory aggregates and database aggregates, data warehouses of all sizes, but especially medium, large, and extra large warehouses, can benefit from the greatly decreased query times.

The expectation is that most data warehouses already have database aggregates in some form: Either pre-existing aggregates for use by other applications, or aggregates added by the DBA to accelerate certain queries.

The process to identify relevant and useful database aggregates can be difficult and time-consuming. For data warehouses that do not yet have aggregates, or if you want to supplement existing database aggregates with in-memory and other in-database aggregates, run the Aggregate Advisor to get recommendations.

The DBA has the flexibility to choose which of the recommended database aggregates to implement, but can rely on output from the Aggregate Advisor that these aggregates are relevant. Also, after the modeler incorporates the aggregate cube information into the model, Cognos Dynamic Cubes can guarantee routing to these database aggregates.

The aggregate cache and the in-memory aggregates that it holds should be enabled, when possible. In-memory aggregates are meant to be a turn-key aggregate management solution. Recommendations that are provided by the Aggregate Advisor can be easily saved and loaded the next time the cube starts.

Small data warehouses might be able to provide improved query performance with only in-memory aggregates cache if the queries to the underlying database have sufficient performance.

Improving overall query performance

The intent is to improve the overall query performance across queries in a normal workload against Cognos Dynamic Cubes. Use of aggregates can improve performance for most user queries. However, not all queries need to be covered by aggregates. There might be some outlier queries that do not hit the aggregate cache or route to a database aggregate, because either there is no aggregate that can satisfy this query or it is not run often enough to justify having an aggregate to cover it.

In addition to the aggregate cache and database aggregate routing, the result set cache, expression cache, data cache, query pushdown, and other features of Cognos Dynamic Cubes also help to provide the fast query response times that users expect.

For new cube deployments, where the workload of queries is not known, run the Aggregate Advisor to get a set of recommendations to start with and apply to the cube data source. In the absence of workload information, the Aggregate Advisor uses its cube model-based analysis logic. Then, after a known or a sample workload is determined and captured, run the Aggregate Advisor again. This time, set it to consider workload information to get a new set of recommendations to use that are more relevant to actual user queries.

Over time, if overall query performance becomes less than satisfactory, it might be because the workload characteristics have changed. Recapture the workload information and rerun the Aggregate Advisor. See 11.3.6, “Rerunning the Aggregate Advisor” on page 366for more scenarios about when to rerun the Aggregate Advisor. Alternatively, consider enabling automatic optimization of in-memory aggregates that will continually monitor workload and automatically run the Aggregate Advisor in the background at certain intervals to adjust the in-memory aggregates. See 11.4.5, “Automatic optimization of in-memory aggregates” on page 375 for more information.

11.10.3 Optimizing the cube model

The cube model is the basis for all SQL queries generated, so consider optimizing your model before looking into database tuning or creating summary tables. This is an important and often overlooked area of cube performance.

Performance Issues tab in Cognos Cube Designer

When modeling in Cube Designer, you can see entries under the Performance Issues tab. These warnings are intended to alert you to modeling issues that might have room for improvement.

Calculated measures

The following is an example of a performance warning message for calculated measures:

```
The measure dimension of the cube Sales contains a calculated measure Product % of
Total Sales. Calculated measures should be restricted to expressions that use
functions which cannot be pushed to the database. Determine whether measures could
be modeled as a measure.
```

Evaluate the calculated measure expression. If the measure uses dimensional functions and cannot be rewritten as a direct expression against the database, you can ignore this warning for the measure.

Scalar expressions should be performed in the database and, if possible, during ETL, and not modeled in the cube. This approach reduces the work of queries to obtain data.

There are advantages to pushing the expression to the database, and there are advantages to having the expression as a calculated measure. Depending on usage of summary tables, the nature of the expression, and the expected usage of the expression, you might decide to pick one approach over the other. One advantage of keeping the calculated expression is that it can take advantage of summary tables that include the referenced regular measures.

Measure with regular aggregate of Calculated

The following is an example of a performance warning for any measures with `Regular Aggregate` value of `Calculated`:

```
The measure Profit Margin in the cube Sales has a regular aggregate of Calculated.
Queries at grains higher than the fact will perform slower than other
aggregations. Create aggregate cubes to handle these queries and avoid requiring
the routing of the query to the fact table.
```

Measures of this regular aggregate type are treated as non-additive measures. Summary tables can only be used if the table includes the calculated values and it is an exact match. For example, you can use the summary table if it has a `Profit Margin` column with the calculated values and it is an exact match. Summary tables cannot be used for higher level rollups of non-additive values. This is one of the trade-offs to consider when pushing a calculated member expression to the database.

Snowflake dimensions

The following is an example of a performance warning for snowflake dimensions:

```
The dimension Product is a snowflake dimension. Queries using snowflake dimension
perform slower than queries using dimension tables because additional table joins
are required in the generated SQL.
```

Typically, because having a snowflake schema is by design and there is no plan to normalize any of the dimension tables, you should ensure that the join columns are correctly defined in the model. Typically, the fewer joins in the SQL, the better. However, using a snowflake dimension has several advantages, so you can ignore this performance warning.

Level keys

The following are examples of performance warnings related to level keys:

```
The level key Quarter in the level Quarter in the dimension Time has an attribute
with a data type that is not Integer. Integers should be used as keys in order to
maximize performance.

The level Quarter in the dimension time uses a composite key for its level unique
key. Composite keys can affect performance by unnecessarily including columns in
the generated SQL. Ideally you would want to have a unique key for each level if
necessary by a surrogate key. You may need to trade-off that in order to unique
identify members, especially in slowly changing dimensions. Determine if all of
the keys are necessary to unique identify the members.
```

Level keys are an integral part of the cube model and getting them correct is crucial to the correctness and response time of the cube. Level keys are used to determine the uniqueness of members on a level and are used in every aggregation query pushed to the underlying data source.

It is optimal to have a single, numeric surrogate key for the level key. Performance warnings appear if the level keys are composite (that is, made up of more than one column) or are not integer or long columns. This is not to say that use of other numeric data types, such as decimal or floating point, are worse for performance. It will, however, depend on the underlying implementation of the relational database. In general, use of numeric data types is faster than use of string and character data types in relational databases, where integers are typically the most optimal.

To see how the modeled level keys affect the generated SQL, consider an example report for Total Sales in the first quarter of 2015. Example 11-14 shows the SQL for an example where the `Quarter level` key uses composite level keys and a column that is not an integer data type. Notice the additional columns, including a filter on a character column when using both `Quarter` and `Year` columns in the composite level key.

Example 11-14 Quarter level key using composite level keys and column is not an integer data type

```
SELECT
 TIME.YEAR,
 TIME.QUARTER,
 SUM(FACT.SALE_TOTAL)
FROM
 DW.TIME TIME,
 DW.SALES FACT
WHERE
  TIME.DAY_KEY = FACT.ORDER_DAY_KEY
 AND
```

```
  TIME.YEAR = 2015
 AND
  TIME.QUARTER = 'Q1'
GROUP BY
 TIME.YEAR,
 TIME.QUARTER;
```

Compare Example 11-14 on page 419 with the simpler SQL shown in Example 11-15 where the `Quarter level` key is a single integer column.

Example 11-15 Quarter level key is a single integer column

```
SELECT
 TIME.QUARTER_KEY,
 SUM(FACT.SALE_TOTAL)
FROM
 DW.TIME TIME,
 DW.SALES
WHERE
  TIME.DAY_KEY = FACT.ORDER_DAY_KEY
 AND
  TIME.QUARTER_KEY= 201501
GROUP BY
 TIME.QUARTER_KEY;
```

The following is a performance warning for a level key that uses an expression:

```
The level key Month Calculated Key in the level Month in the dimension Time has an
attribute with a data type that is not Integer. Integers should be used as keys in
order to maximize performance
```

In this case, the Month Calculated Key attribute is an expression: `Year * 100 + Month_number`. This case results in a single numeric value for the level key as opposed to using multiple attributes in a composite level key. However, this situation can actually lead to a less than optimal SQL at run time because the level key expression is included in multiple places in the SQL. Example 11-16 shows the SQL for a report for Total Sales in May of 2015:

Example 11-16 SQL example for report for Total Sales in May, 2015

```
SELECT
 TIME.YEAR * 100 + TIME.MONTH_NUMBER,
 SUM(FACT.SALE_TOTAL)
FROM
 DW.TIME TIME,
 DW.SALES FACT
WHERE
  TIME.DAY_KEY  = FACT.ORDER_DAY_KEY
 AND
  (TIME.YEAR * 100 + TIME.MONTH_NUMBER) = 201505
GROUP BY
 (TIME.YEAR * 100  + TIME.MONTH_NUMBER) ;
```

Note that use of attribute expressions is not all bad. If you use an expression to create a member caption or an attribute value, there will only be a one-time impact during metadata load. For example, use of a calculated expression in the attribute for `Day Caption` on the `Day` level. Level keys, however, are used in all data queries, so having expressions in level keys is not recommended.

Dimension filters

When creating a dynamic cube, the fact or dimension tables might have more data than you are interested in. For instance, the Time dimension table can be preloaded with a few decades worth of rows, whereas you are only interested in the last two years. Or, the product dimension might have products that have not yet been released that therefore have not sold any units. Not including extraneous dimension members can reduce the amount of memory and time required to load the metadata member cache and reduce the dimensional query space.

There are different options for reducing the scope of your dimensions or fact table. One option is to create a view on your dimension or fact table to filter the rows as appropriate in the database. Another option is to use dimension filters in the cube model. See 5.3.5, "Dimension filters" on page 127 for more information about defining dimension filters in Cognos Cube Designer.

11.10.4 Optimizing reports

Report authors can get faster report results by understanding how to construct an efficient OLAP report and how these reporting constructs affect the multidimensional queries sent to and processed by query service.

Writing efficient OLAP queries

For a proven practice document about report design that can be helpful as report writers transition from relational reporting to OLAP style reporting, see *Writing Efficient OLAP Queries* at:

http://www.ibm.com/developerworks/data/library/cognos/page128.html

For more information about dimensional reporting style and guidelines, see the *Cognos Business Intelligence 10.2.2 Product Documentation* at:

http://www.ibm.com/support/docview.wss?uid=swg27042003

In this document, see the topic *Relational and dimensional reporting styles* and Chapter 11 *Dimensional Reporting Style*.

Performance nuances of filters

Use of different report filtering constructs can have different performance characteristics because of the corresponding multidimensional functions used in the queries.

For different approaches to focusing or filtering dimensional data in a crosstab, see the document *IBM Cognos Report Studio User Guide, Version 10.2.2,* Chapter 11 *Dimensional Reporting Style,* section *Limitations When Filtering Dimensional Data Sources*. There are important limitations and possible alternative approaches to consider when using the context filters described in this document.

See also Appendix D, *Limitations when using dimensional data sources* section *Limitations When Filtering Dimensional Data Sources* for other limitations to be aware of when filtering dimensional data sources.

Validate your report and resolve any warnings after you understand and follow the guidelines for dimensional reporting. This approach is especially important for warnings related to the aforementioned limitations, as resolving these issues will likely result in improved performance.

Suppression

Large, sparse result sets consume a lot of memory in the engine and take a long time to run. Suppression is almost always a necessity for these large, sparse reports. This is generally a preferred practice to follow for reports so that the engine does not have to process a bunch of empty data points that do not exist.

By default, IBM Cognos Report Studio does not have any suppression options enabled. To suppress empty cells in Report Studio, complete these steps:

1. From the **Data** menu, select **Suppress** → **Suppression Options**.
2. Under Suppress, select what sections to suppress.
3. Under Suppress the following, select which values to suppress.

These suppression option settings are associated and saved on a per report basis. Creating a report in IBM Cognos Report Studio requires setting the suppression options for the new report independently.

If you want to enable query-specific null suppression, you can set the `Suppress` query hint to `Nulls` in the selected properties of the query in Report Studio. This technique uses the NON EMPTY clause in the MDX query posed to dimensional data sources.

The null, zero, error, and NA suppression options on rows and columns available in Cognos studios can be applied in addition to the suppression that this query hint applies.

For more information about suppression in Report Studio, see *Suppress Empty Cells* in the document *IBM Cognos Report Studio User Guide, Version 10.2.2*.

For more information about suppression in IBM Cognos Dynamic Query, see *5.5 Suppression* in *IBM Cognos Dynamic Query*, SG24-8121.

11.10.5 Performance and the data warehouse

As mentioned throughout this book, a properly designed data warehouse is a recognized best practice to provide a well performing experience for users. As such, a properly designed data warehouse with a star or snowflake schema is a requirement to use Cognos Dynamic Cubes.

Data warehouse considerations

Database views can be used for dimensions and fact tables. However, they should be used only if it is absolutely necessary because performance of views is dependent on the relational database implementation. Cases for using views include versioned or filtered hierarchies.

For optimal database performance, use integer level and surrogate keys and avoid using other numeric and character types.

See “Level keys” on page 419 for details about the correlation between keys and SQL performance.

Tuning the data warehouse is essential to performance because cold cache queries (that is, queries that miss the caches) are performed by the database. Concepts such as data and join indexing, partitioning, and summary tables are still important in traditional data warehouses for performance.

To avoid having large, sparse sets of data that are difficult to optimize across a set of users, avoid large dimension tables with few levels, and avoid a large number of dimensions.

Also avoid a *coded record* fact table that increases the number of fact records and adds costly CASE logic to SQL queries.

Although referential integrity is usually not enforced in the database to allow faster inserts during ETL, and is not required for modeling, be aware of the possible impact to data quality. Problems with referential integrity affect the quality of the data in summary tables, in-memory aggregates, and data caches. See 6.6, “Data quality and integrity” on page 185 for steps to detect referential integrity issues in the database.

ETL window impacts

Adding a dynamic cube to a data warehouse requires the following operations to be performed within a pre-existing, fixed size ETL window:

1. Stop the cube.
2. Update or re-create database aggregate tables.
3. Start the cube (that is, load dimension members)
4. Load in-memory aggregates.
5. Run priming queries.

Depending on the type of data warehouse changes to be updated in the cube, leveraging the near real-time capability in the dynamic cube can reduce ETL window impact. See Chapter 12, “Near real-time updates” on page 427 for more information about this topic.

The Aggregate Advisor might suggest a number of aggregate tables and in-memory aggregates. Due to fixed ETL windows time constraints, it might be necessary to create a subset of these aggregates or take advantage of all available techniques to allow the creation of as many aggregates as needed. For example, consider stacked aggregates where aggregates can derive from one another. See 11.5.5, “Database aggregates tips and troubleshooting” on page 390 for more information.

Using the query service log to understand where time is spent

This section describes advanced troubleshooting techniques based on using more verbose logging levels and examining the query servicelog to see where time in the query service is spent during queries that go to the database.

To generate some insight about database queries and query performance, enable the logging event groups specifically for these activities:

1. On the query service system, edit the DQM logging configuration, which is located relative to the server installation directory:

   ```
   configuration\xqe.diagnosticlogging.xml
   ```

2. Edit the following event groups and log levels:

   ```
   <eventGroup name="ROLAPQuery.AggregateStrategy" level="trace"/>
   <eventGroup name="QueryService.SQL" level="trace"/>
   <eventGroup name="ROLAPQuery.Performance" level="info"/>
   <eventGroup name="Timing" level="info"/>
   ```

3. Save the `xqe.diagnosticlogging.xml` file.

4. It is not required to restart the query service for the diagnostic logging changes to take effect. Wait for the logging refresh interval default of 30 seconds.
5. Start the cube.
6. Run the problem query.
7. By default, the query service log is written to a file, which is located relative to the server installation directory:

   `logs\XQE\xqelog-<timestamp>.xml`
8. When troubleshooting by using the query service log is complete, revert the changes to the log level in the `xqe.diagnosticlogging.xml` file.

The `ROLAPQuery.AggregateStrategy` log entries identify when an aggregate table is being used for a query. See “Using the query service log to understand database aggregate routing” on page 392 for details.

The `QueryService.SQL` log entries show the SQL generated to the database. Note that the `SQL query completed.` log entry is the time that the database took to acknowledge receipt and prepare the query. It does not include actual query execution time.

For database query execution time and data retrieval time, look at the `ROLAPQuery.Performance` log entries. The `Finished data query.` log entries indicate the time the database took to run the query and is ready to return the first row. This timedoes not include the data retrieval times to fetch the data from the database.

Data is consumed through a pipeline of threads, so the time to retrieve data is overlapped with MDX query processing. Even so, the query retrieval time is still a good indicator of time to retrieve data from the database. The `Finished iterating result set for data query` log entries help understand how many values were requested and retrieved, and provide some insight in how long it took to retrieve the data. Look for queries that return a lot of rows of data from the database.

For an indication of where and how much data was found or fetched for a query, see `ROLAPQuery.Performance` log entries, `"Finished execution of Query Strategy for report"`.

For a summary of the total requests, see `Timing` log entries, `Total request time`. The `Total request time` is followed by a break-down of the time spent in virtual and base cubes activities, how much data was processed, and SQL statement summary of the number of SQL calls, time to execute, fetch, discard, and how many values this resulted in the cube.

For more information about enabling logging to diagnose issues, see *IBM Business Analytics Proven Practices: IBM Cognos Dynamic Cube Advanced Logging Settings* at:

http://www.ibm.com/developerworks/library/ba-pp-infrastructure-cognos_specific-page655/index.html

11.10.6 Other possibilities for slow queries

After optimizing the data warehouse, loading recommended aggregates and caches, and following suggested practices for the cube model and reports, you might still experience some instances of suboptimal performance. This section highlights several possible problems and solutions for slow queries.

Insufficient CPU cores

If database performance is good, database aggregates are being used, and in-memory aggregates available are being used, but the performance is still slow, it might be due to not enough CPU cores, whether it be physical or 100% allocated virtual cores. CPU cores are used to allow user queries to run concurrently and to perform specific tasks within a query in parallel.

Insufficient heap space for the number of concurrent users

With a number of concurrent users, if not enough physical heap space is available, it results in long GC pauses where all activity is paused. Deploying the cube across multiple servers to spread the load of the concurrent users might help in this situation.

Data cache is too small

When the data cache hits a certain threshold, it will remove least-recently used data to make room for new data. If the cube metrics show data cache memory usage is high compared to defined space, and queries are slow, it may be an indication that the data cache is thrashing. Increase the heap size and data cache size, or consider partitioning users if they access disjoint portions of a cube.

Control the amount of data retrieved by database queries

If you see that the SQL to the database is requesting more members or more measures than your report actually contains, it is due to a couple of thresholds in the cube properties. These thresholds are a speculative pre-fetch of data to include additional members or additional measures in the SQL. However, if you find that this behavioris not what you expect or want, you can update these thresholds for the cube as follows:

1. Go to IBM Cognos Administration.
2. Select **Status**.
3. Select **Dynamic Cubes**.
4. Select **Data Stores (All)**.
5. Select the cube.
6. Select the server group.
7. Select the **QueryService**.
8. In the **QueryService** menu, select **Set properties**.
9. To control the percentage of members in a level that will be referenced in a filter predicate, enter a value for the `Percentage of members in a level that will be referenced in a filter predicate` property. A reference to the containing level will be used in the filter predicate if the percentage of members exceeds the value specified by this property. The value must be between `0` and `100`. Specify `0` if no limit is required, meaning that the filter predicate will always retrieve measure values for all of the members at the level. Specify `100` to never retrieve measure values for all members at the level unless all members, i.e., 100% of members at the level, are specified in the report query.
10. To control the percentage of measures that must be exceeded before the SQL query generated retrieves all measures, enter a value for the `Measures threshold` property. Calculated measures, non-visible measures, and semi-aggregate measures are not included. The value must be between `0` and `100`. Specify `0` to always include all measures in SQL queries. Specify `100` to include all measures only if all measures, i.e., 100% of applicable measures, are specified in the report query
11. Click **OK**.

If the cube is deployed to multiple server groups and multiple instances of the query service, then each cube instance has its own set of cube properties.

Large spikes in memory usage

Large spikes in memory usage reported by the JVM metrics of the query service in the Metrics - QueryService pane in IBM Cognos Administration or the JVM process from the operating system resource monitor, are typically a symptom of users performing large data extract queries that cross join multiple levels, especially at deep, low levels.

Processing of such queries require intermediate sets in the query service to consume large portions of the JVM heap.

Educating the users to enable null suppression can significantly reduce the number of values in intermediate result sets. See 11.10.4, "Optimizing reports" on page 421 for details.

As an administrative precaution, you can set the query service advanced setting `qsMaxCrossjoinOrderOfMagnitude` to prevent users from running large reports that are not optimal and could use more memory than it is available.

This setting stops a query from running when the MDX engine of the query service detectsthat the report requires processing of more tuples than what is allowed. A tuple is data at the intersection of two or more dimensions.

If a report requests more tuples than the limit, an error is generated and all memory associated with that request is released. For more information, see the IBM technote *Preventing large queries from exceeding memory capacity by specifying a tuple limit* at:

http://www.ibm.com/support/docview.wss?uid=swg21691156

12

Near real-time updates

This chapter describes the near real-time updates capability that is built into IBM Cognos Dynamic Cubes and how it is applied.

This chapter introduces the concept of how to make new data available to information consumers while avoiding dynamic cube downtime. This capability avoids loss of cached information, providing users with near real-time access to the latest data along with the benefits of fast, interactive queries on large data volumes.

The intent of this chapter is to provide an overview of the near real-time capability and demonstrate how it is applied and managed.

This chapter contains the following sections:

- Near real-time updates overview
- Modeling near real-time updates
- Applying near real-time updates

12.1 Near real-time updates overview

IBM Cognos Dynamic Cubes delivers exceptional performance on large data sets by using a mixture of technologies and approaches. These include intelligent hybrid queries, in-memory processing, aggressive caching, and aggregate awareness where aggregates are stored both in the database and in-memory.

One of the challenges when using techniques such as caches and aggregates is managing the timeliness of data. As data becomes stale, so can its value in the decision-making process.

In some cases, being able to refresh the data on a weekly or daily basis is acceptable, but there are some use cases where it is not acceptable.

> **Example:** A financial institution requiring near real-time updates capabilities
>
> An investment bank usually holds large volumes of trade data that they analyze and look for trends, which is a good use case for a dynamic cube. The analysis often looks at areas such as reversals of trades and profitable traders. There is an increasing desire to have up-to-the-minute information that provides the organization a competitive edge. It also can be seen as providing traders with a high performance environment where they can perform queries and make decisions.

Without the near real-time updates capability, a dynamic cube must have all its dependant objects refreshed with new data. This approach involves adding the data to the main fact table, updating in-database caches, and then reloading the dynamic cube in-memory aggregates. At the same time, the user caches designed to improve performance are all invalidated. In this case, the user can experience either inconstant performance with lags on refresh or even query downtime.

As the velocity of information increases alongside the volume of data, it is not acceptable for a system to go offline while data is recalculated or refreshed. For this reason, IBM added near real-time capability to IBM Cognos Dynamic Cubes.

The near real-time capability adds an understanding of new data to a dynamic cube in the form of a transaction identifier (TID) where each new set of data can be made available to the dynamic cube at a time. The system understands, for a query, that the data up to the current, known TID must be retrieved from the base database table whereas the historic and already queried, new data (prior TIDs) can be acquired from the various caches.

Example: Query logic as new data is added

Cognos Dynamic Cubes keeps track of a TID so that it knows of and ensures that the queried data is valid up to and including that TID. The cube administrative command **Incrementally update data** (you can run this command from the DCAdmin command-line tool `incrementallyLoadCubes` or by using the GUI as shown in Figure 12-10 on page 437) must be started before the system adds new data to the query results.

In the case where a user requests the Total sales value for each Order method:

1. The system retrieves the total value for each Order method from an in-memory aggregate.
2. Some new data is added with a TID of 1 and the dynamic cube is notified by using the incrementally update data command that 1 is the current TID (as described in 12.3, "Applying near real-time updates" on page 436).
3. When the same request for Total sales by Order method is issued, the system queries for the Sum of sales grouped by Order method where the TID <=1 and the TID is not Null (that is, the new data). The results are placed in cache with a key of 1. The value is added to the prior cached, in-memory result to return the Total sales for the request (that is, historic data and new data).
4. If the user requests the same data again, the result can be returned entirely from cache without the need to query the fact table for new data. If more data is added and the incrementally update data command is run, for example up to TID 5, then the system adds the new data to cache by retrieving the data needed for the query and filtering the fact table for new TIDs. In this example, TID >1 and TID <=5 and TID is not NULL.

The system intelligently tracks the data results or *-tuples* that are available for new data in cache, removing the need to query again the underlying fact table. The system must retrieve the data only when new data (new TIDs) is added or a query for data not in cache is performed. This approach, combined with the other performance techniques and the historic data already in various caches, helps to ensure a high level of performance while new data is added.

One potential performance issue might arise if the new data volumes are so large that the initial queries to retrieve new data take too long. The impact on the initial query depends greatly on the underlying relational database management system (RDBMS) technology and infrastructure as well as the type of query and optimization, such as indexing, in place.

In most modern RDBMS production warehouses, even adding millions of data rows should not have a large negative impact on the system. If adding data has a negative impact, techniques such as regular seeding queries (a report is run after a TID update to incorporate the new data into cache) can help ensure a fast user experience.

Notes:

- IBM Cognos Dynamic Cubes was able to provide near real-time updates in prior versions by using a virtual cube technique. Although this technique allows for cross database amalgamation of both historic and new data, it does so at the expense of query performance because all of the new data is required each time. For more information about this approach, see the IBM developerWorks® article *IBM Business Analytics Proven Practices: Dynamic Cubes - Near Real-Time and Cross Database Queries* at:

  `http://www.ibm.com/developerworks/library/ba-pp-performance-cognos_specific-page690/index.html`

- Currently, near real-time updates are limited to new fact rows only. It is not possible to apply near real-time updates to updated or deleted rows in the fact table (although deltas can be added). New, updated, or deleted rows in dimension tables require a dimension refresh. Other unsupported items for near real-time are `Measures with Custom (Unknown)` aggregate types, and virtual cubes with the data cache and result set cache enabled.

12.2 Modeling near real-time updates

This section details the requirements for the underlying database and how to enable the near real-time update capability in a new or existing model.

In this book, the screen captures and examples are from the samples that are delivered with IBM Cognos Business Intelligence 10.2.2, in particular the `gosldw_sales` cube sample. For more information about the samples in Cognos BI 10.2.2, see *Samples* at:

`http://www-01.ibm.com/support/knowledgecenter/SSEP7J_10.2.2/com.ibm.swg.ba.cognos.inst_cr_winux.10.2.2.doc/c_instsamplesoverview.html?lang=en`

12.2.1 The database structure

As mentioned in 12.1, “Near real-time updates overview” on page 428, the underlying database requires a field that can be used as a TID. The data type for the column can be set to any type that supports SQL comparison operators and the SQL MAX function. It is advisable to use the BIGINT, INTEGER, or TIMESTAMP data types. If the frequency or volume of updates is high, it might be beneficial to use an index or other RDBMS optimization technique on the TID.

The fact table should include a TID column as shown in Figure 12-1.

NRT_FACT_EXAMPLE

TID	ORDER_DAY_KEY	ORGANISATION_KEY	ORDER_METHOD_KEY	QUANTITY	UNIT_COST
	20140703	40	1	1000	10.99
	20140703	34	3	1300	9.99
	20140703	4	2	1200	9.98
1	20140704	110	4	1400	10.99
1	20140704	420	1	1600	30.99
2	20140704	140	2	1300	5.99
2	20140704	340	4	1200	23.99
2	20140704	240	6	1600	1.99

Figure 12-1 Example of a fact table that contains measures, keys, and TID

After it is added, the TID is available for modeling in cube designer.

12.2.2 Enabling near real-time updates in the cube model

This section provides information about enabling near-real time updates in the cube model.

The cube should be modeled following standard practices that are described in 4.1, "Introduction to Cognos Cube Designer" on page 90. For additional information and guidelines, see the *Cognos proven practices documentation* at:

http://www.ibm.com/developerworks/topics/cognos%20proven%20practices%20documentation/

Also, the *Dynamic Cubes User Guide* can be downloaded from the *Cognos Business Intelligence 10.2.2 Product Documentation* website at:

http://www-01.ibm.com/support/docview.wss?uid=swg27042003

To enable near real-time updates in the cube model, complete the following steps:

1. Refresh the metadata by clicking the **Refresh** icon or right-click the connection in the metadata pane as shown in Figure 12-2.

> **Note:** In this example, a database administrator has added a TID to the underlying database object and therefore the modeler only has to refresh the metadata.

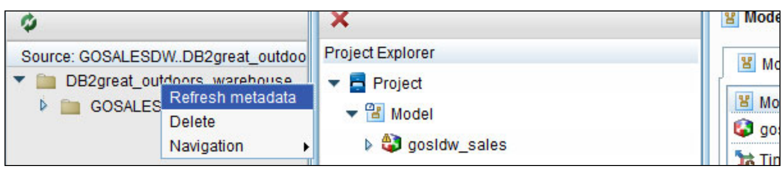


Figure 12-2 Refresh metadata

2. After the metadata is refreshed, navigate to the Measures folder for the cube (in this case the `gosldw_sales` cube).
3. Double-click **Measures**.
4. In the Properties tab select the column that you want to use as the Transaction ID column (**TID** in Figure 12-3).

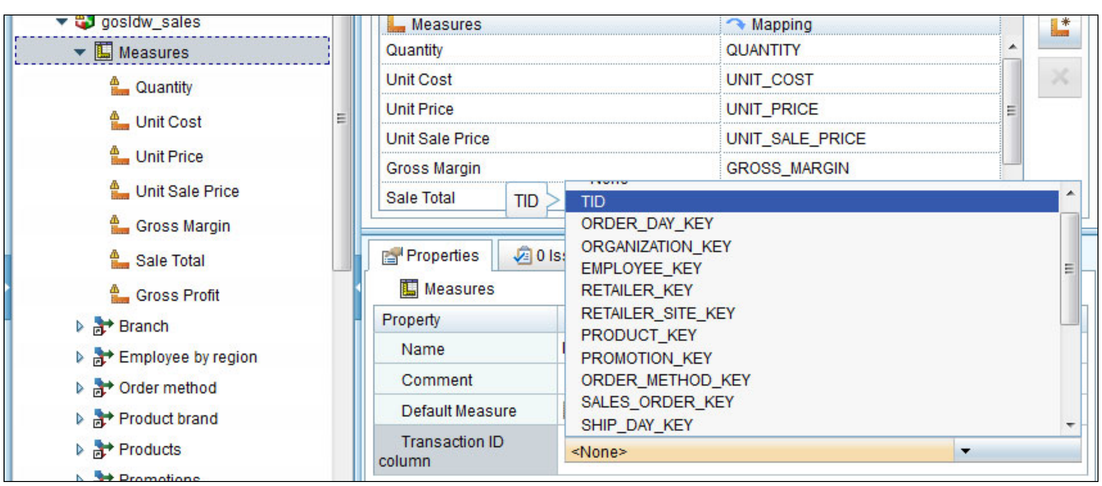


Figure 12-3 Setting the transaction ID column

5. Republish the cube model and restart the cube as shown in Figure 12-4.

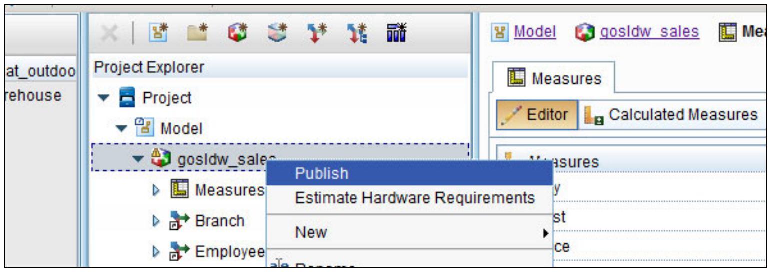


Figure 12-4 Publishing the cube

> **Note:** After modeling the cube and adding the transaction ID in the properties, other operations such as running the aggregate advisor, and starting and stopping cubes are performed as usual.

12.2.3 Adding near real-time updates to an existing cube

It is likely that, for an existing system, the underlying database might be missing a suitable column to use as the TID. Existing ID or transaction date columns are rarely suitable given that historic rows should be `NULL`. In this scenario, there might be reluctance to change the base table to accommodate the TID.

One option is to use a new table (identical to the main fact table, but with the addition of the TID column) and a view to unite the results. This approach is especially suitable for RDBMS that support and are optimized for partitioned views. The reason is that queries that specify a `NOT NULL` predicate against the TID avoid the historic data completely, which improves performance. Consult with your database administrator (DBA) because there might be other, more efficient means of adding the TID column.

In the case of IBM DB2 with BLU Acceleration, the partitioned view approach works well. It allows the historic data to use columnar storage while the new data uses row storage. This method improves new data insert speed and allows for a batch update of the columnar fact table.

For the `gosldw_sales` cube sample database, the script detailed in Example 12-1 can be used to create the data table with the TID column and the view. This example uses DB2 syntax designed for the DB2 based GS_DB sample database. Because this is a sample data set, the volumes do not warrant other optimizations such as indexes. Seek guidance on the appropriate approach for your chosen database.

Example 12-1 New table and view syntax for the DB2 samples

```
-- Create an extra table for new data

DROP TABLE GOSALESDW.NRT_SLS_SALES_FACT;

CREATE TABLE GOSALESDW.NRT_SLS_SALES_FACT (
      TID IntegerNULL,
      ORDER_DAY_KEY INTEGER NOT NULL,
      ORGANIZATION_KEY INTEGER NOT NULL,
      EMPLOYEE_KEY INTEGER NOT NULL,
      RETAILER_KEY INTEGER NOT NULL,
```

```
        RETAILER_SITE_KEY INTEGER NOT NULL,
        PRODUCT_KEY INTEGER NOT NULL,
        PROMOTION_KEY INTEGER NOT NULL,
        ORDER_METHOD_KEY INTEGER NOT NULL,
        SALES_ORDER_KEY INTEGER NOT NULL,
        SHIP_DAY_KEY INTEGER NOT NULL,
        CLOSE_DAY_KEY INTEGER NOT NULL,
        QUANTITY BIGINT NOT NULL,
        UNIT_COST DECIMAL(19 , 2),
        UNIT_PRICE DECIMAL(19 , 2),
        UNIT_SALE_PRICE DECIMAL(19 , 2),
        GROSS_MARGIN DOUBLE NOT NULL,
        SALE_TOTAL DECIMAL(19 , 2),
        GROSS_PROFIT DECIMAL(19 , 2)
   )
   ORGANIZE BY ROW
   DATA CAPTURE NONE
   COMPRESS NO;


-- Create the near realtime view
Create View GOSALESDW.NRT_Sales_Fact as
SELECT
   NULL as TID,
   ORDER_DAY_KEY,
   ORGANIZATION_KEY,
   EMPLOYEE_KEY, RETAILER_KEY,
   RETAILER_SITE_KEY,
   PRODUCT_KEY,
   PROMOTION_KEY,
   ORDER_METHOD_KEY,
   SALES_ORDER_KEY,
   SHIP_DAY_KEY,
   CLOSE_DAY_KEY,
   QUANTITY,
   UNIT_COST,
   UNIT_PRICE,
   UNIT_SALE_PRICE,
   GROSS_MARGIN,
   SALE_TOTAL,
   GROSS_PROFIT
  FROM GOSALESDW.SLS_SALES_FACT
UNION ALL
SELECT
   TID,
   ORDER_DAY_KEY,
   ORGANIZATION_KEY,
   EMPLOYEE_KEY, RETAILER_KEY,
   RETAILER_SITE_KEY,
   PRODUCT_KEY,
   PROMOTION_KEY,
   ORDER_METHOD_KEY,
   SALES_ORDER_KEY,
   SHIP_DAY_KEY,
   CLOSE_DAY_KEY,
```

```
  QUANTITY,
  UNIT_COST,
  UNIT_PRICE,
  UNIT_SALE_PRICE,
  GROSS_MARGIN,
  SALE_TOTAL,
  GROSS_PROFIT
 FROM GOSALESDW.NRT_SLS_SALES_FACT
```

After it is applied, the cube model must be updated.

It is currently not possible to change the base fact table for a cube. However, instead of re-creating the complete cube definition, you can update it manually as follows:

1. Refresh the metadata as shown in Figure 12-2 on page 431.
2. Remove all existing measures from the existing cube (see Figure 12-5). Before doing so, you should make a backup of the model. Note that any members calculated based on the deleted measures will be lost and will have to be re-created.

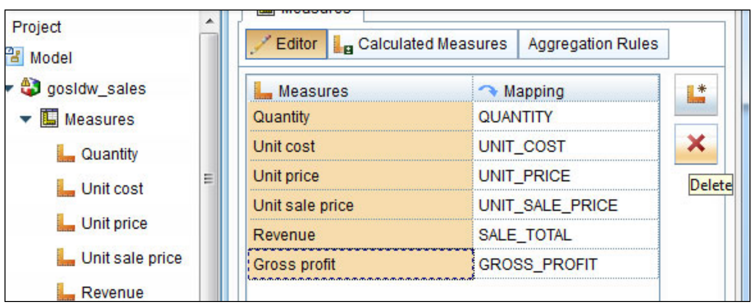


Figure 12-5 Deleting existing measures

3. Drag the new measures from the view into the `Measures` folder (see Figure 12-6). Check that the names are the same as the old names and update any calculated items that are based on the old measures.

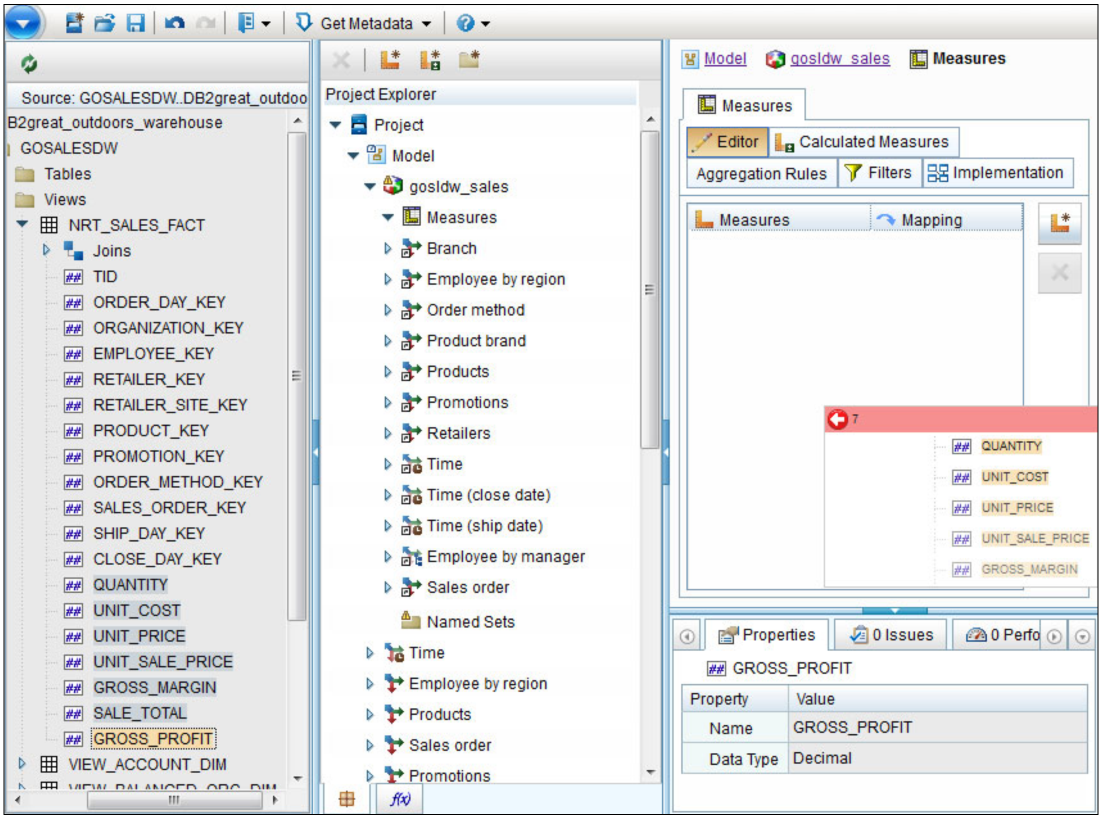


Figure 12-6 Adding view measures to the cube

4. Set the focus on the cube so that you can see the relationship editor (see Figure 12-7).

Figure 12-7 Cube relationship editor

5. Click **Edit** for the first relationship (you will see an error for the existing relationship as shown in Figure 12-8).

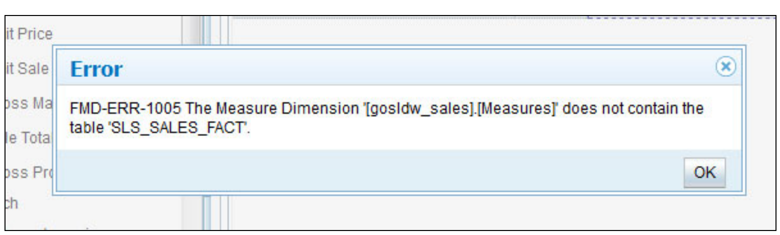


Figure 12-8 Invalid relationship error

6. Correct the column mapping for the new fact view (see Figure 12-9).

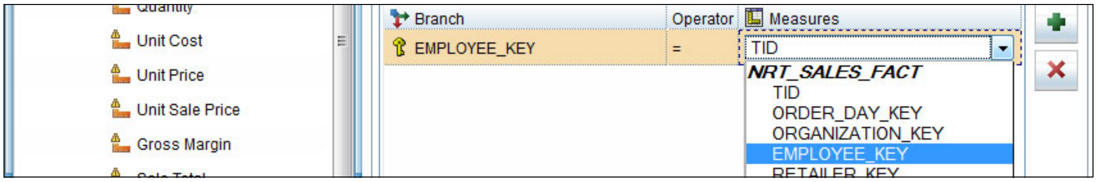


Figure 12-9 Setting the new relationship for the fact view to the dimension

7. Repeat step 5 and 6 until all dimension relationships are fixed.
8. If you have aggregate cubes, you also must redefine the measures for them. Relationships for aggregates that contain the keys should not have to be updated.
9. Check for errors. Use the Issues tab at the cube level to determine whether you missed any items that require changes.
10. Update the TID column as shown in Figure 12-3 on page 431.
11. Save the model, then publish it as shown in see Figure 12-4 on page 432.

> **Note:** A simpler and less time consuming approach is to ensure that the view has the same name as the original table. Alternatively, alter the model by opening the `.fmd` model file in a text editor, search and replace the old fact table name with the new one, then save and open the file in Cognos Cube Designer again ready to check and publish. Use caution when altering the model. It is strongly advised that you back up the `.fmd` file before you change it.

12.3 Applying near real-time updates

Applying the near real-time updates to an existing cube that is running is a simple matter of notifying the query service that a new TID is ready for that cube.

After the dynamic cube is notified of the new TID, any subsequent queries acquire data for information not held in cache by running direct SQL statements against the database. These queries filter the query using the TIDs that exist between the last cached version and up to the new TID provided.

> **Note:** Queries for new data are performed as the new request is received, not when the system is first notified of the new TID. The query service only satisfies requests for new data and it does not preinstall all new rows.

12.3.1 Manually incrementing data

The steps to manually increment data are as follows:

1. In IBM Cognos Administration, select the Status tab and click **Dynamic Cubes**.
2. In the Scorecard pane, from the list next to the cube for which you want to increment the TID, select **Incrementally update data** as shown in Figure 12-10.

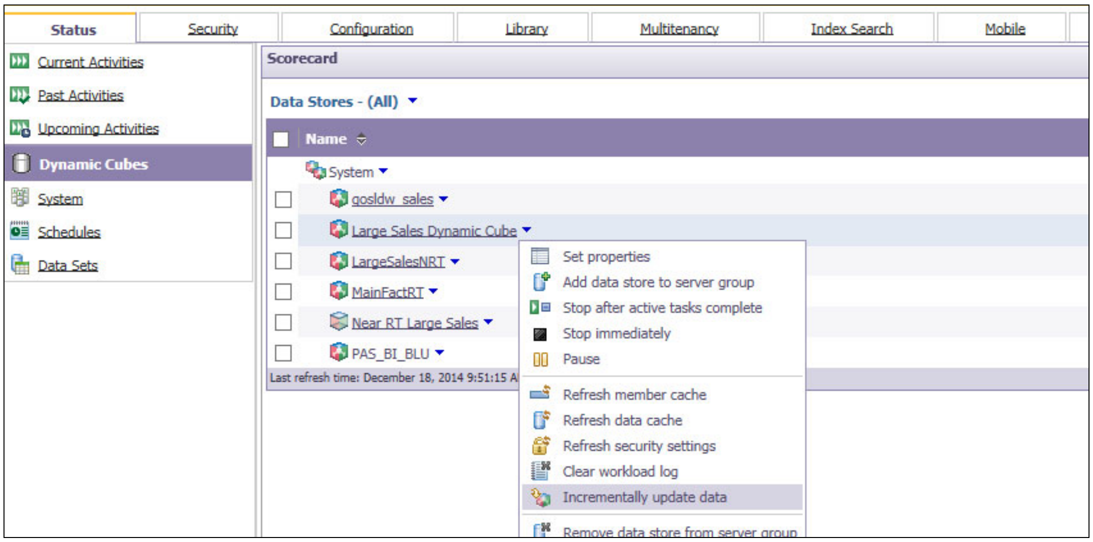


Figure 12-10 Incrementally update data

3. Status should be `Succeeded` as shown in Figure 12-11.

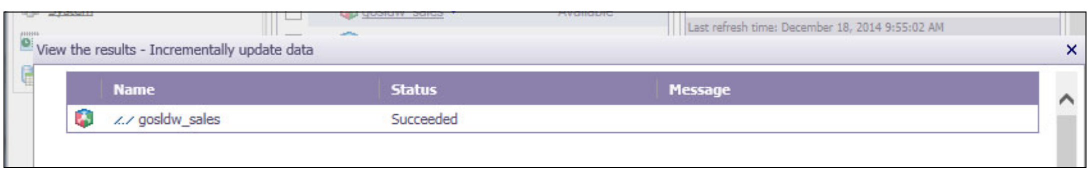


Figure 12-11 Increment of TID success message

4. The current value of the TID can be displayed by drilling-down on the cube to a chosen dispatcher and looking at the cube metric for **TID Value of latest increment** as shown in Figure 12-12.

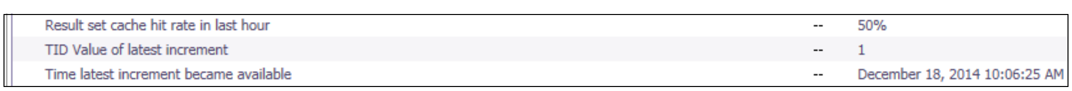


Figure 12-12 Current TID value state

Note: If the `TID Value of latest increment` is `Null`, then there are no TID rows with a value or the cube has not had a TID column specified in Cognos Cube Designer.

The manual process shown in Figure 12-10 consists essentially of selecting the highest TID column from the fact table or view.

If large volumes of new rows are expected, pay special attention to indexing the TID column and to the frequency at which the new rows change their TID value. If the process of inserting new data is manual, then the risk for errors is low.

However, if the new data is added programmatically, there is a risk that the cube TID is incremented and a user queries data for the new TID while new rows for this TID (the new TID) are still being added to the underlying table. This situation causes incorrect results and might also produce reports that have summaries that differ from the aggregated detail data. This occurs because the summary comes from the cache and the details come from a new query (which includes the extra TID rows that are not supplied by the cache).

A manual refresh of the TID value should be limited to where the new TID value for data is loaded before incrementing, or where the value is incremented for every batch insert, that is, with an integer or a time stamp that includes milliseconds.

To test this process on the samples data, use the `Profit by Product Brand` sample report provided to validate changed data before incrementing the TID (see Figure 12-13) and after incrementing the TID (see Figure 12-14 on page 439).

Use the SQL statement shown in Example 12-2 (DB2-specific) to add new data for 2014 with a TID of `1`. The SQL statement in the example simulates new data by replicating part of the historic data set while adding a TID and incrementing the ORDER_DAY_KEY.

Example 12-2 SQL statement to add new data with a TID of 1

```
INSERT INTO GOSALESDW.NRT_SLS_SALES_FACT
 SELECT
   1,
   ORDER_DAY_KEY +100,
   ORGANIZATION_KEY,
   EMPLOYEE_KEY, RETAILER_KEY,
   RETAILER_SITE_KEY,
   PRODUCT_KEY,
   PROMOTION_KEY,
   ORDER_METHOD_KEY,
   SALES_ORDER_KEY,
   SHIP_DAY_KEY+100,
   CLOSE_DAY_KEY+100,
   QUANTITY,
   UNIT_COST,
   UNIT_PRICE,
   UNIT_SALE_PRICE,
   GROSS_MARGIN,
   SALE_TOTAL,
   GROSS_PROFIT
   FROM GOSALESDW.SLS_SALES_FACT
 where ORDER_DAY_KEY = 20140720;
 COMMIT;
```

Figure 12-13 shows the `Profit by Product Brand` sample report provided to validate changed data before incrementing the TID.

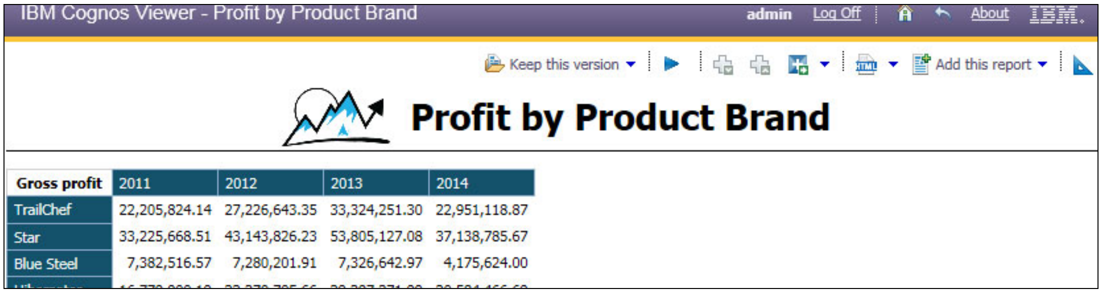


Figure 12-13 Profit By Product Brand report before incrementally updating data and the TID

Figure 12-14 shows the `Profit by Product Brand` sample report provided to validate changed data after incrementing the TID.

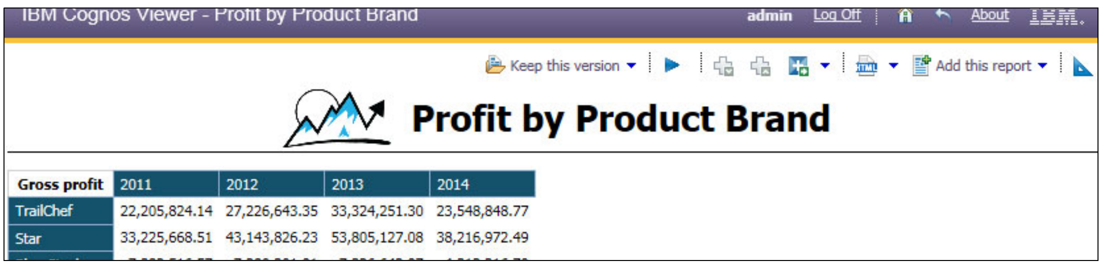


Figure 12-14 Profit By Product Brand report after incrementally updating data and the TID

12.3.2 Scheduled incremental updates

Besides incrementing the TID manually, it can also be incremented on a scheduled basis. Because incrementing the TID by using a schedule selects the highest TID from the base data, care should be taken to ensure that subsequent TIDs are not inserted before the last updated value for the system (see Figure 12-12 on page 437). Doing so causes incorrect values in reports and analysis. To avoid this issue, each commit of new data should increment its TID value in the data table.

To schedule a task to increment the TID, complete the following steps:

1. Create an administration task:
   a. In IBM Cognos Administration, select the Configuration tab and click **Content Administration**.
   b. From the **New Query service administration task** list, select **Dynamic cube** (see Figure 12-15).

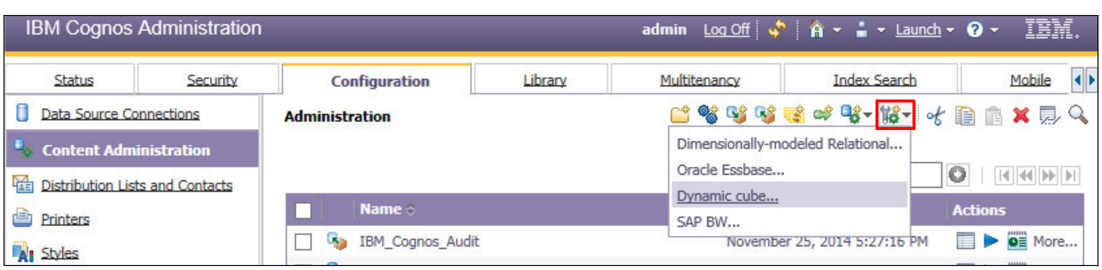


Figure 12-15 New Query service administration task list

c. Give a name to the new administration task (see Figure 12-16) and, optionally a description. Click **Next**.

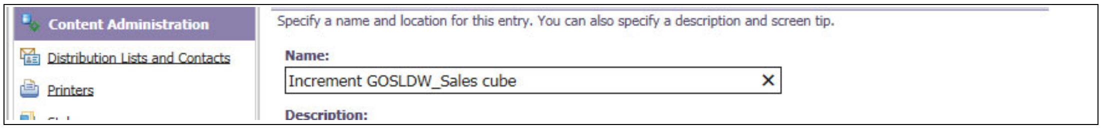


Figure 12-16 Name the administration task

d. In the New Query Service Administration Task wizard window,select **Incrementally update data** and select the cubes to apply the TID update to, the server group, and dispatcher as shown in Figure 12-17.

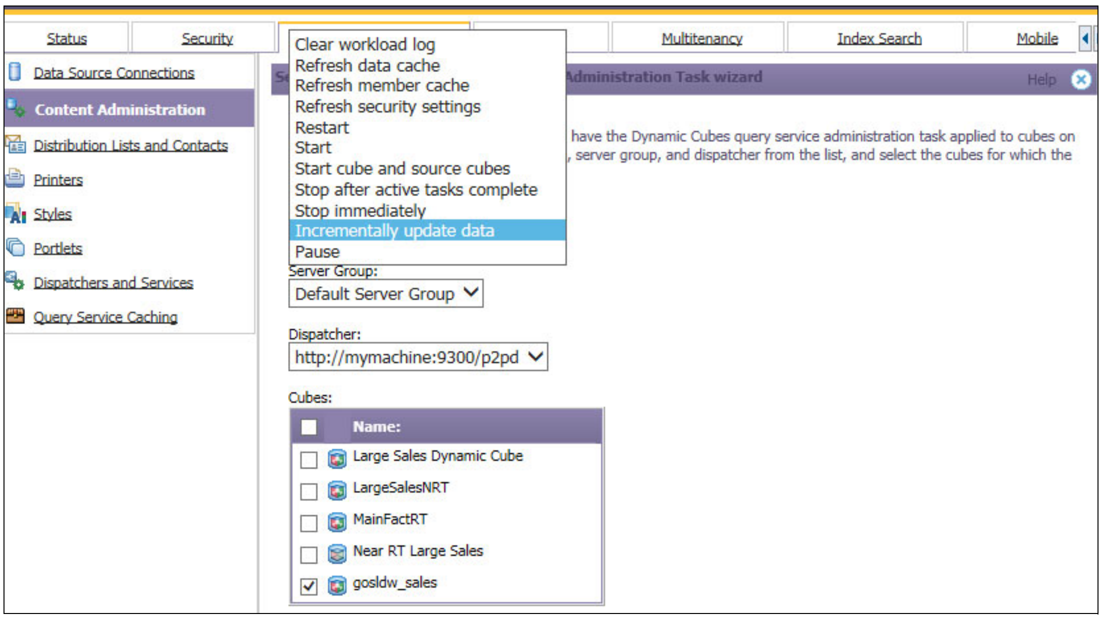


Figure 12-17 New Query Service Administration Task wizard

2. Schedule the task to run automatically.

a. Click the **Schedule - Increment GOSLDW_Sales cube** icon (see the icon next to More in Figure 12-18).

Figure 12-18 Content Administration list view

b. Set the scheduling preferences using the standard IBM Cognos schedule options (see Figure 12-19 on page 441). There are several options such as:

- Schedule based on frequency (daily, weekly, monthly, and so on).
- Schedule based on an external trigger (using the command line or a shell script)
- Schedule based on a start and end date and time.

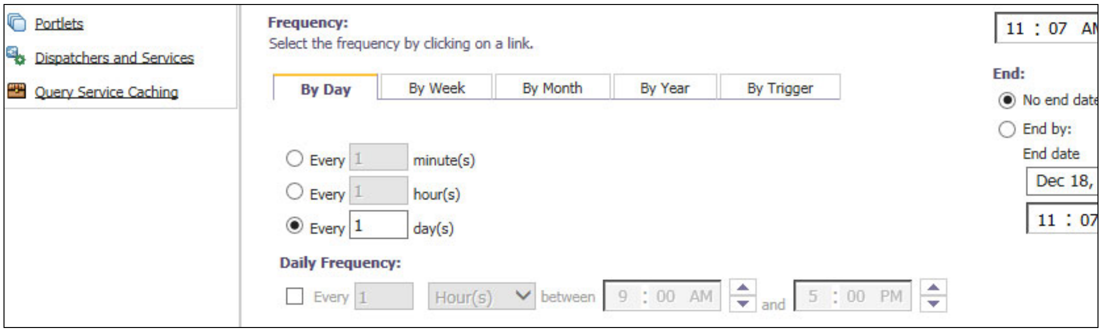


Figure 12-19 Scheduling options window

The current TID value for a cube can be viewed by looking at the cube metric for `TID value of latest increment` (see Figure 12-12 on page 437).

Note: Administration tasks can be added to a job and triggered externally. However, it is a good idea to use the DCAdmin command-line tool available with IBM Cognos Business Intelligence server for full automation of new data processing within near real-time updates.

12.3.3 Using the DCAdmin command-line tool to apply incremental updates

Using the DCAdmin tool (described in 7.7, “Administering cubes from command line” on page 218), you can apply incremental updates to dynamic cubes.

Use the command line parameter `incrementallyLoadCubes` to pass a new TID value to the cube as shown in Example 12-3.

Example 12-3 Using the DCAdmin tool to pass a value of 1 for the new TID (Windows server)

```
dcadmin.bat  -s ServerName -l "LDAP,adminUser,adminPwd" -arg transactionID 1
incrementallyLoadCubes gosldw_sales
```

The DCAdmin tool is useful to programmatically control when values are made available and the update frequency using extract, transform, and load (ETL) or orchestration tools.

Example: Using the DCAdmin tool for process automation

Sometimes new data must be validated before making it available for general consumption such as a financial signoff or when final processing is complete for closed transactions. In this case, you can use the DCAdmin tool to set a specific TID, add the data to the system with the new TID, and make it available when approval is complete.

Note: Before data is approved and made available in the cube, a subset of users can run dynamic query-based reports (based on a Framework Manager model) against the new data. The caching is disabled and the report filtered by a parameter for the TIDs not yet available to the dynamic cube. This approach enables the users responsible for approvals to see the data before it is made available to others.

An approach is to use an ETL tool to read the current TID in a cube by running the `getCubeMetrics` command in the DCAdmin tool to check the `valueOfLastNearRealTimeTID` metric and insert new rows beyond that point. For more information about this method, see

Loading incremental updates to dynamic cubes using IBM Cognos Business Intelligence 10.2.1.1 Fix Pack 3 at:

http://www-01.ibm.com/support/docview.wss?uid=swg27040417

Then, the ETL process issues the `incrementallyLoadCubes` command for the current TID and starts inserting new data using the next increment of the TID. This operation would be performed at a particular point such as on a schedule, after a defined volume of new data, or another logical break.

Another option for tracking TID values is to use a table in the database to track which TID is current or new. This table can also be read and used as a parameter to filter a new data report. This table is not part of the dynamic cubes system, and is created and maintained outside Cognos BI by the new data load process.

12.3.4 New dimension element processing

New dimension members such as a new product or channel are not currently supported in near real-time updates though there are strategies for dealing with them.

When the Refresh Member Cache (`refreshCubeMemberCache`) command is run using either the DCAdmin tool, an administration task as shown in Figure 12-20, or the Administration Console as shown in Figure 12-21, the new dimensional framework is loaded into memory. The old framework is swapped for the new framework and then discarded.

Figure 12-20 Administration task for refreshing member cache

Figure 12-21 shows using the Refresh Member Cache action in the Administration Console.

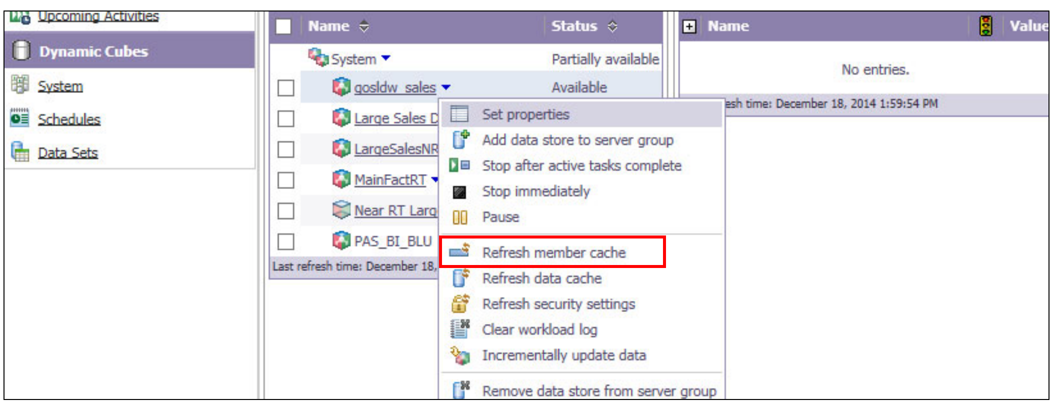


Figure 12-21 Refresh member cache action

One of the downsides of this process is that it invalidates the other caches, which means reloading in-memory aggregates, having no data cache primed, and so on. On large data volumes, it can be time consuming, require lots of resources, and is similar to a full restart of the cube.

Note: To avoid the performance issues caused by loading new members, you can have future dates and times available in your time dimensions or use holding members (orphan members) so that new data can still be assigned without requiring you to refresh the member cache. Real new member names can be assigned later when the cube is fully restarted at a convenient time.

Another option is to use a virtual cube to merge a historic cube with a cube for new data that has near real-time updates enabled. The virtual dimensions are then used to merge new members. Merging members still requires a refresh of the member cache and a reload of data caches but the impact is only on the new cube with a smaller data set. For this reason, it is quicker and less computationally expensive to complete.

12.3.5 Committing new data to the historic storage

It is likely that the volume of new data missing in the in-database aggregates will eventually cause performance issues. The severity of the performance issue depends on the underlying database and tuning. However, it still requires attention at some point, especially when cubes are restarted.

The process for committing new data to the historic data set is a simple one and consists of the following steps:

1. Increment to the latest TID, record the current TID value for the dynamic cube, or both.
2. Pause the dynamic cube using the DCAdmin `pauseCubes` command or the administration task as shown in Figure 12-17 on page 440 or the Administration Console as shown in Figure 12-21 on page 442.
3. Update or add all the new data that is not NULL, up to and including the current TID, to the in-database aggregate tables using your preferred method, for example an ETL tool, SQL, and so on.
4. Update the TID column to NULL for all values up to and including the current TID. If you are using a view as discussed in 12.2.3, “Adding near real-time updates to an existing cube” on page 432, move these rows to the main fact table.
5. Start or unpause the dynamic cube using the DCAdmin command `startCubes` or the administration task as shown in Figure 12-17 on page 440 or the Administration Console as shown in Figure 12-21 on page 442.

Notes:

- Although the pause step can be replaced by stopping the cube, note that pausing is less destructive on performance. When paused, a dynamic cube continues running, so the data caches remain valid. Data that is held does not require requerying and data that is not currently in a cache is available in the database aggregates leading to a reduced impact on user performance.
- When pausing a cube and committing the new data to the historic storage, each insert transaction for subsequent data must use a TID value greater than any previous transaction. For the fact rows that have been set to NULL value, the next TID with new data should resume at a value greater than the last TID used before set to NULL.

For more information about maintaining aggregate tables, see *Maintaining aggregate tables using IBM Cognos Business Intelligence 10.2.1.1 Fix Pack 3* at:

http://www-01.ibm.com/support/docview.wss?uid=swg27040458

For more information about near real-time updates, see *Updating data in near real time using IBM Cognos Dynamic Cubes 10.2.1.1 Fix Pack 3* at:

http://www-01.ibm.com/support/docview.wss?uid=swg27040299

13

Dynamic cube lifecycle

This chapter describes the five main stages in the dynamic cube lifecycle and how they fit within a three-tiered approach of moving from development to test to production. This chapter also offers best practice approaches for moving a dynamic cube onto separate systems.

This chapter contains the following sections:

- Overview of the dynamic cube lifecycle
- Mechanisms for moving dynamic cubes
- Effective practices for deploying between environments

13.1 Overview of the dynamic cube lifecycle

The main objective of customers that use Cognos Dynamics Cubes is to deliver their benefits throughout the organization by making high performing cubes and reports available to their business users. A number of tools and activities are required for this availability to occur.

This section details the dynamic cube lifecycle and management of the development process from initial concept to optimization and refinement. Figure 13-1 illustrates, at a high level, the basic Cognos Dynamic Cubes workflow and the primary tools that are used to perform the main activities in the cycle.

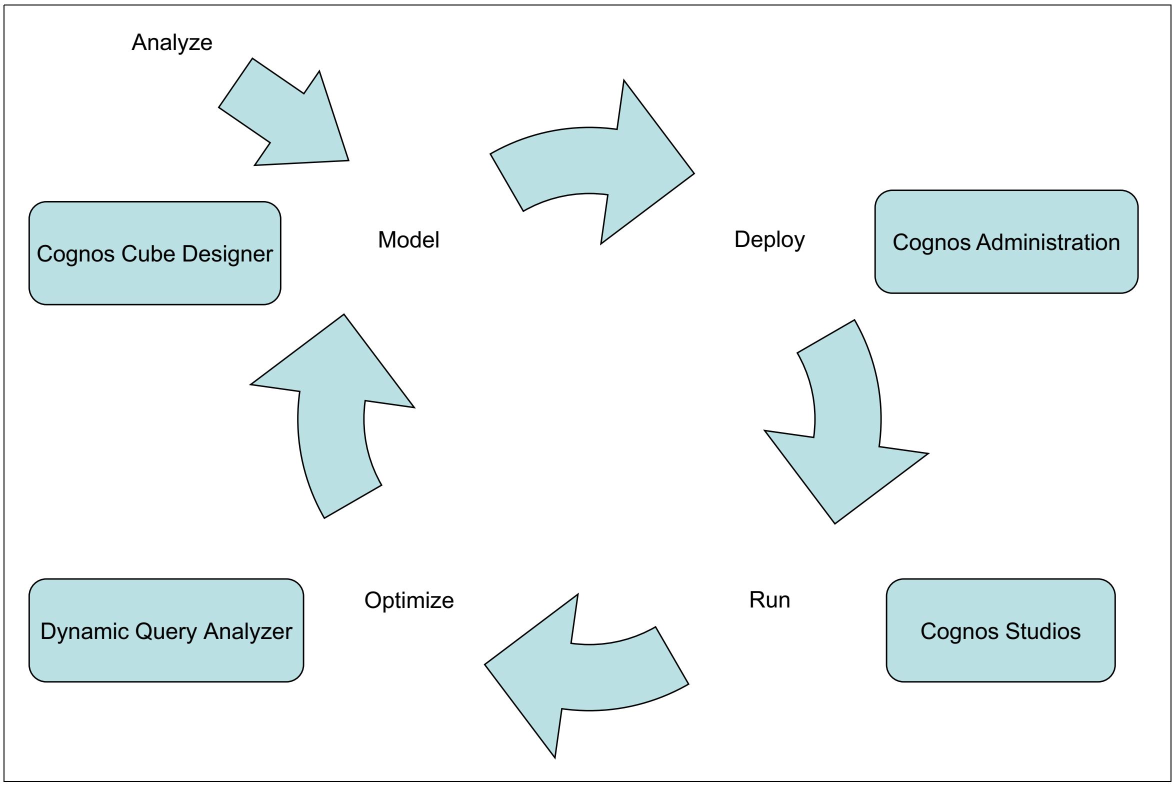


Figure 13-1 Relationships between Cognos Dynamic Cubes activities and tools

This section describes each of the workflow activities in the dynamic cube lifecycle.

13.1.1 Analyze

Although this initial stage of the lifecycle is not part of the ongoing cycle, it is still an important part of the process. The analyze stage is where the architects look at their data, hardware, business requirements, and so on, and prepare for implementing Cognos Dynamic Cubes.

In this stage, architects perform the following tasks:

- Determine business requirements
- Design the database physical model
- Acquire the appropriate hardware
- Install and configure IBM Cognos Cube Designer

See Chapter 3, “Installation and configuration of IBM Cognos Cube Designer and IBM Cognos Dynamic Query Analyzer” on page 65 for details about installing and configuring Cognos Cube Designer.

The objective of the model stage is to design and create a functional dynamic cube by using IBM Cognos Cube Designer. A system analyst determines the high-level business requirements, and a modeler creates a dynamic cube to satisfy those requirements.

In this stage, a modeler can perform the following tasks:

- Model dimensions and hierarchies
- Model measures
- Bring measures and dimensions together into a cube
- Model in-database aggregates and user-defined in-memory aggregates
- Define and model security filters and views
- Create named sets
- Generate estimates for hardware sizing
- Publish cubes and packages

See Chapter 4, “Modeling with IBM Cognos Cube Designer” on page 89, Chapter 5, “Basic modeling” on page 105, and Chapter 6, “Advanced topics in modeling” on page 159 for information about modeling cubes with Cognos Cube Designer and a range of modeling techniques. Considerations for modeling virtual cubes are outlined in Chapter 8, “Virtual cubes” on page 223.

13.1.2 Deploy

After a dynamic cube is published, an administrator configures the cube and makes it available for use. The tool used for tasks in the deployment stage is IBM Cognos Administration.

In this stage the administrator might perform the following tasks:

- Adjust the memory allocation for the query service
- Adjust the configuration of a cube instance (for example, memory allocation for in-memory aggregates)
- Assign users, groups, and roles to security views
- Start and stop cubes
- Monitor cube metrics
- Refresh the caches
- Enable workload logging
- Schedule administrative tasks

The administrator might also choose to use the DCAdmin command-line tool to perform some of these tasks. The use of this tool is detailed in 7.7, “Administering cubes from command line” on page 218.

Information on dimensional security, cube security, and securing the Cognos environment can be found in Chapter 9, “Dimensional security” on page 255, and Chapter 10, “Securing the IBM Cognos BI environment” on page 321.

13.1.3 Run

The run stage of the dynamic cube lifecycle begins when the cube has been configured and is started. Report authors can now create reports by using the various reporting applications, such as Workspace Advanced, Report Studio, or Analysis Studio.

In this stage, business users can perform the following tasks:

- Validate the design of the cube
- Author reports
- Run reports
- Evaluate the performance to determine whether any optimization is required

13.1.4 Optimize

The optimization stage is required if the performance of the reports does not meet expectations. This is an optional stage, but it includes some tasks that should be performed periodically to ensure optimal performance. For example, after the task of monitoring cube metrics in response to user-feedback regarding report performance, or in the event of significant changes to workload characteristics. Aggregate awareness is a key feature of Cognos Dynamic Cubes and without this stage aggregates cannot be used to their fullest. The optimize stage uses a combination of tools, including Cognos Administration, the Aggregate Advisor feature in IBM Cognos Dynamic Query Analyzer (DQA), and in some cases Cognos Cube Designer.

In this stage, administrators can perform the following tasks:

- Adjust the various performance parameters in Cognos Administration
- Use the Aggregate Advisor in IBM Cognos Dynamic Query Analyzer to generate various recommendations for creating aggregates
- Create aggregate tables in the database (which the modeler adds to the cube)
- Add in-memory aggregates to the cube

The use of Aggregate Advisor to generate aggregate recommendations for optimizing query performance and other techniques and scenarios for performance tuning, see Chapter 11, "Optimization and performance tuning" on page 341.

13.1.5 Lifecycle across different environments

The various stages, from start to finish, in the lifecycle of a Cognos dynamic cube occur over multiple environments. The actual number of environments might differ slightly from one organization to the next, but a three-tiered approach is the most common setup as follows:

- Development

  This is the primary environment used by cube modelers and report developers to design the cube and to create reports. Development is the most flexible environment for allowing them to make changes. The database often contains only a subset of the production data. The objective in this environment is to deliver functional objects to the test environment.

- Test

  This is a more rigid environment that is geared for receiving deliverables from development and performing the necessary testing to validate these deliverables. The scale of the test database is important. This database must contain enough data to be able to validate performance of the deliverables.

- Production

  This is a highly controlled environment. Business analysts are actively using this system for real work, so changes in the production environment should be made only after the changes have been verified in the test environment.

Figure 13-2 shows a flowchart of the dynamic cube lifecycle across development, test, and production environments.

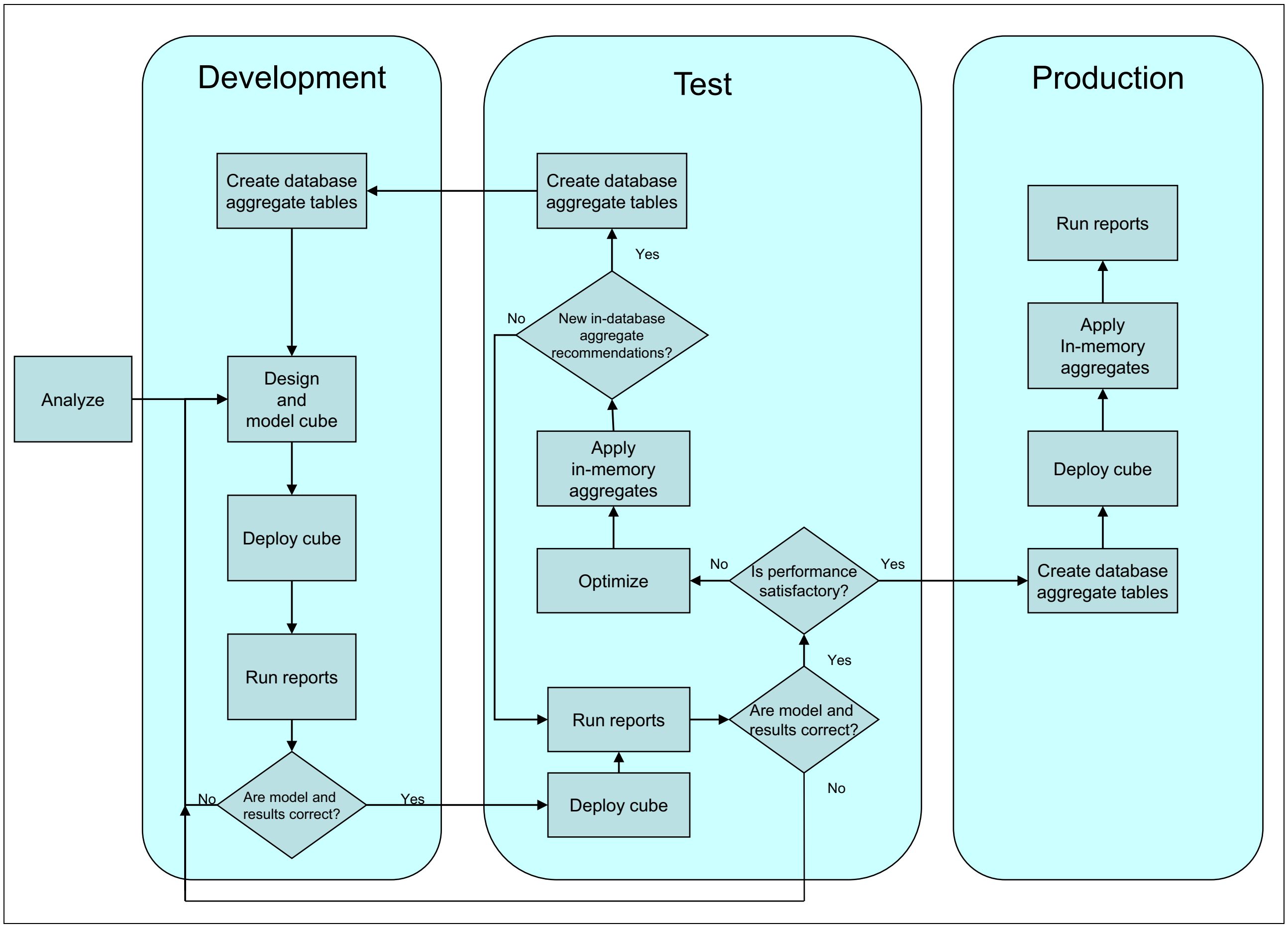


Figure 13-2 Dynamic Cube lifecycle across development, test, and production environments

13.1.6 Development

The development environment is the primary environment used by cube modelers and report developers to design the cubes and to create reports. The developers create defined business requirements that the modelers then create a cube model to satisfy. Dynamic cubes should be modeled in an iterative fashion. This involves defining the cubes with a few hierarchies and measures, and testing them to make sure that they are correct before gradually expanding the cubes to include more dimensions and more complex objects. If necessary, security filters and views are added into the model. Adding security filters and views should be done towards the end of the modeling process.

When the cube is ready for deployment, the modelers might want to take advantage of the feature in Cognos Cube Designer that allows the modeler to publish the cube, add the cube to the dispatcher, set the access account, and even start the cube all in one step. The development environment is one in which users generally have more authority, such as access to administrative tasks. Using the `Publish with Additional Options` feature to handle some of the administrative tasks can speed up the iterative workflow.

After the cube is made available in Cognos Connection and started, report authors create sanity reports to validate that the basic functionality of the cube is correct. If any of the dimensions, hierarchies, measures, or other components do not appear to be correct, the modeler returns to the cube model to investigate, correct, and republish the cube. This cycle repeats until the cube is deemed satisfactory by the report developers and the reports against it return the expected data. When this point is reached, the dynamic cube is ready for testing. The cube objects in the content store are transferred to the test environment by using one of the methods described in 13.2, "Mechanisms for moving dynamic cubes" on page 452.

If issues exist in the subsequent environments, the work effort shifts back into the development environment for further investigation or updates. Additionally, if there are any in-database aggregate recommendations implemented from the Optimize phase of the subsequent environments, those tables must also be created in the development environment and modeled as in-database aggregates in the cube.

13.1.7 Test

When the dynamic cube is transferred into the test environment, a more comprehensive set of reports are authored and run. First, the testers review the dimensions, hierarchies, measures, or other components in the cube. If any of these items do not appear to be correct, the modeler returns to the cube model again to investigate. After the testers are satisfied that the cube in the test environment is modeled correctly, they can start analyzing the performance of the reports.

If the performance of the reports is not acceptable, then the tester or Cognos administrator can adjust some of the tuning parameters on the cube configuration or the query service hosting the cube instance. Another option is to analyze the query workload against the cube structure to obtain recommendations for in-memory and in-database aggregates to be added to the cube. The Aggregate Advisor reviews the workload log of the cube and its structure to determine which measures and levels are queried the most and experiment with various combinations of measures and slices to provide the required coverage. In-memory aggregates can be applied immediately, whereas in-database aggregates require some additional work.

Analyzing query workload against cube structure involves the following steps:

1. Enable workload logging.
2. Run reports to generate and log the query workload.
3. Run the Aggregate Advisor to obtain aggregate recommendations for the cube.
4. Review cube metrics and diagnostic logging to confirm that the recommended aggregates are used.

For any in-database recommendations, a database administrator (DBA) needs to create the tables in both the development and test environments. The new aggregate tables are required in the development environment because this environment is where the modeling efforts take place. The modeler uses these tables and Cognos Cube Designer to model the in-database aggregates. After the in-database aggregates are added to the model in the development environment, the cube model is cycled back to the test environment and more performance testing occurs.

When the reports achieve a satisfactory level of performance, the cube, package, and reports can be deployed from the test environment into the production environment.

13.1.8 Production

The production environment is likely to be a highly controlled environment. Any changes that occur in this environment should be rigorously tested before they are implemented at this level.

If the dynamic cube is transferred by using a deployment archive from the test environment, any in-memory aggregates that were applied to the cube before creating the deployment archive will also transfer across to the production environment. If the dynamic cube is transferred by using Cognos Cube Designer, then an administrator must use DQA to apply the in-memory aggregates to the cube in the production environment.

In IBM Cognos Business Intelligenceversions before 10.2.1 Fix Pack 3, the Aggregate Advisor recommendations are stored on the DQA client system. When using these versions, the administrator can update the dispatcher in the Preferences dialog of DQA to point to the production server before selecting the option to apply the in-memory aggregates to the production environment.

In IBM Cognos Business Intelligenceversion 10.2.1 Fix Pack 3 and later, the Aggregate Advisor recommendations are stored as a set of files on the query service server against which the Aggregate Advisor was run. Relative to the server installation directory, the recommendation files can be found in the following location:

`logs\XQE\ROLAPCubes\{cube name}\advisor\recommendations\`

For example, using one of the sample cubes, replace {cube name} with `gosldw_sales`:
`logs\XQE\ROLAPCubes\gosldw_sales\advisor\recommendations\`

To transfer these recommendations from test to production, complete these steps:

1. Place a copy of the recommendation files on the target production server in the same directory structure as in the test environment.
2. If you have an instance of DQA installed for each environment, open DQA for the target production environment and proceed to step 3.

   If you have a single installation of DQA that is used across environments, update the Dispatcher URI and Gateway URI environment properties in the Cognos Configuration for DQA to point DQA to the production server.
3. Open DQA and select the option to apply the in-memory aggregates to the production environment.

If the cube includes in-database aggregates, then a final set of aggregate tables must be created, this time in the production environment. These tables are required to support the in-database aggregates defined in the production version of the cube. It is not necessary to create these additional tables in the production environment if the cube does not include in-database aggregates.

If security is a requirement for the system and differs between environments, the administrator must also map users and groups to the security views of the production cube.

Even if the test and production environments have mirror databases and identical resources, make sure that all migrated reports still run within the expected time frames before releasing them to the business users.

13.2 Mechanisms for moving dynamic cubes

Dynamic cubes can be moved from one environment into another by using one of two methods:

- Importing the cube by using a deployment archive from the source environment
- Republishing the cube by using Cognos Cube Designer

The first method is the simplest option: Create a deployment archive in one environment and import it into the new environment. This approach fits in with how Cognos traditionally handles the movement of content between environments.

13.2.1 Importing the cube using a deployment archive

Dynamic cubes can be imported to a new environment by using a deployment archive that was created in the source environment. When you create a deployment archive, selecting **Include data sources and connections** causes all data sources (including dynamic cube data sources) to be included.

Figure 13-3 shows the Directory content page of the New Export wizard in IBM Cognos Administration.

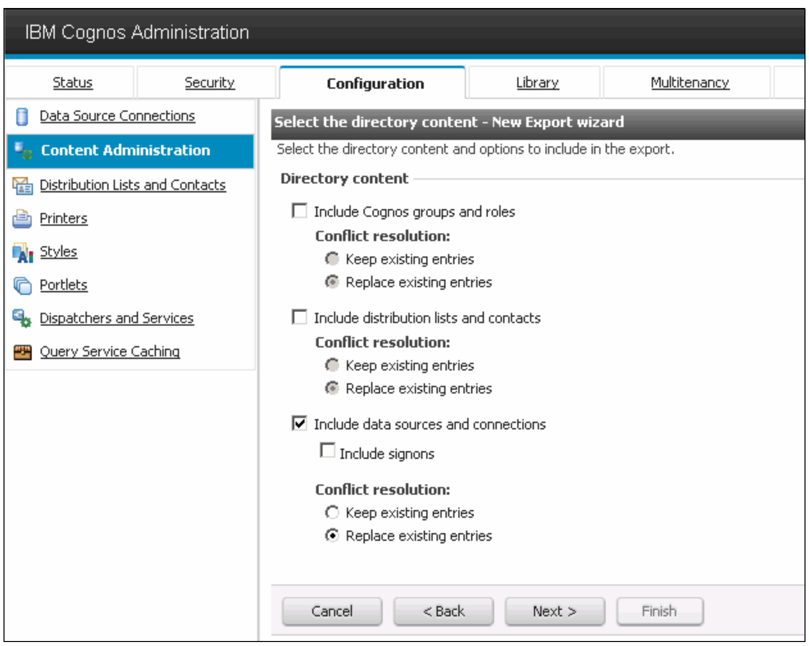


Figure 13-3 Directory content page of the New Export wizard in Cognos Administration

The resulting deployment archive includes all data source connections and the nominated directory content. However, the deployment archive does not include the data access account for the dynamic cube data source connection or any dynamic cube configurations. Using this deployment archive results in the dynamic cube data source connection and its corresponding relational data source connection being transferred to the new environment. The administrator must then use Cognos Administration in the target environment to allocate a user account to the dynamic cube data source connection and configure the cube properties and query service properties.

Note that a deployment archive produced using this method includes all data source connections in the Cognos BI environment, not just those of the content selected for inclusion in the deployment archive. Hence, other dynamic cube data sources and relational data sources that are not relevant in the target environment need to be taken into account. It is possible to obtain a deployment archive that contains only the objects that are required to be transferred to the target environment.

The administrator can consider one of the following approaches:

- Use the development environment:
  a. Create a deployment of the entire development environment to be used as a backup.
  b. Edit the development environment down to only what needs to be exported to the target environment.
  c. Create a second deployment of the development environment.
  d. Import the first, original deployment back into the development to re-create the original development environment from backup.
  e. Import the second deployment into the target environment.
- Use a sandbox environment:
  a. Create a deployment of the entire development environment.
  b. Import the deployment into a sandbox environment.
  c. Edit the sandbox environment down to only what needs to be exported to the target environment.
  d. Create a deployment in the sandbox environment.
  e. Import the sandbox deployment into the target environment.

It is also important to be aware that the imported relational data source connection will retain the connection information for the database of the initial environment, which is the environment in which the deployment archive was created. If the source and target environments each have a separate version of this database, the administrator must update the connection information of the imported relational data source connection to point to the correct database for the new environment.

A final consideration is that the deployment archive does not include the data access account for the dynamic cube data source. It also does not include any cube configurations. The administrator should verify assignment of data access account and reconfigure any cube properties and query service properties in the target environment after the deployment archive is imported.

In most cases, importing a deployment archive that contains a dynamic cube data source is a valid technique for moving a dynamic cube between environments. However, it is not suitable for all scenarios. If any of the following scenarios apply, then you must use Cognos Cube Designer to publish the cube into the new environment instead of importing a deployment archive:

- The table names or table schemas differ between two environments.
- The dynamic cube in the target environment contains user mappings to dimensional security views that you do not want to be overwritten.
- The target environment contains any data sources that you do not want to be overwritten.

13.2.2 Republishing the cube by using Cognos Cube Designer

Republishing a cube by using Cognos Cube Designer provides more flexibility over importing a deployment archive that contains a dynamic cube data source. This approach does not have the limitations that users can encounter with the deployment archive when moving a dynamic cube between environments.

Depending on where Cognos Cube Designer is installed, you can choose to reuse the project file across the various Cube Designer instances. Alternatively, the configuration of a single instance of Cube Designer can be modified to redirect Cube Designer to a server in a different environment.

Reusing a project file across separate Cognos Cube Designer instances

If each environment has its own installation of Cognos Cube Designer, perform the following steps:

1. Transfer the dynamic cube project file from the source environment to the target environment.
2. Start Cognos Cube Designer and open the project file.
3. Locate the required cube in Project Explorer, right-click the cube, and select **Publish**.
4. Complete the publish wizard.

Redirecting Cognos Cube Designer to a different server

If multiple environments share a single installation of Cognos Cube Designer, perform the following steps:

1. Run the Cognos Configuration tool for Cognos Cube Designer.
2. Modify the Gateway URI and the Dispatcher URI for external applications environment properties to reflect the URI of the server in the target environment.

   Figure 13-4 shows the properties to modify in Cognos Configuration when redirecting Cognos Cube Designer to a different environment.

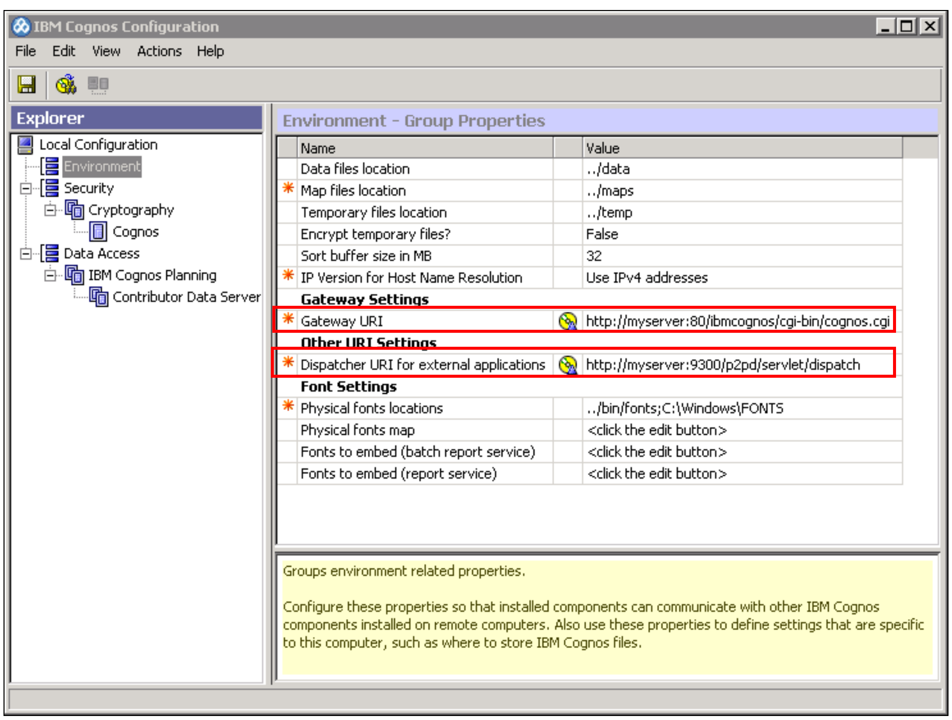


Figure 13-4 Configuration properties to modify when redirecting Cognos Cube Designer to a different environment

3. Save your configuration settings.
4. Start Cognos Cube Designer and open your existing project.
5. Locate the required cube in Project Explorer, right-click the cube, and select **Publish**.
6. Complete the publish wizard.

Note: Installing Cube Designer in its own directory creates a dedicated instance of Cognos Configuration and prevents the action of redirecting Cube Designer to a different server from impacting the configuration of other components in the Cognos BI environment. This technique can also be used to install multiple copies of Cube Designer on the same client. A commonly used approach is for developers to install Cube Designer multiple times on their personal computer and configure each installation to a different environment.

See Chapter 3, “Installation and configuration of IBM Cognos Cube Designer and IBM Cognos Dynamic Query Analyzer” on page 65 for more details.

Using Cognos Cube Designer to change data source, table schema, or table names

When deploying dynamic cubes between environments, accommodate any differences in the databases of each environment. Cognos Cube Designer provides a simple mechanism for modelers to alter the data source name or table schema name that is used in the dynamic cube project. To ensure that the cube definitions are valid after deployment, the modeler must remap all measures and query items that rely on tables whose names differ in the new environment.

Perform the following steps:

1. Either transfer your project file or redirect the Cognos Cube Designer to use the target server. See “Reusing a project file across separate Cognos Cube Designer instances” on page 454 and “Redirecting Cognos Cube Designer to a different server” on page 454 for more detail.
2. Start Cognos Cube Designer and open your existing project. Cube Designer is now connected to the nominated dispatcher in the new environment.
3. From the toolbar, click **Get Metadata Browse Content Manager Datasource**.
4. Select the database schema from which to import data, and then click **OK**.
5. In Project Explorer, expand the `Data Sources` folder and select the data source connection for your relational data source.
6. In the Properties tab, you can make global changes to the relational data source and table schema references. Modify the value of the `Content Manager Data Source` property to change the relational data source references. Modify the value of the `Schema` property to change the table schema references.

Figure 13-5 shows the properties to modify in Cognos Cube Designer when changing the relational data source and table schema references.

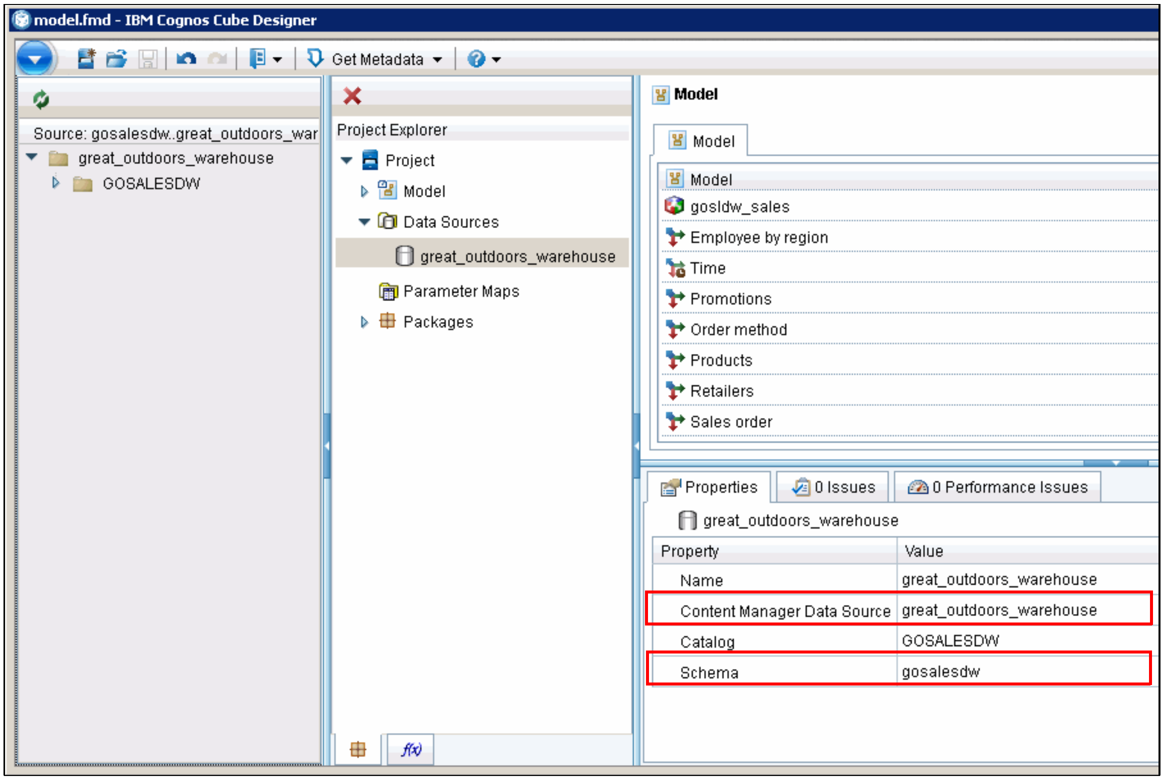


Figure 13-5 Properties in Cube Designer to modify relational data source and table schema

7. If the table names of the imported metadata do not match those of the previous environment, remap the attributes of each object (measure, level, and so on) that uses those tables.
8. Locate the cube in Project Explorer, right-click the cube, and select **Publish**.
9. Complete the publish wizard.

13.3 Effective practices for deploying between environments

This section includes various hints and tips to facilitate movement between environments during the dynamic cube lifecycle.

13.3.1 Keep a copy of the dynamic cube project file under source control

Consider the dynamic cube project file as the original version of the cube model. When a dynamic cube is published as a data source, Cognos Cube Designer transforms the cube model defined in the dynamic cube project into a metadata format that the Dynamic Cube Server can use at run time. However, this transformation is unidirectional. There are no means to reverse-engineer published dynamic cube metadata (a dynamic cube data source) back into a dynamic cube project.

Although the transformed metadata is within the content store and can be transferred from system to system through a deployment archive, the cube metadata itself remains static. To change the design of the published cube model, including the application of in-database recommendations, the modeler must locate the dynamic cube project from which it was published. The project file is opened in Cognos Cube Designer, the model adjusted to reflect the required changes, and the cube republished. This is the only way to modify a published cube model.

If the project file is ever lost or damaged, the modeler needs to re-create the cube model from the beginning to modify the cube. Therefore, it is important to track the project file even after a cube goes into production.

Keeping the project file under source control ensures that the file is backed up and helps to manage changes to the model when multiple modelers are working on the same cube.

13.3.2 Use consistent data source, schema, and table names in all environments

The metadata objects in a dynamic cube model contain references to the names of the underlying relational data source, relational tables, and table columns. Having the fact and dimension tables defined with identical schema and table names in all environments helps to reduce the effort involved in moving a cube between environments.

The use of consistent data source, schema, and table names in all environments provides the following benefits:

- Deployment archives can be used to move a dynamic cube between environments.
- The same scripts can be used in all environments to create the aggregate tables that are required by in-database aggregates.

The modeler can address data source name and table schema name inconsistencies between environments with little extra work. However, to resolve inconsistent table names require a sizable amount of effort and should be avoided.

13.3.3 Skip the optimize step in the development environment

The Aggregate Advisor takes many factors into consideration when devising aggregate recommendations, including the amount of data in the fact table and the cardinalities of the dimensions that influence the advisor results. Therefore, running the advisor against a database that contains a small subset of your data can generate drastically different recommendations compared with running it against the complete database. Where data volumes differ in such a way between environments, the aggregate recommendations must be regenerated in each environment.

For example, recommendations that are generated against cubes with comparatively small data volumes in the development environment, despite working well there, might not provide the expected performance improvement when applied to the same cubes in the test and production environments. Conversely, this example also supports the practice of ensuring that database volumes are the same in the test and production environments.

With the amount of manual effort required to create aggregate tables and model the in-database aggregates, it might not make sense to implement recommendations based on a small database in the development environment. The recommendations generated for the smaller source environment might not apply when you move to the larger target environment, resulting in wasted effort.

Therefore, the optimization stage should occur directly in the test environment, rather than trying to apply the optimizations from the development environment.

13.3.4 Use a separate database for each environment

A separate database should be used for each of the environments. More specifically, the development and test environments should not use the production database as demonstrated in the following scenarios.

Activities performed in one environment can affect the users of other environments that share a database. For example, during report development, authors might unintentionally create reports that result in resource-intensive queries and negatively affect the performance of other consumers of that database. Similarly, if the test and production environments share a database, the queries of reports undergoing performance testing might perform unexpectedly, thus negatively impacting performance in production. The Aggregate Advisor might also generate resource-intensive queries during its run.

The database for development can be a smaller subset of the data. Because there will be numerous iterations and a high level of experimentation in the development system, providing a smaller volume of fact data can accelerate workflow in the development environment.

13.3.5 Include complete dimensional data in development environment

The development environment usually has a substantially smaller subset of data in the fact table. However, if possible, the development environment should have complete dimension data or at least all dimension data that is explicitly referenced in reports, calculations, or security.

When modeling dimensional security, modelers are required to drag in members from the member browser into the security filter expressions. The members that are displayed in the member browser are based on the data in the dimension tables. Without access to all the dimension data, modelers might not be able to define the security filters they require.

Report definitions can also contain references to members. Hence, when investigating problematic reports from the production environment, it is important to be able to run the report in the development environment. If the development environment does not contain all the required dimension data, the report specification might include members that do not exist in the development environment, and the report will fail to run.

13.3.6 Make the test system similar to the production system

The test system must be sufficiently similar to the production system in scale so that performance testing and optimization is meaningful. In ideal circumstances, the database that is used in the test system is a mirror copy of the production database.

This approach allows the full range of optimization and testing to occur in the test environment, and ensures that the aggregates recommendations generated in test remain applicable for the production environment. In cases where the test and production systems are of comparable scales (such as within a factor of two), the recommended aggregates should still be acceptable, despite the text system not being a mirror copy of production. If a greater variance exists between the systems, then the recommended aggregates might not perform as well for production cubes.

Running Aggregate Advisor on the production system is generally undesirable because the advisor might generate some resource-intensive queries while creating the aggregate recommendations.

To obtain aggregate recommendations that reflect the user workload on the production system, complete these steps:

1. Enable workload logging on the production system and capture workload for an appropriate amount of time.
2. Disable workload logging when it is no longer necessary to capture usage.

Use the following steps to transfer production workload logs to the test environment where aggregate recommendations can be generated and verified without impacting the production system:

1. Place a copy of the workload log file in the test system in the same directory structure as on the production system. See 13.1.7, “Test” on page 450 for the location of the workload log file.
2. Run Aggregate Advisor on the test system with a database that is a mirror copy of the production database and select the option to include workload information in the run.
3. Run reports to verify performance of the new aggregate recommendations in the test environment.
4. After verification is complete, follow the steps described in 13.1.8, “Production” on page 451 to apply the recommended aggregates to the production environment.

14

Troubleshooting

This chapter describes the troubleshooting tools available for visualizing, investigating, and resolving problems with Cognos Dynamic Cubes.

This chapter contains the following sections:

- Common problems
- IBM Cognos Dynamic Query Analyzer troubleshooting features
- Enabling query execution tracing
- Enabling query plan tracing
- The query service log file
- Enabling Cognos Dynamic Cubes logging
- Reviewing Cognos reports with DQA
- Open query service log with DQA
- IBM Support Assistant

14.1 Common problems

This section describes several problems that you might encounter and how to resolve them. Additional examples are described in 6.10, "Troubleshooting" on page 191, 11.10, "Scenarios for performance tuning" on page 415, and the IBM Knowledge Center *Troubleshooting* topic at:

http://www-01.ibm.com/support/knowledgecenter/SSEP7J_10.2.2/com.ibm.swg.ba.cognos.ug_cog_rlp.10.2.2.doc/c_cog_rlp_troubleshooting.html

14.1.1 Cube does not start due to logon error

The following messages are symptoms of a cube that does not start:

```
XQE-ROL-0051 Unable to logon to the access account specified for the dynamic cube:
[cube name].

XQE-ROL-0193 An access account is not specified for the dynamic cube <cube_name>.
Assign an access account for the dynamic cube.
```

Figure 14-1 shows the cube that does not start.

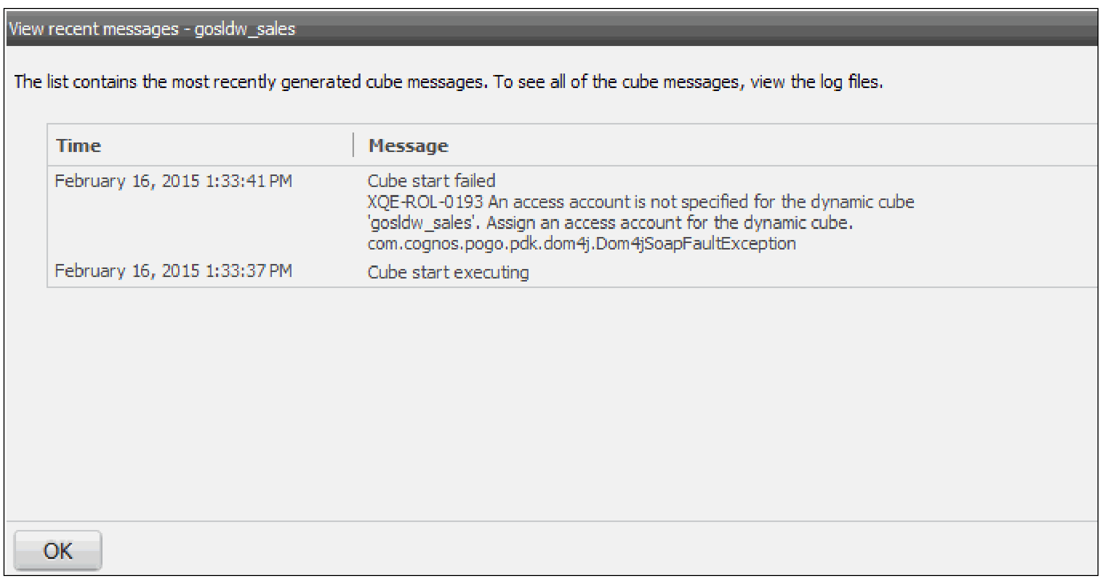


Figure 14-1 Cube does not start

To resolve this issue, make sure that the access account is defined for the cube as described in 10.2.1, "Setting up the access account" on page 332.

14.1.2 Unexpected data values returned

As described in 2.3, "Cognos Dynamic Cubes caching" on page 48, a requested value might be from various caches, so it is often necessary to determine where the incorrect value comes from. The technique to determine from where the value is retrieved involves disabling and then enabling the various caches.

Note: When changing the configuration settings of a cube, the cube must be restarted for the changes to take effect.

Perform the following steps:

1. Go to IBM Cognos Administration.
2. Select **Status**.
3. Select **Dynamic Cubes**.
4. Select **All Server Groups**.
5. Select the server group the server is in.
6. Select the dispatcher.
7. Select the cube.
8. In the **QueryService** menu, click **Set Properties**.
9. Disable all dynamic cube caches by making the following changes:
   a. Select the **Disable result set cache** check box.
   b. Enter the value 0 for **Data cache size limit (MB)**.
   c. Select the **Disable in-database aggregates** check box.
   d. Enter the value 0 for **Maximum space for in-memory aggregates (MB)**.
10. Click **OK** to apply the change.

Note: Wait for a minute for the applied changes to take effect.

11. In the **QueryService** menu, select **Restart**.
12. After the cube is available, test the query or report.

Figure 14-2 shows a cube with all the caches disabled. A cube with this configuration causes a query to access the database fact table to retrieve the data.

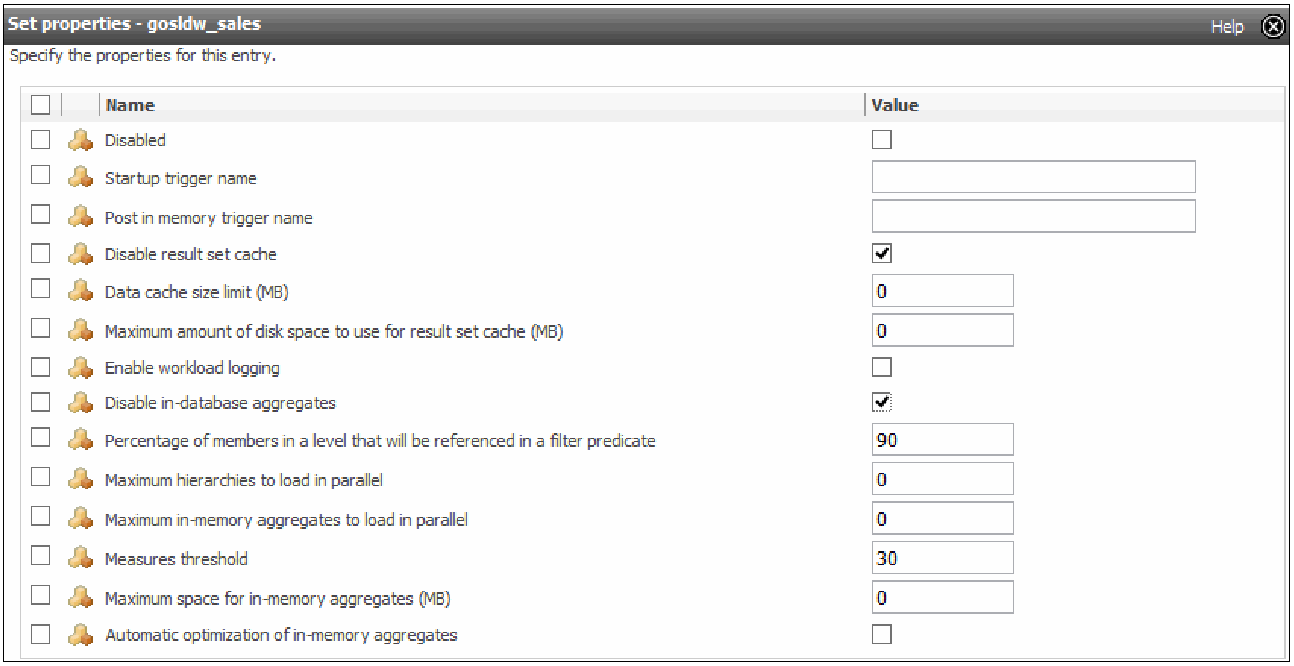


Figure 14-2 Cube with all caches disabled

To enable the caches, complete step 1 on page 463 through 12 on page 463. On step 9 on page 463, enable all dynamic cube caches by making the following changes:

a. Click to clear the **Disable result set cache** check box.

b. Click to clear the **Disable in-database aggregates** check box.

c. Enter a value greater than 0 for **Maximum space for in-memory aggregates (MB)**.

For more information about configuring these caches, see 11.4, “In-memory aggregates” on page 367, 11.5, “Database aggregates” on page 385, and 11.10, “Scenarios for performance tuning” on page 415.

14.1.3 Troubleshooting dynamic cubes aggregate matching

For information about troubleshooting issues related to the aggregate caching strategy, see 11.4.6, “In-memory aggregate tips and troubleshooting” on page 379 and 11.5.5, “Database aggregates tips and troubleshooting” on page 390.

14.1.4 Incorrect numbers from in-memory aggregate

As described in “Incorrect data values” on page 381, problems with referential integrity can affect the quality of the data in summary tables, in-memory aggregates, and data caches.

For more information about how to run a series of SQL queries to identify any referential integrity issues that can cause data discrepancies in aggregates, see 6.6, “Data quality and integrity” on page 185.

14.1.5 Incorrect auto summary value

An incorrect auto summary value might be due to security restrictions. If security is applied to the cube and incorrect auto summary values are returned, you should thoroughly review Chapter 9, “Dimensional security” on page 255 to determine whether your security setup might be causing this problem.

14.1.6 Query returns empty cell for a calculated member

This issue is likely caused by a calculated member (not a measure) referencing a secured member.The reason is that the value of the secured member is treated as null values in the calculation. For more information about calculated members, see 9.7, “Calculated members” on page 272.

14.1.7 Cube loaded but relative time members are missing

To fix this problem, in Cognos Cube Designer, under Time hierarchy, make sure to set `Add Relative Time Members` to `true` as shown in Figure 14-3.

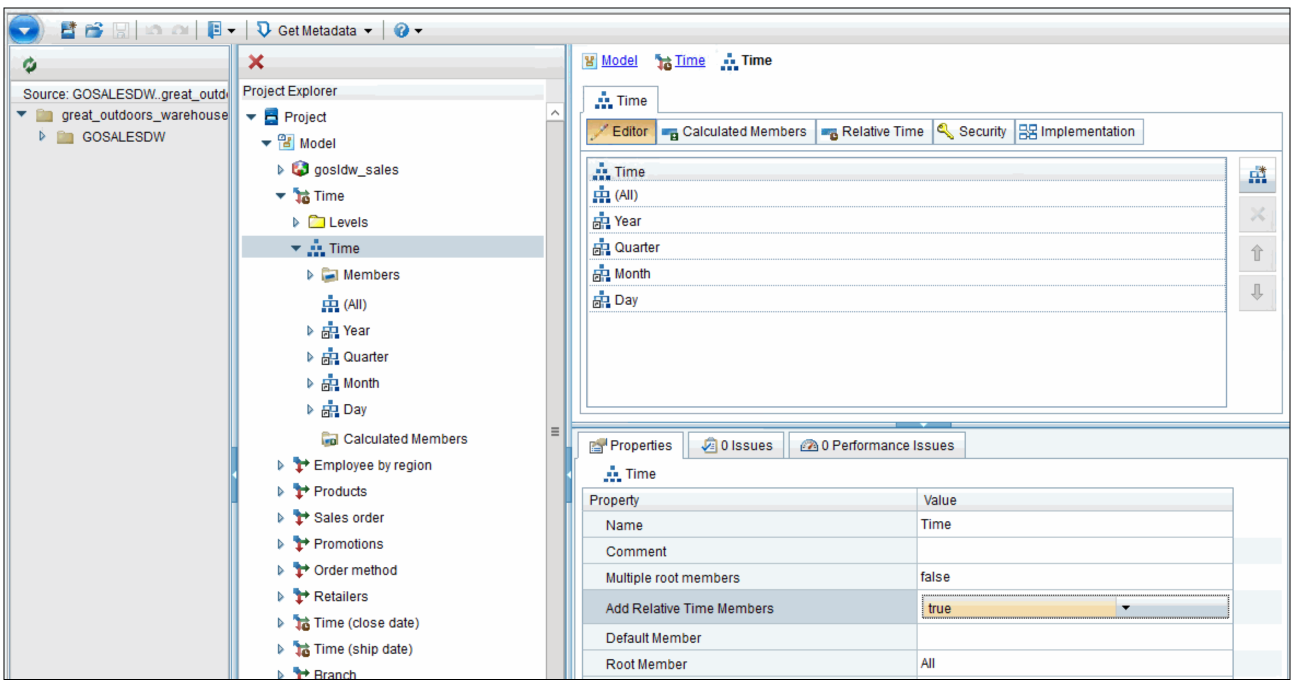


Figure 14-3 Add Relative Time Members property

Redeploy the cube model and restart the cube. For more information, see Chapter 6.2, “Time dimensions and relative time” on page 168.

Another cause for the relative time members not being displayed correctly is a bad expression in a current period expression, such as a typographical error, missing quotation marks around a string, syntax error expression, and so on.

For example, in Figure 14-4, a Current Period expression is selected as valid in the cube modeler.

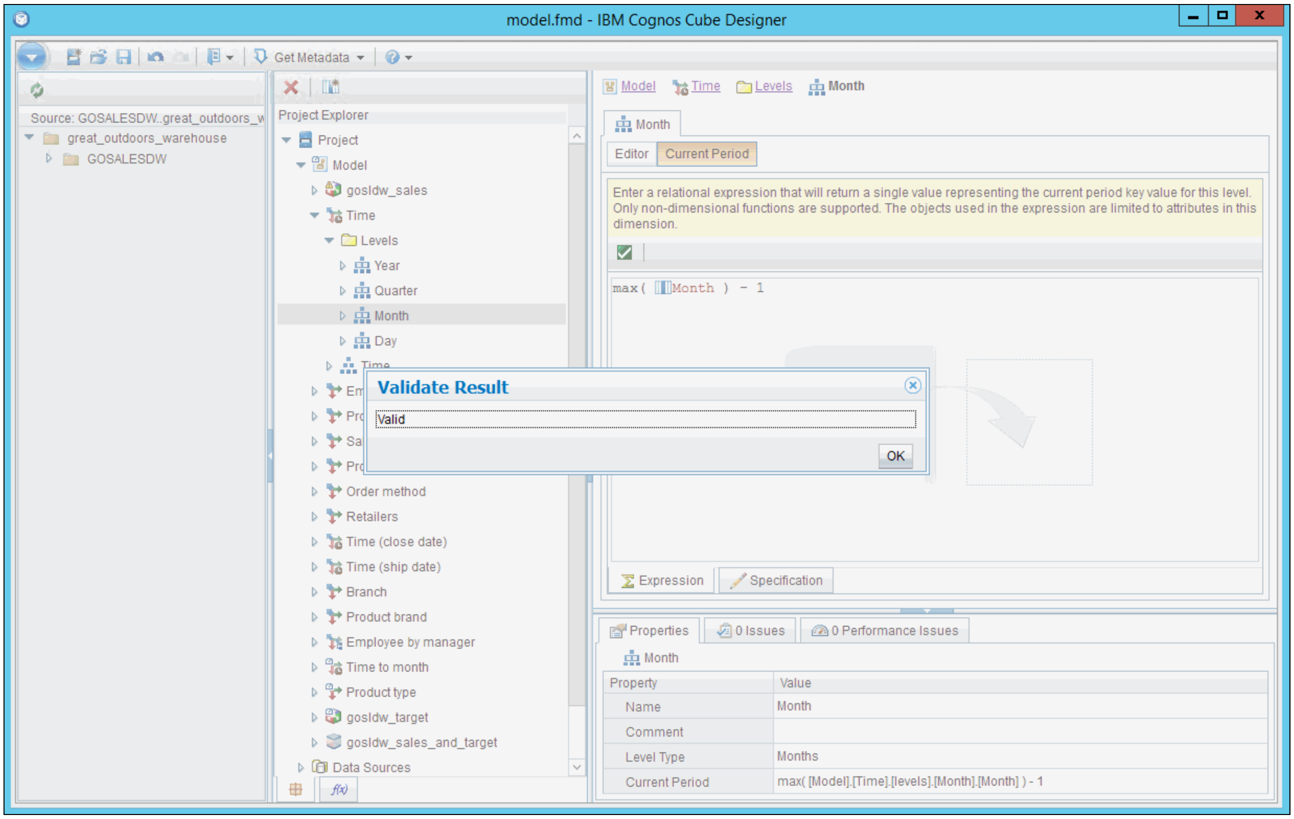


Figure 14-4 Valid Current Month

However, when the metadata tree is expanded in Cognos Workspace, none of the relative time members is listed. See Figure 14-5.

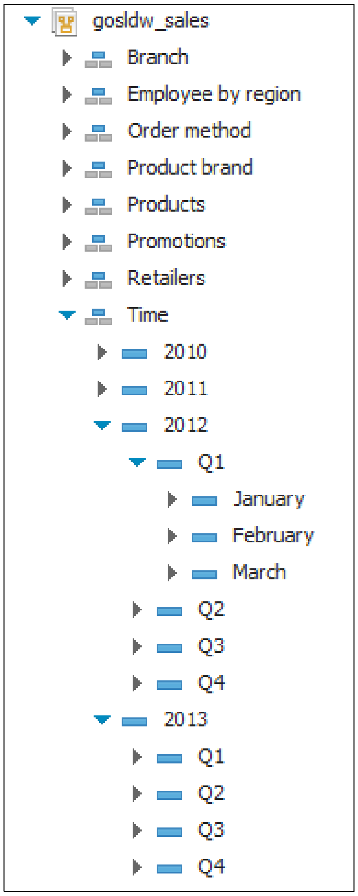


Figure 14-5 No relative time members

To resolve this issue, check the query service diagnostic log to find the error or warning messages about `Time hierarchy`. Enable the event group `"ROLAPCubes.Loader"` with trace level to see more details as shown in Example 14-1.

Example 14-1 Enabling the event group ROLAPCubes.Loader with trace level

```
<event ... group="ROLAPCubes.Loader" ... rolapCube="gosldw_sales"
rolapDimension="Time" rolapHierarchy="Time" ... ><![CDATA[Current period
expression failed to execute. XQE-DAT-0001 Data source adapter error:
com.ibm.db2.jcc.am.SqlDataException: Invalid character found in a character string
argument of the function "DECFLOAT".. SQLCODE=-420, SQLSTATE=22018, DRIVER=4.17.29
- when processing query: SELECT DISTINCT
    max("GO_TIME_DIM2"."MONTH_EN") - 1 AS "Time_Month_key"
FROM
    "GOSALESDW"."GO_TIME_DIM" "GO_TIME_DIM2"
WHERE
    "GO_TIME_DIM2"."CURRENT_QUARTER" <> 0
ORDER BY
    "Time_Month_key" ASC
FOR FETCH ONLY.
   ...
]]></event>
```

Note: Use the technique described in this section to troubleshoot problems when calculated members or calculated measures are missing.

The expression `max(Month) -1` as shown in Figure 14-4 on page 466 is attempting to find the month before the last month in the GO_TIME_DIM dimension table (November 2013). However, the `Month` attribute is referencing the MONTH_EN column, which is a character string column. The `Month key` attribute should be used instead of Month as follows:

```
max(Month key) -1
```

After the incorrect expression is removed, the correct Relative Time Hierarchy can be seen in the metadata tree as shown in Figure 14-6.

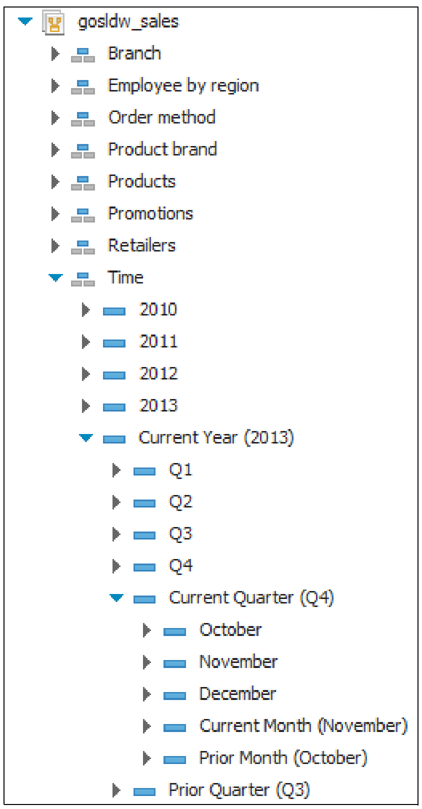


Figure 14-6 Relative time metadata

For information about relative time see 6.2, “Time dimensions and relative time” on page 168. Examples of level current period expressions are available at:

http://www-01.ibm.com/support/knowledgecenter/SSEP7J_10.2.2/com.ibm.swg.ba.cognos.ug_cog_rlp.10.2.2.doc/c_ug_cog_rlp_examplesoflevelcurrentperiodexpressions.html

14.1.8 Dispatcher global unique identifier (GUID) not found

This error occurs when you try to view table data in Cognos Cube Designer by right-clicking the menu on a table and selecting **View Data**. The error `Dispatcher GUID not found` is returned, as shown in Figure 14-7.

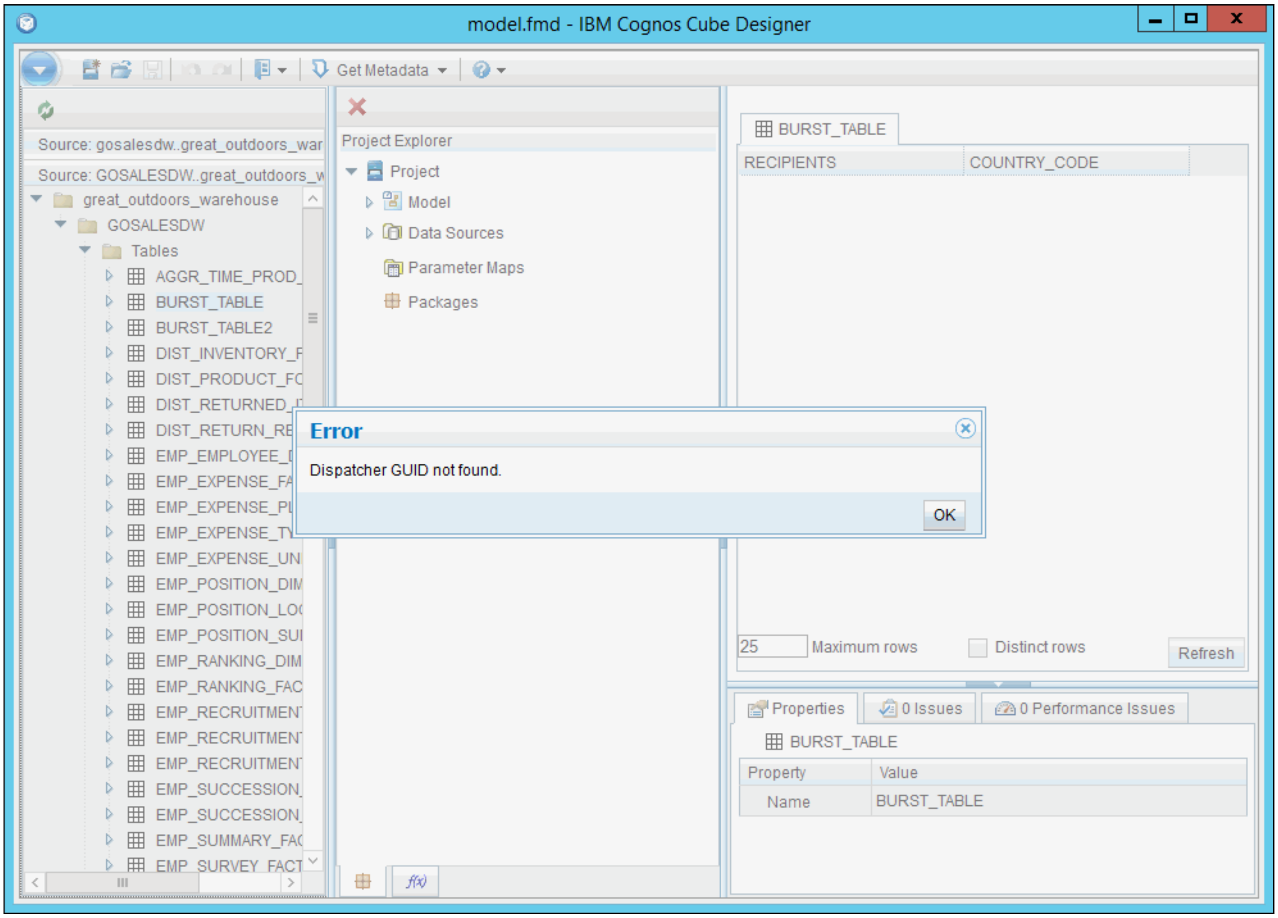


Figure 14-7 Dispatcher GUID not found

To fix this error, make sure that the IBM Cognos BI server is not using localhost in the Dispatcher URIs for gateway setting in the IBM Cognos Configuration as shown in Figure 14-8. Save the Cognos Configuration and restart the IBM Cognos service.

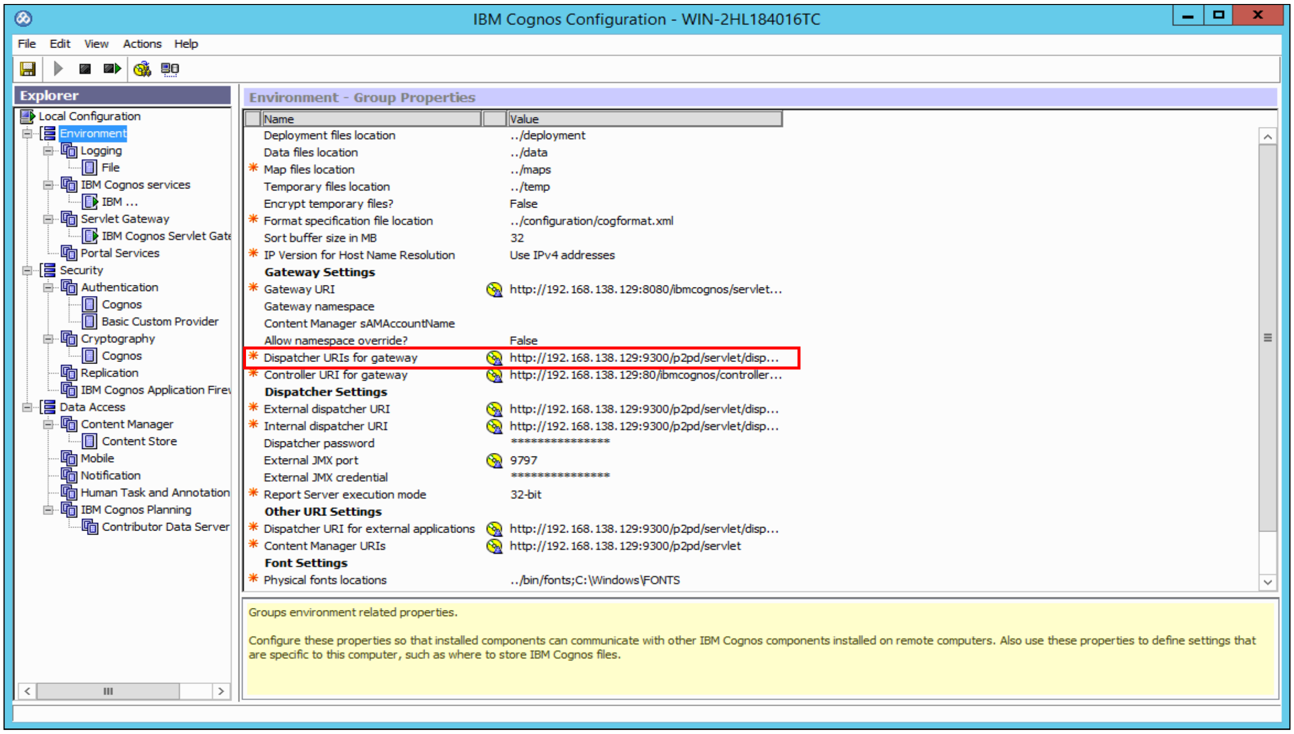


Figure 14-8 IBM Cognos Configuration - Dispatcher URIs for gateway

14.2 IBM Cognos Dynamic Query Analyzer troubleshooting features

The IBM Cognos Dynamic Query Analyzer (DQA) provides Cognos report developers and system administrators with an Eclipse-based client user interface that can visualize and review the composition of queries to a dynamic cube and to the relational source data that the dynamic cube is based on.

This section focuses on the features of DQA that support troubleshooting and investigation into IBM Cognos Dynamic Cubes and their associated reports. DQA is the primary query visualization tool for dynamic query mode (DQM). Many of the concepts discussed in this section can be applied to reports that use DQM data sources, but are not based on dynamic cubes. DQA includes the following features:

- Aggregate Advisor
- Analyzing query log files and query service log files
- Enabling logging on a report basis

Aggregate Advisor

Aggregate Advisor is a performance optimization utility that is embedded in DQA. You can use Aggregate Advisor to obtain recommendations based on dynamic cube model and query workload. Cognos Dynamic Cubes takes advantage of aggregate table and in memory aggregate definitions for aggregated data results, optimizing query performance. Aggregate Advisor is discussed in Chapter 11, “Optimization and performance tuning” on page 341.

Analyzing query log files and query service log files

DQA enables you to view and analyze query log files. The graphical representation of the log files allows you to see performance bottlenecks. The visualizations provided by DQA help understand the queries requested to satisfy a report, their cost, and their ability to take advantage of existing cached information.

With DQA, you can also view the query service logs. These logs contain information about the state of the query service. Dynamic cube logging information is also shown in these logs.

Enabling logging on a report basis

DQA provides you the option to enable logging only for reports that you run from DQA. All the reports that are run from outside DQA are not affected by this logging configuration. 14.3, "Enabling query execution tracing" on page 480 describes how to enable this logging. In IBM Cognos Business Intelligence versions before V10.2.2, the logging level of the server component for dynamic query had to be enabled for all the reports run on the Cognos Server.

For more information about DQA, see the *Dynamic Query Analyzer User Guide* at:

http://www-01.ibm.com/support/knowledgecenter/SSEP7J_10.2.2/com.ibm.swg.ba.cognos.ug_dqa.10.2.2.doc/c_ug_dqa_gettingstarted.html

14.2.1 Installing and configuring DQA

The steps required to install and configure DQA are listed in 3.5, "Installing Cognos Dynamic Query Analyzer" on page 76.

14.2.2 Creating a virtual directory to access log files

If you do not install DQA on the same computer where the Cognos server is installed, you can create a virtual directory, or alias, to allow DQA to read query log files and workflow log files.

The following page lists the steps required to create the virtual directory:

http://www-01.ibm.com/support/knowledgecenter/SSEP7J_10.2.2/com.ibm.swg.ba.cognos.ig_dqa.10.2.2.doc/t_ig_dqa_createvirtualdirectory.html

14.2.3 Running DQA

To start DQA, click **Start** → **Programs** → **IBM Cognos 10**, and then click **IBM Cognos Dynamic Query Analyzer**.

14.2.4 The DQA workspace

The DQA workspace is organized in perspectives and views. This section explains these concepts.

Perspectives

A Perspective defines the initial set and layout of views in the DQA window. Each perspective provides a set of functions aimed at accomplishing a specific type of task or work with specific types of resources. The following perspectives are available in DQA:

- Aggregate Advisor
- Analyze Logs

Aggregate Advisor

This is the default perspective when you run DQA. It provides access to the Aggregate Advisor functions. Figure 14-9 shows the Aggregate Advisor perspective.

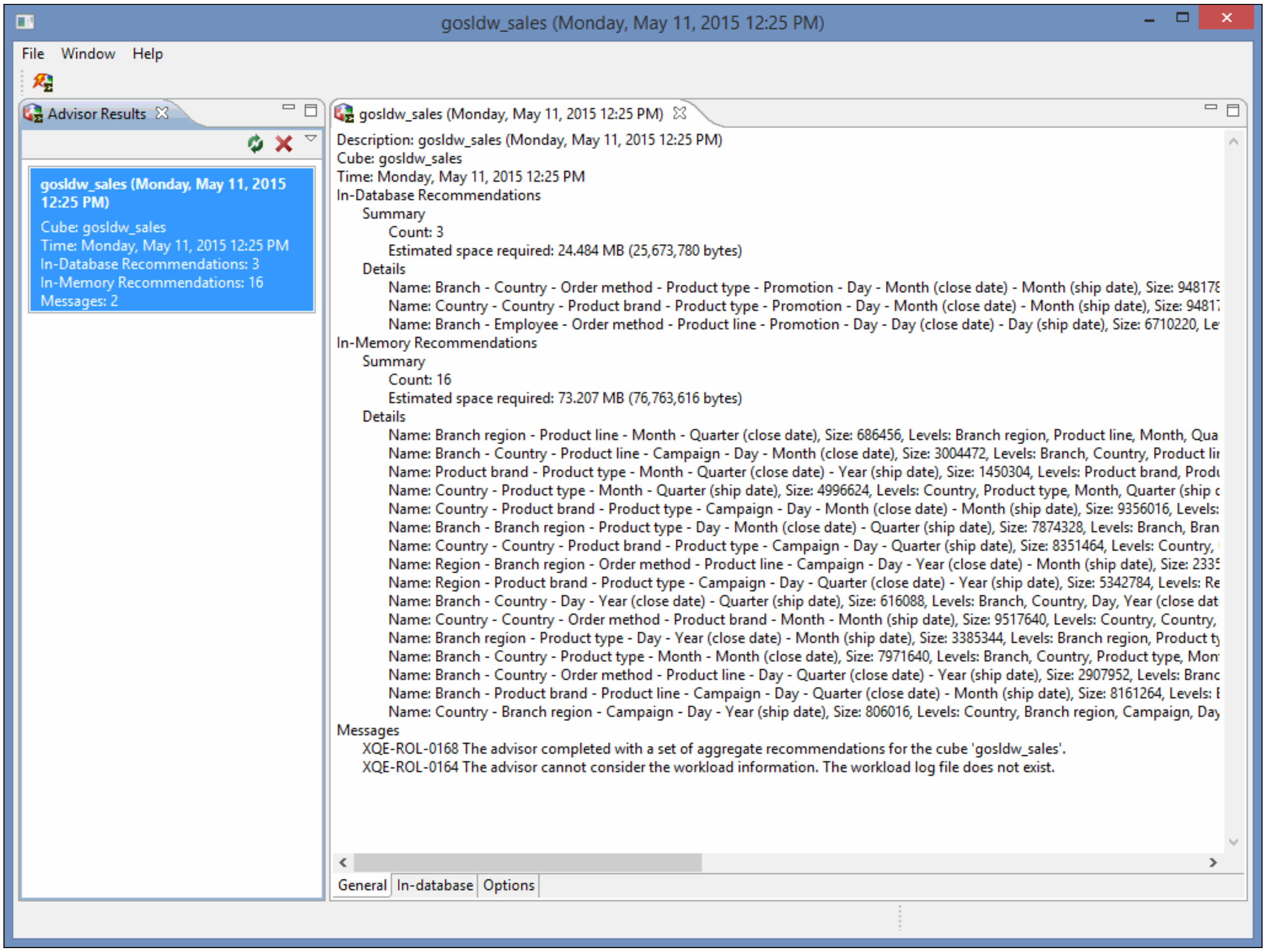


Figure 14-9 Aggregate Advisor perspective

Analyze Logs

This perspective provides access to the logging functions in DQA. Figure 14-10 shows the Analyze Logs perspective.

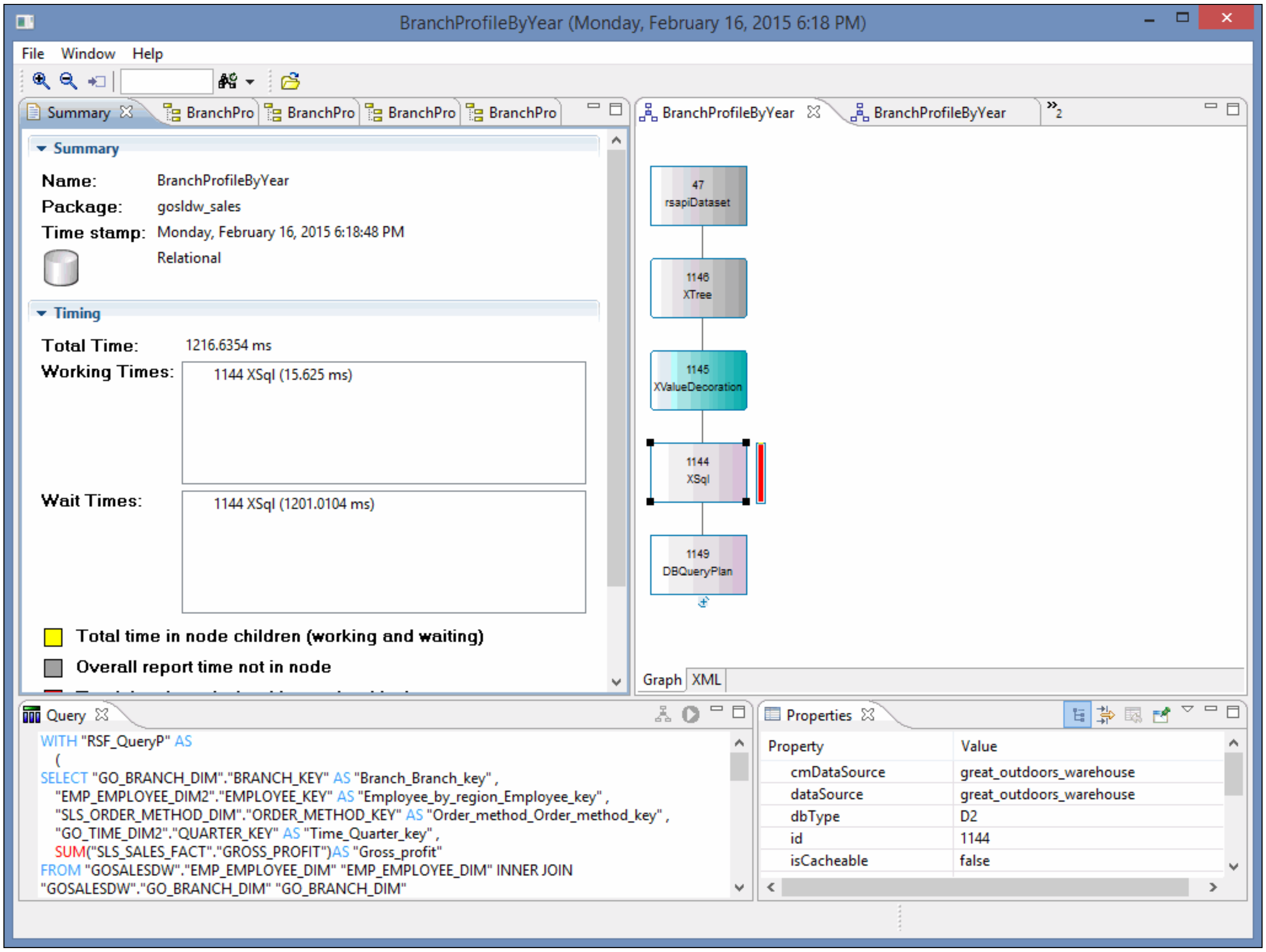


Figure 14-10 Analyze Logs perspective

From the Aggregate Advisor perspective, you can switch to the Analyze Logs perspective by selecting **Window** → **Analyze Logs**. In addition, from the Analyze Logs, you can select **Window** → **Aggregate Advisor** to switch to the Aggregate Advisor perspective.

DQA views

DQA presents information in a series of visual component called views. DQA views help you focus on the information that interests you. All views within DQA can be opened by selecting **Window** → **Show View** as shown in Figure 14-11.

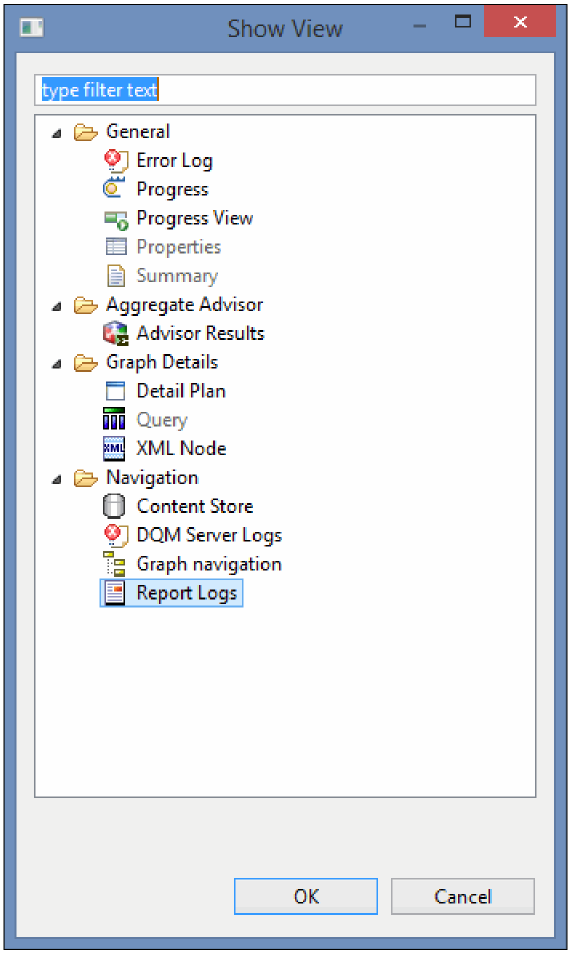


Figure 14-11 All DQA Views

This section describes the views included in the Analyze Logs perspective. See 11.3.5, “Opening Aggregate Advisor results and viewing details” on page 357 for information about views included in the Aggregate Advisor perspective.

The following views are available in the Analyze Logs perspectives:

- Graph view
- Navigation view
- Summary view
- Query view
- Content Store view

Graph view

A graph view is displayed when a query log file is opened that shows a series of linked nodes. The graph represents the query execution plan as a tree of execution nodes. Figure 14-12 shows the graph view.

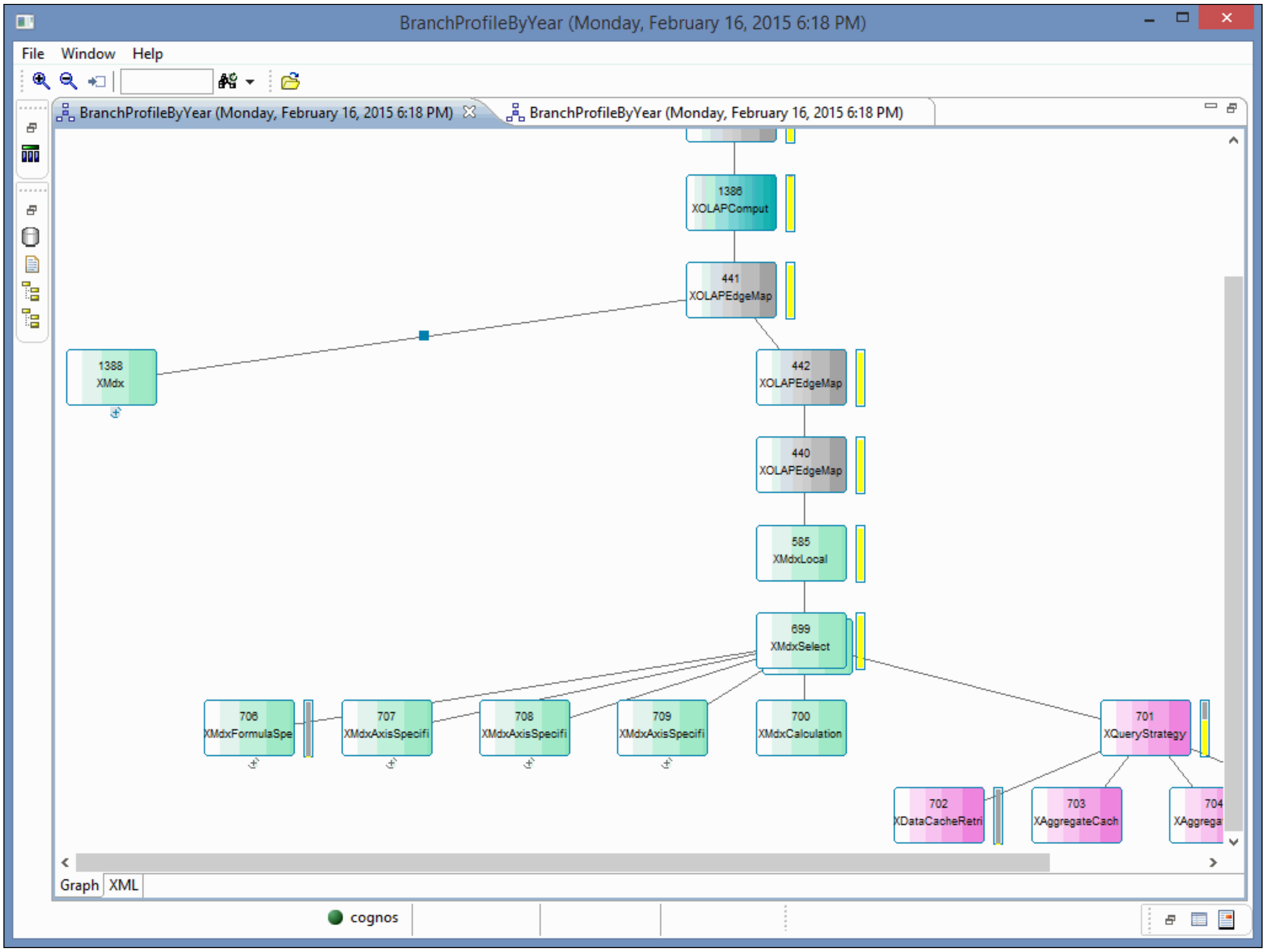


Figure 14-12 Graph view

Each node on the graph represents either an operation that occurred when the report was run or an attribute of an operation (such as the data that was being processed). Figure 14-13 provides a description for the different node representations.

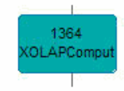
Node	Description
	Represents an operation that occurred when the report was run.
	Represents an attribute of an operation, such as the data being processed. By default, most of these nodes are suppressed in the graph. To display all of them, click **Window**, **Preferences**, and select **Visualization**, **Node filtering**.
	Represents a collapsed node. To display the hidden nodes, double-click the node, or right-click it and click **Show Subtree**.
	Represents a node that contains hidden children due to the **Node filtering** settings. To display the hidden nodes, right-click the node and click **Expand Filtered**.
	Represents the node that is currently selected. The properties of this node are displayed in the **Properties** view. You can select a node by clicking it.
	Represents a node that has subqueries that can be opened in another graph. To display the subquery graphs, right-click the node and click **Open sub queries**. Some subqueries do not have an associated node in the parent trace and can be opened using this option.
	Represents timing information for the node. The color red represents the time for the report spent in the node. The color yellow represents the proportion of time spent in the children of the node. The color gray represents the time spent outside the node and its children. If the node is selected, the times are also displayed in the **Properties** view.

Figure 14-13 Node representations

Figure 14-14 shows more nodes that map to objects within the dynamic cube caching strategy.

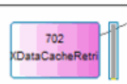
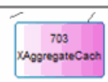
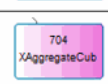
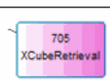
Node	Description
	Use of dynamic cube caching and requests to the underlying relational database are represented as execution tree nodes of XQueryStrategy. The XQueryStrategy node summarizes the total time spent in retrieving data values for use in satisfying the MDX request for the report against the dynamic cube.
	The XDataCacheRetrieval node represents the total time spent satisfying the data request that is reusing data that is held in the data cache.
	The XAggregateCacheRetrieval node represents the total time spent satisfying the data request that is reusing existing data in the aggregate caches.
	The XAggregateCubeRetrieval node represents the total time spent satisfying the data request that is uses external aggregate tables, and caching the data returned.
	The XCubeRetrieval node represents the total time taken satisfying the data request through requests to the underlying (base) relational data and caching the data returned.

Figure 14-14 Dynamic Cube nodes

Navigation view

The Navigation view is a tree representation of a graph. Each graph has its own Navigation view, so the contents do not change when you select another graph. You can compare graphs by opening multiple Navigation views.

You can click **Link with editor** on the toolbar so that the editor reflects what is selected in the graph, and vice versa. The Navigation view allows you to navigate to entries in the tree and to focus on them by double-clicking individual entries.

Figure 14-15 shows the Navigation view of a graph. This figure also shows the **Link with editor** button highlighted in red. The node `1144 XSql` that is selected in the graph was selected in the Navigation view because **Link with editor** is enabled.

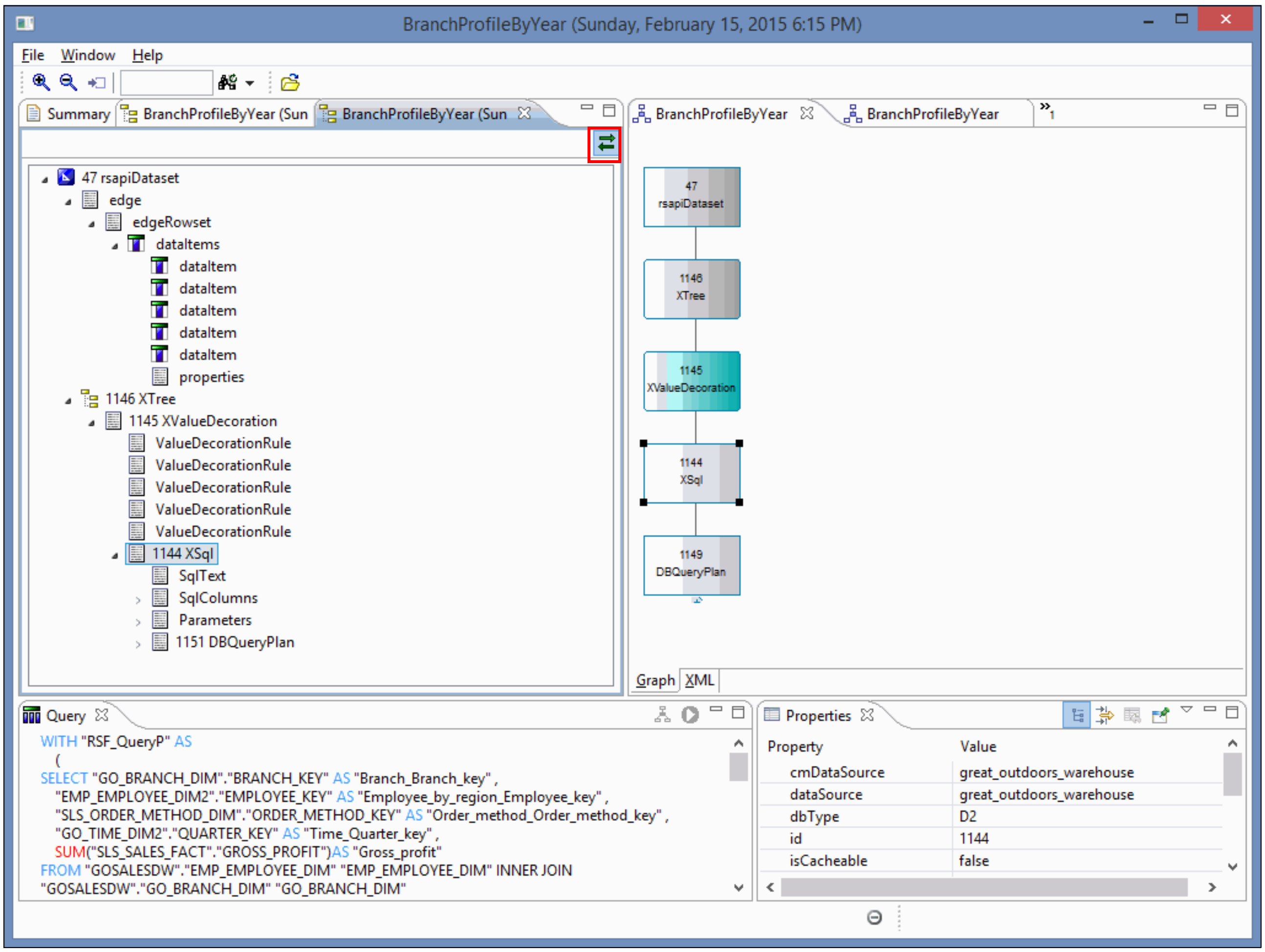


Figure 14-15 Navigation view

Summary view

The Summary view shows the overall information about a graph. The Summary view has four sections:

1. The Summary section shows the name, package, and time the report was run, with a graphic to quickly indicate whether the data source type is relational, OLAP, or dimensionally modeled relational data (DMR).
2. The Timing section is where users doing profiling spend most of their time. The working and waiting time for each node is shown in descending order. Double-clicking any of the entries takes you to the named node.

3. The Analysis section shows additional information about the report that was run, including whether planning was successful, any query hints that were applied, and any filters that were applied. These factors can change the performance of a report significantly.
4. The Node shapes and colors section is a legend that explains the node types and colorings that are used in the Summary view.

Figure 14-16 shows the Summary view as it is opened by default, indicating the summary and timing information.

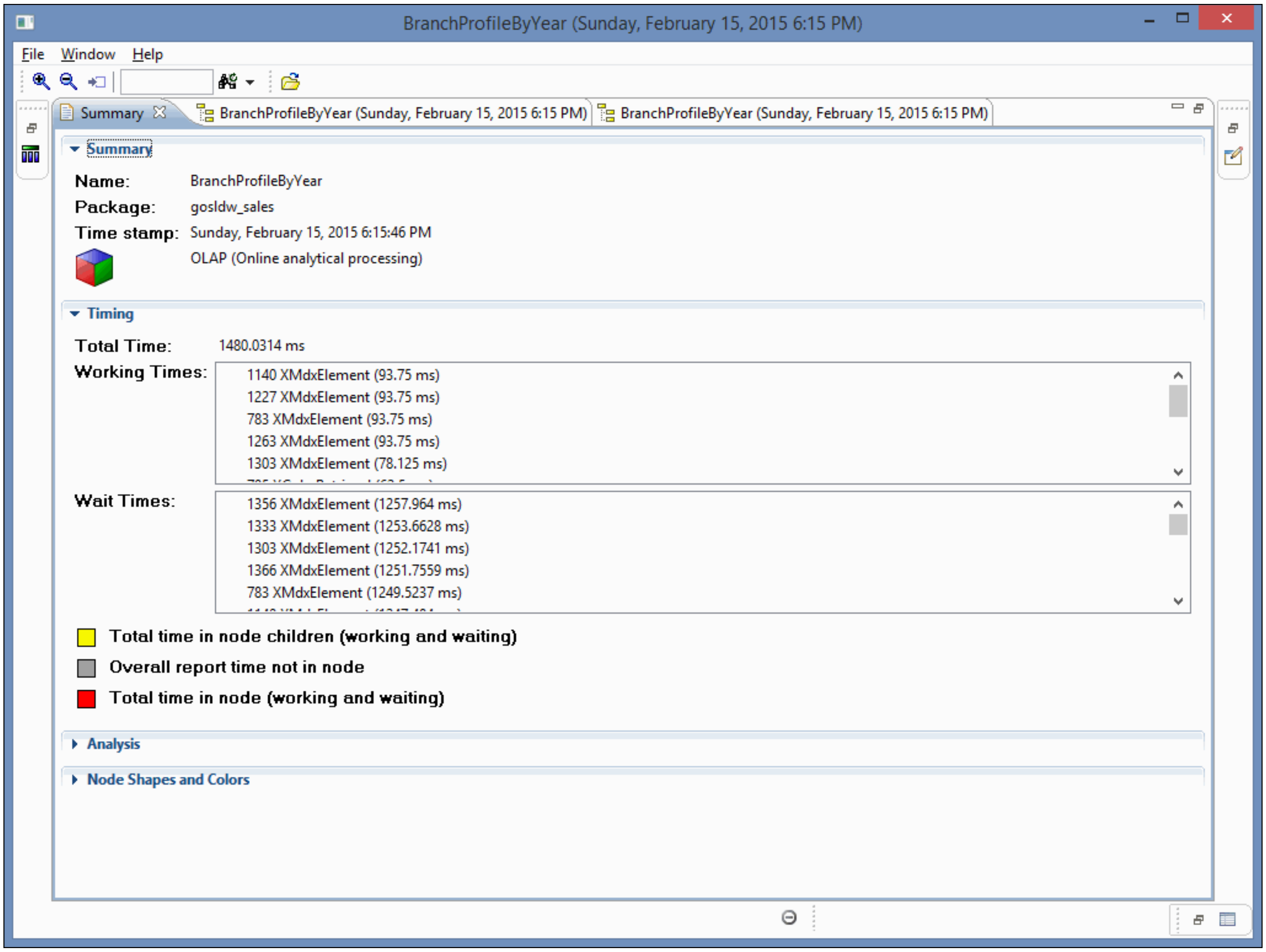


Figure 14-16 Summary view

Query view

The Query view shows the Multidimensional Expression Language (MDX) or SQL query that was run to generate a report. You can test the SQL from within DQA by clicking the **Re-run the SQL** statement on the server option. The MDX query is much more tightly linked to the report execution and can be used to find the places in the graph that match the commands in the MDX. Selecting an MDX command in the Query view highlights the corresponding node in the graph, assuming that you enabled the **Link MDX to graph** setting.

Figure 14-17 shows an MDX command selected in the Query view and after clicking the **Link MDX to graph** icon (highlighted in red).

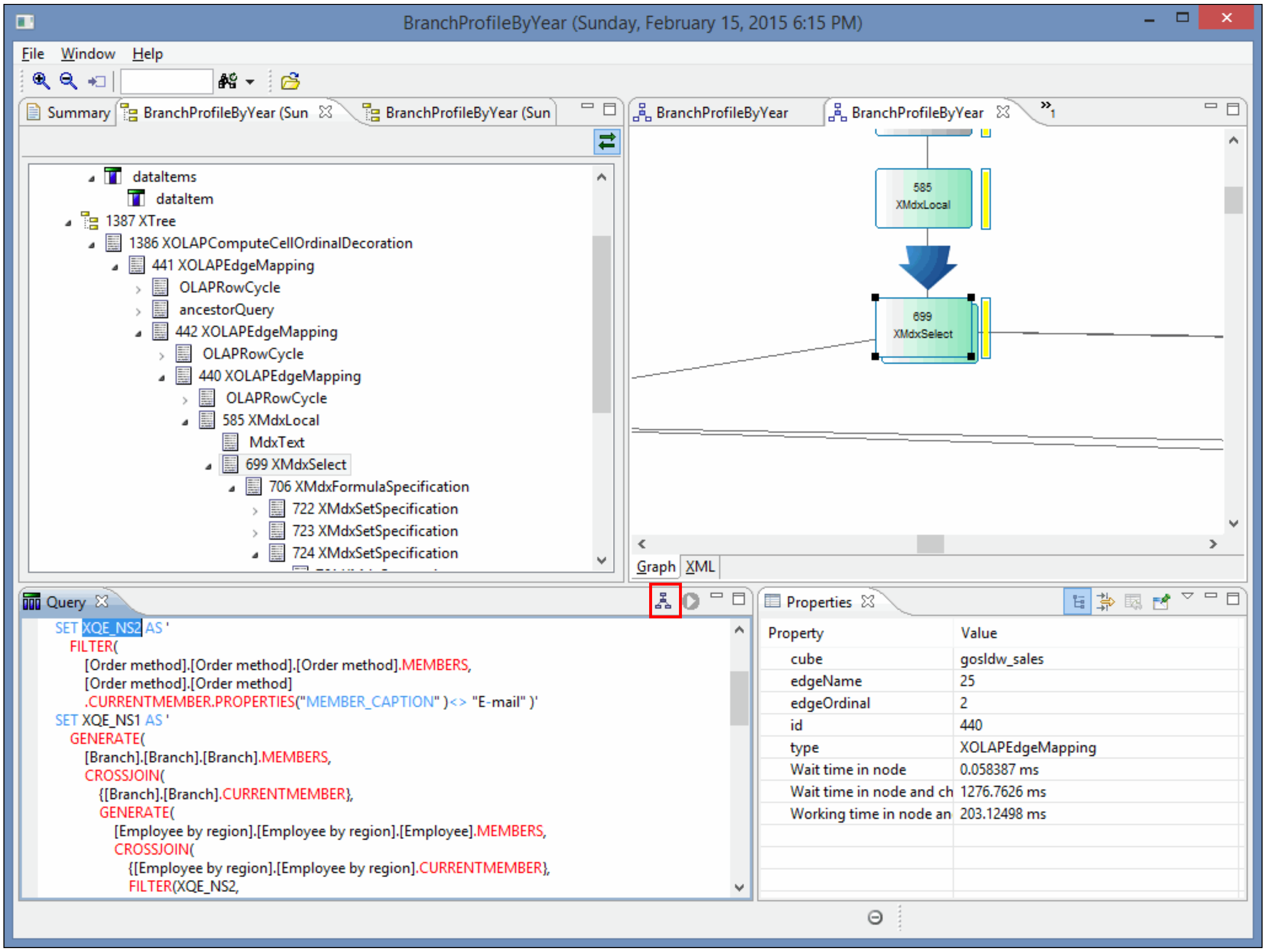


Figure 14-17 Query view

Content Store view

The Content Store view shows the reports that are available in your environment and enables you to run any report directly from DQA. To open the Content Store view, select **File** → **Open IBM Cognos Portal**. The list of folders and reports is the same as that in the Cognos Connection web portal. Unlike reports run from the portal, reports run from Dynamic Query Analyzer can generate logs on a report-by-report basis.

After a report runs, you can see the log for the run under the listing for the report in the content store view.

Figure 14-18 shows the content store view with the log of a report run beneath the report entry.

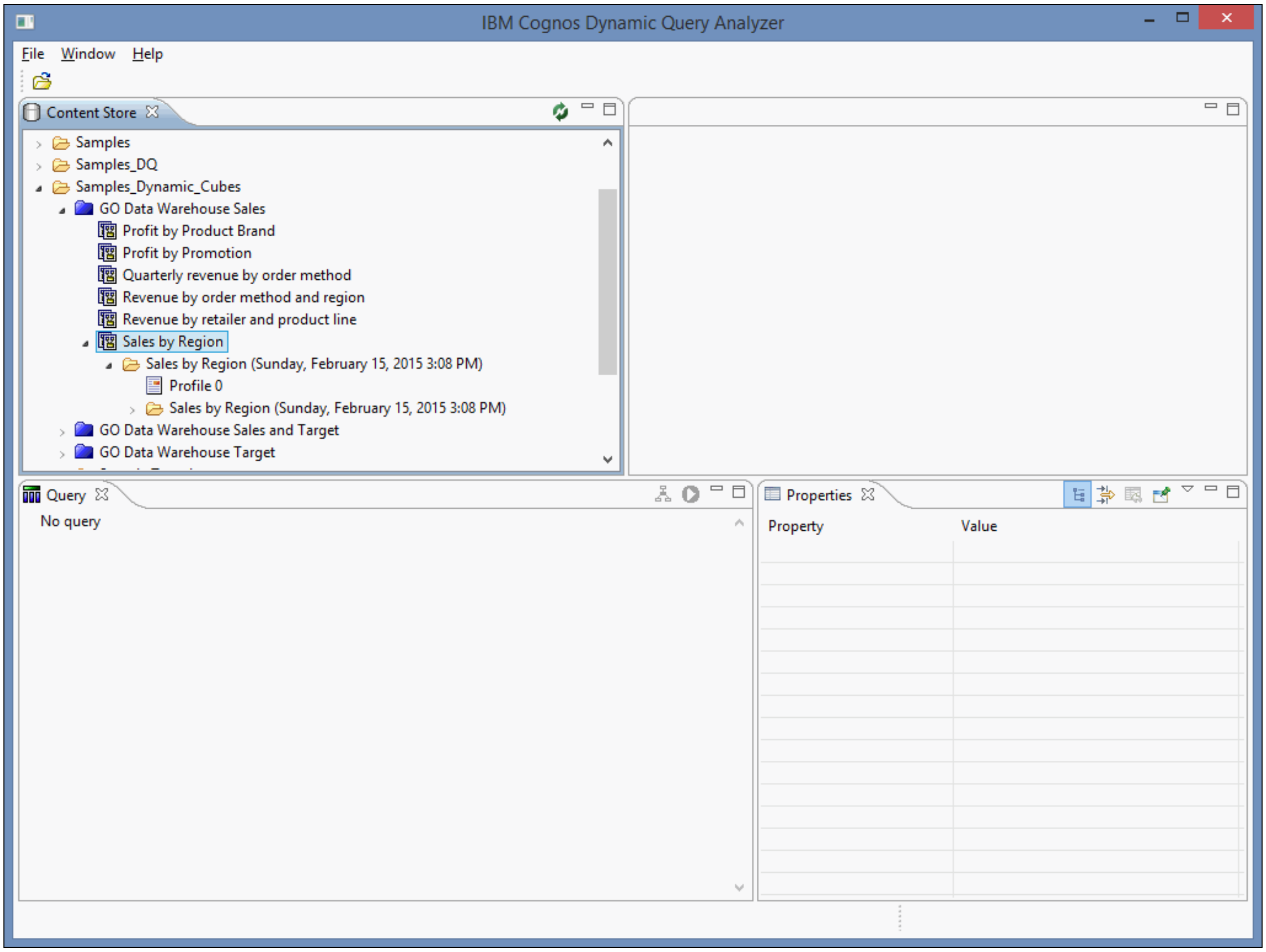


Figure 14-18 Content Store view

14.3 Enabling query execution tracing

This section describes the two methods to enable query execution trace. You can use this type of trace to troubleshoot reports or to find which dynamic cube caches were used during the query execution.

> **Note:** The query execution trace writes information such as the native SQL or MDX statements, and execution times in nanoseconds. This log is different from workload log. For information about workload log file, see "Workload log file" on page 348.

14.3.1 Enabling query execution tracing on a report basis from DQA

You can use DQA to selectively set a query execution trace, run reports, and review the generated logging messages. DQA only traces the reports that it runs, minimizing the performance impact on the Cognos server. In addition, system administrator privileges are not required to enable query execution trace.

To enable query execution tracing for a report run from DQA, complete the following steps.:

1. Start DQA.
2. Click **Windows** → **Preferences**.
3. Click **General** from the Preferences window (Figure 14-19).

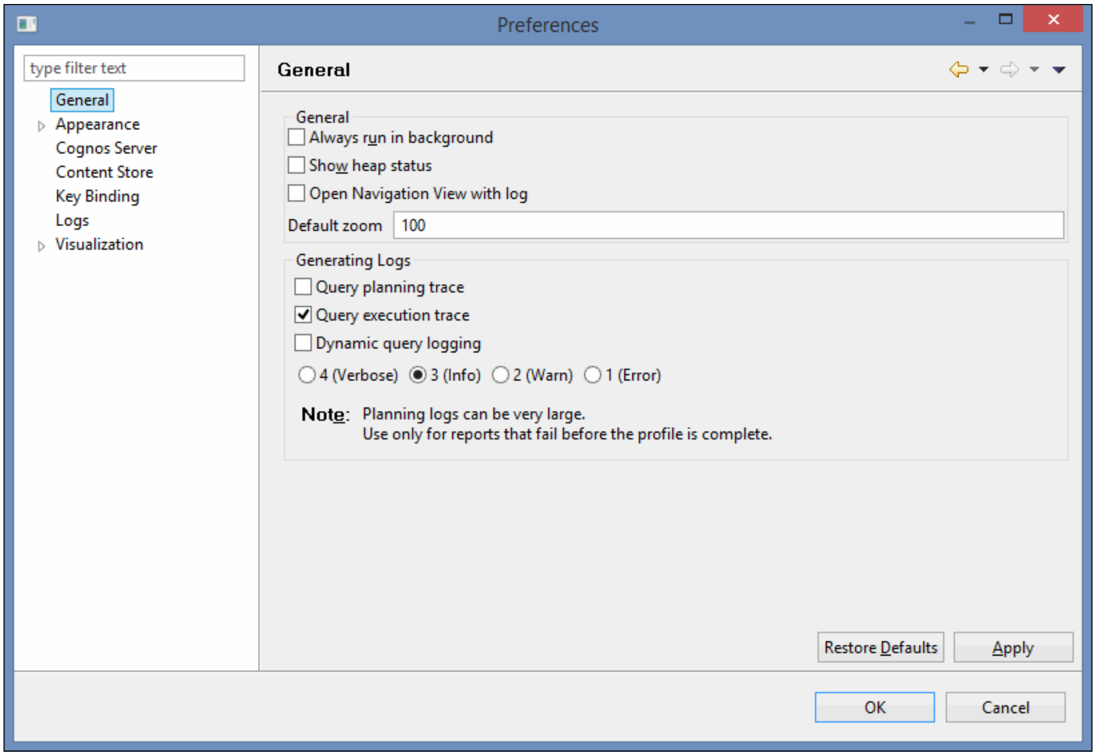


Figure 14-19 Set DQA query execution tracing

4. Click **Query execution trace**.
5. Set the logging level.
6. Click **OK** to apply changes and close the Preferences window.

Note: Query execution tracing is enabled by default in DQA.

Disabling DQA query execution tracing

Disable query execution tracing by using these steps:

1. Click to clear the **Query execution trace** check box.
2. Click **OK** to disable tracing and close the Preferences window.

14.3.2 Enabling query execution tracing for all requests

Query execution tracing can also be enabled for all requests to a designated query service. This trace is configured from Cognos Administration. Appropriate system administration privileges are required to enable query execution.

Important: Enabling query execution tracing for all request can result in the creation of large log files and can affect overall query performance. Disable query execution tracing as soon as you capture the information that you need.

Use the following procedure to enable query execution for a target query service, tracing all requests including cube startup:

1. Open Cognos Connection.
2. Click the **Launch** menu and select **IBM Cognos Administration**.

   If IBM Cognos Administration is not available as an option from the **Launch** menu, and you believe you should have access to Cognos system administration functions, contact your Cognos systems administration team.
3. Select **Status**.
4. Select **System**.
5. Select the Cognos server and dispatcher that is running Cognos Dynamic Query Mode (query service) and the dynamic cube that you are targeting for query execution tracing.
6. Select **QueryService**.
7. Select **Set properties** from the query service menu.
8. Select the Settings tab.
9. Select the check box in the Value column for **Enable query execution trace** to establish server-based query execution tracing. See Figure 14-20.
10. Click **OK** to enable tracing and return to Cognos Connection.

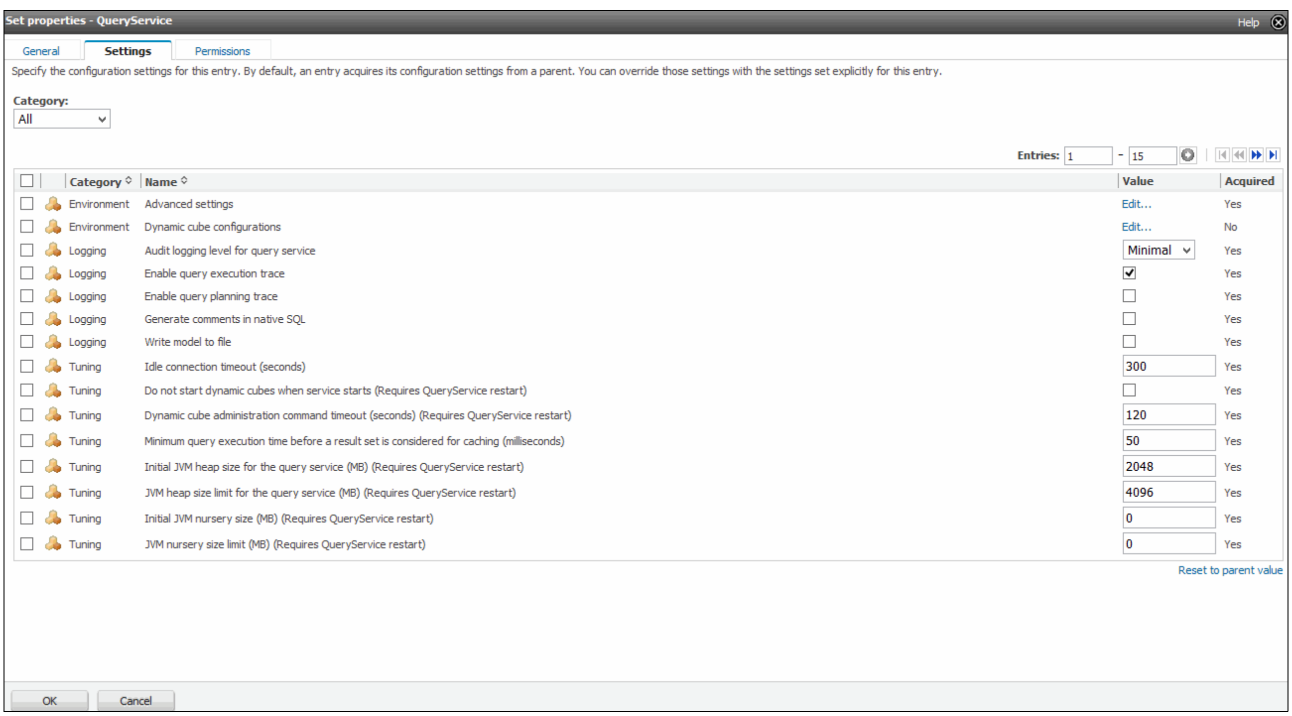


Figure 14-20 Enable query service query execution tracing

Disabling server query execution tracing

Disable server query execution tracing by using these steps:

1. Click to clear the check box for **Enable query execution trace**.
2. Click **OK** to disable tracing and return to Cognos Connection.

14.4 Enabling query plan tracing

To supplement query execution tracing, you might need to also enable query plan tracing. Query plan tracing can be enabled for individual reports run through DQA. To enable query plan tracing, select **Enable query execution trace** in the **Preferences** → **General** window available from the **Window** menu option.

> **Important:** This step is normally performed with guidance from the IBM support team. The planning logs can be large, and therefore have an impact on query performance when this setting is enabled.

14.5 The query service log file

The information in the log file for the query service managing a dynamic cube, and the associated report executions can provide insight into issues during report execution that are not available from the query execution log. The query service log messages are recorded under the `<CognosInstall>/Logs/XQE` folder in files with the following name format:

```
xqelog-<date>-<timestamp>.xml
```

A new log is created when the query service is started. A new log is also created when the file size reaches 10 MB. To modify this limit, edit the `configuration/xqe.diagnosticlogging.xml` file, and change the value specified in the following parameter:

```
<param name="rollOverSize" value="10000"/
```

14.6 Enabling Cognos Dynamic Cubes logging

This section describes how to enable logging for dynamic cubes to capture information that can help to diagnose issues.

Dynamic cubes use the same logging mechanism as the query service. The logging is configured by using the `configuration/xqe.diagnosticlogging.xml` file. This file uses the concept of event groups to distinguish between the different areas of the dynamic cubes or query service that can be logged.

To enable a specific event group, find the following entry in the `xqe.diagnosticlogging.xml` file:

```
<component name="XQE">
```

Modify the event group entry with the new level value. If the event group does not exist, add an event group entry with the following format:

```
<eventGroup name="Name of the Event Group" level="Logging Level"/>
```

Event groups can be enabled with one of the following levels:

- `None`: Nothing is logged for this event group.
- `Info`: This level contains detailed status messages.
- `Warn`: This level contains messages about recoverable errors.
- `Error`: This level contains messages about unrecoverable errors. These error log messages are usually logged regardless of whether the event group has been enabled.
- `Trace`: All possible log messages for this event group are logged. This log level is usually used for diagnostic purposes and should not be enabled in a normally operating production environment.

> **Note:** Starting with IBM Cognos BI V10.2.1 Fix Pack 3, a restart of the query service is not required for the changes to the `xqe.diagnosticlogging.xml` file to take effect. Table 1-1 on page 3 lists the Cognos Dynamic Cubes features introduced in the different releases of Cognos BI.

Example 14-2 shows how to set “`ROLAPCubes.Management`” at the `trace` level:

Example 14-2 Setting ROLAPCubes.Management at trace level

```
<component name="XQE">
...
<eventGroup name="ROLAPCubes.Management" level="trace"/>
```

The logging information is written to the query service log file described in 14.5, “The query service log file” on page 483.

The following sections describe several event groups that can be enabled to troubleshoot common problems.

14.6.1 Query service environment variables and JVM heap size

If a dynamic cube fails to start, you can enable the "`InitTerm`" event group, which causes the query service to write additional information about the initialization of a dynamic cube to the log file. The information that this event group provides includes environment variable values and JVM heap size configuration used by the query service.

Change the event group or groups as follows:

```
<eventGroup name="InitTerm" level="trace"/>
```

Example 14-3 shows the logging information produced showing environment variable values.

Example 14-3 Sample log entries showing environment variables

```
<event ...><![CDATA[Starting QueryService with the following Environment
variables: {LSHOST=demoibm.com, ..., SystemDrive=C:, TEMP=C:\Windows\TEMP,
windows_tracing_flags=3, ProgramFiles=C:\Program Files,
Path=C:\dialer\bin;C:\Windows\system32; ...
USERNAME=VLAUN42743522$,DB2INSTANCE=DB2,}]]></event>
```

Example 14-4 shows sample logging information with JVM heap sizes.

Example 14-4 Logging information showing JVM heap sizes

```
<event ...><![CDATA[Using JVM heap sizes: -Xms2048m, -Xmx10240m, -Xmnx5120m,
-Xmns1024m]]></event>
```

14.6.2 Dynamic cube advanced settings configuration

When you are diagnosing a problem, confirm the configuration parameters that are applied to a cube. Among other information, enabling the "`ROLAPCubes.Management`" event group writes the configuration parameters of the cube to the log file.

Change the event group or groups as follows:

```
<eventGroup name="ROLAPCubes.Management" level="trace"/>
```

Logging information similar to that shown in Example 14-5 is returned.

Example 14-5 Sample log entries after enabling the ROLAPCubes.Management group

```
<event ...><![CDATA[Configured ROLAP advanced settings values: qsMax
CubeLoadThreads=16,...]]></event>
<event ...><![CDATA[[Updating a ROLAP configuration parameter:
rolap.qsMultiDimensionalQuerySizeLimit=0]]]></event>
<event ...><![CDATA[[Before: ROLAP configuration:
[ROLAPConfiguration:...,qsMultiDimensionalQuerySizeLimit=0,...]]]]></event>
<event ...><![CDATA[[After: ROLAP configuration:
[ROLAPConfiguration:...,qsMultiDimensionalQuerySizeLimit=0,...]]]]></event>
```

14.6.3 Dynamic cube management: Start, stop

Depending on the problem being diagnosed, a useful step is to track the various administrative commands that are issued to a cube, such as **`start`** and **`stop`**. Enabling the "`ROLAPCubes.Management`" and "`ROLAPCubes.Info`" event groups adds this information to the log file.

Change the event group or groups as follows:

```
<eventGroup name="ROLAPCubes.Management" level="trace"/>
<eventGroup name="ROLAPCubes.Info" level="info"/>
```

Logging information similar to that shown in Example 14-6 is returned.

Example 14-6 Log entries enabling the ROLAPCubes.Management and ROLAPCubes.Info groups

```
<event ...><![CDATA[Started cube start with source cubes manager task,
cubes=[gosldw_sales]]]></event>
<event ... rolapCube="gosldw_sales"><![CDATA[Cube start executing]]></event>
<event ... rolapCube="gosldw_sales" ...><![CDATA[Cube start succeeded]]></event>
<event ...><![CDATA[Manager task retrieved the status of the cube start with
source cubes task for cube gosldw_sales.  The cube start with source cubes task
was successful.]]></event>
```

```
<event ...><![CDATA[Finished cube start with source cubes manager task,
cubes=[gosldw_sales]]]></event>
```

14.6.4 Dynamic cube hierarchy load

If a cube encounters problems when it starts, obtain additional information about the loading of each of the hierarchies in the cube, one of the primary activities during cube start. Enabling the "`ROLAPCubes.Loader`" event group adds information about the loading of hierarchies to the log file. You can compare the time stamp information in each message to calculate the time that is spent in each step of the process.

Change the event group or groups as follows:

```
<eventGroup name="ROLAPCubes.Loader" level="trace"/>
```

Example 14-7 shows logging information for the process to load the branch dimension started.

Example 14-7 Log entries showing the process to load the branch dimension started

```
<event ... timestamp="2015-04-30 13:58:47.029" ... rolapCube="gosldw_sales"
rolapDimension="Branch" rolapHierarchy="Branch" ...>
<![CDATA[Hierarchy load began.]]></event>
   <event ... timestamp="2015-04-30 13:58:47.043" ... rolapCube="gosldw_sales"
rolapDimension="Branch" ... rolapHierarchy="Branch" ...><![CDATA[Loading level
members for [Branch].[Branch].[Branch]]]></event>
   <event ... timestamp="2015-04-30 13:58:47.044" ... rolapCube="gosldw_sales"
rolapDimension="Branch" rolapHierarchy="Branch" ...><![CDATA[Loading level members
for [Branch].[Branch].[Country]]]></event>
   <event ... timestamp="2015-04-30 13:58:47.044" ... rolapCube="gosldw_sales"
rolapDimension="Branch" rolapHierarchy="Branch" ...><![CDATA[Loading level members
for [Branch].[Branch].[Region]]]></event>
```

Example 14-8 shows log entries for a database query run to get members.

Example 14-8 Log entries for a query to get members

```
<event ... timestamp="2015-04-30 13:58:47.062" ... rolapCube="gosldw_sales"
rolapDimension="Branch" rolapHierarchy="Branch" ...><![CDATA[Started member
query.]]></event>
<event ... timestamp="2015-04-30 13:58:53.060" ... rolapCube="gosldw_sales"
rolapDimension="Branch" rolapHierarchy="Branch" ...><![CDATA[Finished member
query.]]></event>
```

Example 14-9 shows the log entries corresponding to loading members into the member cache.

Example 14-9 Logging showing loading members into the member cache

```
<event ... timestamp="2015-04-30 13:58:53.062" ... rolapCube="gosldw_sales"
rolapDimension="Branch" rolapHierarchy="Branch" ...><![CDATA[Started populating
level members for hierarchy [Branch].[Branch]]]></event>
<event ... timestamp="2015-04-30 13:58:54.770" ... rolapCube="gosldw_sales"
rolapDimension="Branch" rolapHierarchy="Branch" ...><![CDATA[[Branch].[Branch]
loaded 55 members]]></event>
   <event ... timestamp="2015-04-30 13:58:54.781" ... rolapCube="gosldw_sales"
rolapDimension="Branch" rolapHierarchy="Branch" ...><![CDATA[0 enqueues when pipe
full out of 2 total enqueues for hier Branch.]]></event>
   <event ... timestamp="2015-04-30 13:58:54.781" ... rolapCube="gosldw_sales"
rolapDimension="Branch" rolapHierarchy="Branch" ...><![CDATA[Finished populating
level members for hierarchy [Branch].[Branch]]]></event>
   <event ... timestamp="2015-04-30 13:58:54.782" ... rolapCube="gosldw_sales"
rolapDimension="Branch" rolapHierarchy="Branch" ...><![CDATA[Started priming level
members for hierarchy [Branch].[Branch]]]></event>
   <event ... timestamp="2015-04-30 13:58:54.785" ... rolapCube="gosldw_sales"
rolapDimension="Branch" rolapHierarchy="Branch" ...><![CDATA[Finished priming
level members for hierarchy [Branch].[Branch]]]></event>
   <event ... timestamp="2015-04-30 13:58:54.785" ... rolapCube="gosldw_sales"
rolapDimension="Branch" rolapHierarchy="Branch" ...><![CDATA[Started building
level keys to member map for hierarchy [Branch].[Branch]]]></event>
   <event ... timestamp="2015-04-30 13:58:54.786" ... rolapCube="gosldw_sales"
rolapDimension="Branch" rolapHierarchy="Branch" ><![CDATA[Finished building level
keys to member map for hierarchy [Branch].[Branch]]]></event>
   <event ... timestamp="2015-04-30 13:58:54.786" ... rolapCube="gosldw_sales"
rolapDimension="Branch" rolapHierarchy="Branch" ... ><![CDATA[Hier cache info for
Branch]]></event>
```

Example 14-10 shows that the branch dimension load completed successfully.

Example 14-10 Log entries branch dimension successfully loaded

```
<event ... timestamp="2015-04-30 13:58:54.786" ... rolapCube="gosldw_sales"
rolapDimension="Branch" rolapHierarchy="Branch" ...><![CDATA[Hierarchy load
succeeded.]]></event>
```

14.6.5 SQL queries statement

Enabling the "QueryService.SQL" event group set at trace level logs the SQL queries that are run to populate the member cache, aggregate cache, and data cache. By using the trace level, you can look at the SQL statements and determine whether the hierarchy has been modeled correctly with the correct level keys and attributes.

Change the event group as follows:

```
<eventGroup name="QueryService.SQL" level="trace"/>
```

Example 14-11 shows logging information that was produced when a query was run to load the members of the `Time (close date)` hierarchy.

Example 14-11 Log entries showing query to load members of Time (close date) hierarchy

```
<event ... rolapCube="gosldw_sales" rolapDimension="Time (close date)"
rolapHierarchy="Time (close date)" ... ><![CDATA[SQL query started with arguments
... SQL: "SELECT
    "GO_TIME_DIM"."CURRENT_YEAR" AS "Current_year__close_date_",
   ...
FROM
    "GOSALESDW"."gosalesdw"."GO_TIME_QUARTER_LOOKUP" "GO_TIME_QUARTER_LOOKUP"
        INNER JOIN "GOSALESDW"."gosalesdw"."GO_TIME_DIM" "GO_TIME_DIM"
        ON "GO_TIME_QUARTER_LOOKUP"."QUARTER_KEY" = "GO_TIME_DIM"."QUARTER_KEY"
GROUP BY
    "GO_TIME_DIM"."CURRENT_YEAR",
     ...
    "GO_TIME_DIM"."WEEKDAY_TR"
ORDER BY
    "Current_year__close_date_" ASC,
    "Quarter_key__close_date_" ASC,
    "Month_key__close_date_" ASC,
    "Day_key__close_date_" ASC"]]></event>
```

Example 14-12 shows logging information from running a query to load the data cache.

Example 14-12 Log entries for a query ran to load the data cache

```
<event ... rolapCube="gosldw_sales" ><![CDATA[SQL query started with arguments ...
SQL: "SELECT
    "MRK_CAMPAIGN_LOOKUP"."CAMPAIGN_CODE" AS "Promotions_Campaign_code",
    SUM("SLS_SALES_FACT"."UNIT_SALE_PRICE") AS "Unit_sale_price",
    SUM("SLS_SALES_FACT"."GROSS_PROFIT") AS "Gross_profit",
    SUM("SLS_SALES_FACT"."UNIT_COST") AS "Unit_cost",
    SUM("SLS_SALES_FACT"."UNIT_PRICE") AS "Unit_price"
FROM
    "GOSALESDW"."gosalesdw"."MRK_PROMOTION_DIM" "MRK_PROMOTION_DIM"
        INNER JOIN "GOSALESDW"."gosalesdw"."MRK_CAMPAIGN_LOOKUP"
"MRK_CAMPAIGN_LOOKUP"
        ON "MRK_PROMOTION_DIM"."CAMPAIGN_CODE" =
"MRK_CAMPAIGN_LOOKUP"."CAMPAIGN_CODE"
            INNER JOIN "GOSALESDW"."gosalesdw"."SLS_SALES_FACT" "SLS_SALES_FACT"
            ON "MRK_PROMOTION_DIM"."PROMOTION_KEY" =
"SLS_SALES_FACT"."PROMOTION_KEY"
GROUP BY
    "MRK_CAMPAIGN_LOOKUP"."CAMPAIGN_CODE""]]></event>
```

14.6.6 Dynamic cube timing

Enabling the "Timing" event group adds timing information about request execution, including report execution.

The information in this event group includes the time in milliseconds to process the request. It also includes the number of bytes retrieved from each cache, the number of rows, and the response time for each SQL statement. This information is useful when trying to determine where the report request is taking more time to retrieve data, and how much data is returned.

Add event group or groups as follows:

```
<eventGroup name="Timing" level="info"/>
```

Example 14-13 shows logging information that is returned from running a query from IBM Cognos Workspace Advanced. Because this query was never saved, the report name in the message appears as unknown.

Example 14-13 Log entries for a query run from IBM Cognos Workspace Advanced

```
<event ...><![CDATA[Total request time 1,925ms returning 4,628 bytes for report
unknown.
----Cube gosldw_sales
Cache threads waited 0ms enqueing cells to the pipelineResultSet.
ResultSetCache: 1 requests in 0ms with 0 hits.
ExpressionCache: 0 requests in 0ms with 0 hits.
DataCache: 1 requests in 17ms solved 100 (95 non null) out of 300 tuples.
AggregateCache: 0 requests in 0ms solved 0 (0 non null) out of 0 tuples.
SQL Statement Summary: 2 calls took 1,415ms to execute, 178ms to fetch 190 rows, 0
rows discarded, resulting in 475 values.
 #1: SQL query of type 'Fact' took 502ms to execute, 160ms to fetch 95 rows, 0
rows discarded, 1 measures resulting in 95 values.
 #2: SQL query of type 'Fact' took 913ms to execute, 18ms to fetch 95 rows, 0 rows
discarded, 4 measures resulting in 380 values.]]></event>
```

Example 14-14 shows the logging information returned from running a report against a virtual cube.

Example 14-14 Logging of a report run against a virtual cube

```
<event ...><![CDATA[Total request time 1,557ms returning 2,557 bytes for report
Revenue vs Sales Target 2012.
----Cube gosldw_target
Cache threads waited 0ms enqueing cells to the pipelineResultSet.
ResultSetCache: 0 requests in 0ms with 0 hits.
ExpressionCache: 0 requests in 0ms with 0 hits.
DataCache: 1 requests in 0ms solved 0 (0 non null) out of 5 tuples.
AggregateCache: 0 requests in 0ms solved 0 (0 non null) out of 0 tuples.
SQL Statement Summary: 1 calls took 683ms to execute, 1ms to fetch 5 rows, 0 rows
discarded, resulting in 5 values.
 #1: SQL query of type 'Fact' took 683ms to execute, 1ms to fetch 5 rows, 0 rows
discarded, 1 measures resulting in 5 values.
----VirtualCube gosldw_sales_and_target
Cache threads waited 0ms enqueing cells to the pipelineResultSet.
ResultSetCache: 1 requests in 0ms with 0 hits.
ExpressionCache: 0 requests in 0ms with 0 hits.
DataCache: 1 requests in 0ms solved 0 (0 non null) out of 10 tuples.
VC merge operations: 1 requests in 15ms with 10 cells.
----Cube gosldw_sales
Cache threads waited 0ms enqueing cells to the pipelineResultSet.
ResultSetCache: 0 requests in 0ms with 0 hits.
ExpressionCache: 0 requests in 0ms with 0 hits.
DataCache: 1 requests in 0ms solved 0 (0 non null) out of 5 tuples.
AggregateCache: 0 requests in 0ms solved 0 (0 non null) out of 0 tuples.
SQL Statement Summary: 1 calls took 300ms to execute, 1ms to fetch 5 rows, 0 rows
discarded, resulting in 5 values.
 #1: SQL query of type 'AggregateTable' took 300ms to execute, 1ms to fetch 5
rows, 0 rows discarded, 1 measures resulting in 5 values.]]></event>
```

14.6.7 Query Performance

The `"ROLAPQuery.Performance"` event group logs messages related to query performance for dynamic cubes queries. The ROLAPQuery. Performance log entry introduces the `op` and `duration` attributes.

The `op` attribute value is either START or END, allowing the user to see when a specific query action starts and ends in the log. The `duration` attribute value is the difference in time (in milliseconds) between the start and end of a specific query action.

When this event group is enabled, it logs details about the query and the amount of time spent in specific areas of query execution. This information is invaluable in debugging the performance of a cube. Enable both `"ROLAPQuery.Performance"` and `"Timing"` to troubleshoot dynamic cubes queries effectively.

Add the following event group or groups:

```
<eventGroup name="ROLAPQuery.Performance" level="trace"/>
```

Example 14-15 shows the logging information returned.

Example 14-15   Logging information produced by adding the ROLAPQuery.Performance group

```
<event ... timestamp="2015-02-15 15:08:04.700" op="START" ...
rolapCube="gosldw_sales" ...><![CDATA[Started iterating result set for data query:
Sales by Region: Query1]]></event>
...
<event ... timestamp="2015-02-15 15:08:18.993" op="END" duration="14369" ...
rolapCube="gosldw_sales" ... ><![CDATA[Finished execution of Query Strategy for
report Sales by Region
   Requested 20 tuples
   Found 0 tuples in data cache
   Found 0 tuples in aggregate cubes
   Fetched 20 tuples from database
   0 of the tuples were from pushdown queries
   Created 0 tuples from the aggregate cache
      Total time processing aggregates : 0 (ms) using 0 tasks
            0 aggregate cells rolled up into 0 separate locations
]]></event>
```

The `"ROLAPQuery.Performance"` event group log messages include a breakdown of the categories where the result tuples came from and how many tuples were returned per category. They have the following categories:

- Data Cache

  This category contains the number of tuple values obtained from the in-memory data cache. The values are read from the in-memory data cache that is populated from previously run queries. The tuple values from this category are considered data from a warm cache.

- Aggregate Cube

  This category includes the number of tuple values obtained from the in-database aggregates. The values are returned from running an SQL query against the aggregate or summary tables found in the underlying RDBMS database.

- Database

  This category describes the tuple values obtained from running an SQL query against the underlying database. The tuple values from this category are considered data from an empty or cold cache.

- Pushdown Query

  This category is composed of a subset of tuple values that are obtained from running an SQL query against the underlying database in which the query is pushed down to the database for processing instead of locally processed in the MDX engine.

  Pushdown queries are only supported in queries with FILTER, TOPCOUNT, or TOPSUM function calls.

- Aggregate Cache

  This category contains the number of tuples that are obtained from the in-memory aggregates. The values are read from the in-memory aggregate cache.

14.6.8 Virtual Cubes

The "ROLAPCubes.Virtual" event group logs information about virtual cubes including how a virtual cube is initialized, how a virtual cube merges its source cubes, the source objects that are hidden (if any), and the queries that are sent to the source cube.

By default, this event group logs warning messages such as invalid source dimension, hierarchy, level, or members detected during the creation of a virtual cube. It also logs a warning message when a virtual hierarchy that does not have an ALL member and does not merge hierarchies from all the source cubes is created.

Enable the "ROLAPCubes.Virtual" event group with the level value set to trace to help diagnose issues, for example as a virtual cube merging incorrectly or returning unexpected results such as only returning data from one source cube.

This event group logs detailed messages about how the source objects are being merged. If a dimension, hierarchy, level, or member is not merging properly, the resulting virtual tuple log entry could explain why queries are returning only the data from one source cube. Also, look for log messages where one or more dimensions from a source cube are not merged, if a hierarchy is hidden and not defaulted, or a hierarchy is not hidden and defaulted. These issues could also be the reason why a query does not run against a source cube.

Change the event group or groups as follows:

<eventGroup name="ROLAPCubes.Virtual" level="info"/>

Example 14-16 shows logging information that is created from a query run against the gosldw_sales source cube.

Example 14-16 Logging from running a query against a source cube

```
<event ... rolapCube="gosldw_target"...><![CDATA[Executing source cube query for set:
CJS::0{
([Measures].[Sales target])
}1{
([Product type].[Product type].[All].[992]),
...
([XQE_VD2].[VL].[VirtualMember])
}]]></event>
```

Example 14-17 shows logging information that is created from query ran against the gosldw_sales source cube.

Example 14-17 Logging from running a query against the gosldw_sales source cube

```
<event ... rolapCube="gosldw_sales" ... ><![CDATA[Executing source cube query for set:

CJS::0{

([Measures].[Revenue])

}1{

([Products].[Products].[All].[992]),
```

```
...
([Sales order].[Sales order].[All])
}]]></event>
```

14.6.9 Additional event groups

For more information about IBM Cognos Dynamic Cubes event group logging settings, see *IBM Business Analytics Proven Practices: IBM Cognos Dynamic Cube Advanced Logging Settings* at:

http://www.ibm.com/developerworks/library/ba-pp-infrastructure-cognos_specific-page655

14.6.10 Inspecting cube start

Using event groups, you can review the start sequence and queries that are issued to the source database to retrieve dimension members and their associated attributes. "Using the query service log to understand where time is spent" on page 423 describes troubleshooting techniques based on using more verbose logging levels and examining the query service log to see where time in the query service is spent during queries that run against the database. You can apply the techniques described in that section to inspect the cube start sequence and queries.

14.7 Reviewing Cognos reports with DQA

This section describes how you can use DQA to run a report, and review the query execution trace. This chapter uses the reports included in the IBM Cognos BI V10.2.2 samples. See Chapter 5, "Basic modeling" on page 105 for more information about the samples.

Perform the following steps to set query execution trace, run a report, and review trace logging:

1. Open DQA.
2. Enable query execution trace as described in the section 14.3.1, "Enabling query execution tracing on a report basis from DQA" on page 480.
3. Click **Window** → **Preferences**.
4. Click **Content Store** → **Open the latest log** to have DQA automatically open the most recent query execution log associated with a report run. The default preference is to open the most recent log that follows a report run.
5. Verify that the Content Store view is visible. Click **File** → **Open IBM Cognos Portal**.

> **Note:** You can also make the Content Store view visible from the Show View window:
> 1. Click **Window** → **Show View.**
> 2. Select **Content Store** from the Navigation folder in the Show View window.

3. In the Content Store view, expand the folders **Sample_Dynamic_Cube** → **Reports** to browse the available Cognos content and locate the target report `BranchProfileByYear`.
4. Right-click the menu on the BranchProfitByYear report, and select **Run Report**.
5. The report output is displayed in a new browser tab.

   Figure 14-21 shows the DQA Report View.

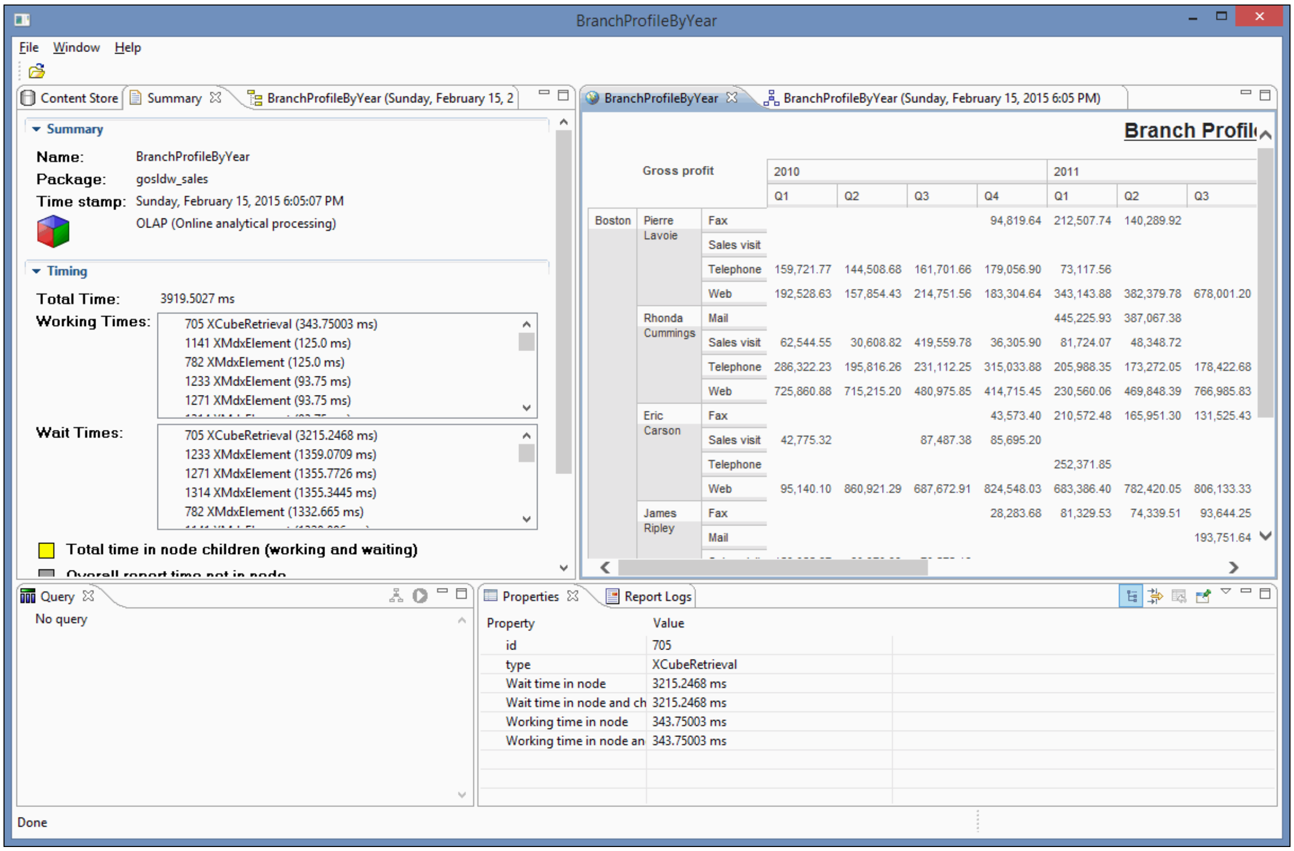


Figure 14-21 DQA Report View

6. The report execution tree that is associated with the report opens in a new graph view.

> **Note:** A report query execution trace can also be loaded from the Report Logs view. Display the Report Logs view (if it is not visible) and load a report query execution trace:
>
> 1. Click **Window** → **Show View.**
> 2. Select **Report Logs** from the Navigation folder in Show View window.
> 3. Click **OK** to close the window.
> 4. Browse the Report Logs view by report name and time stamp to locate the `report query execution trace` folder.
> 5. Expand the `report trace` folder and double-click **Profile 0**, which is the execution profile.

Figure 14-22 shows the DQA BranchProfitByYear query execution trace.

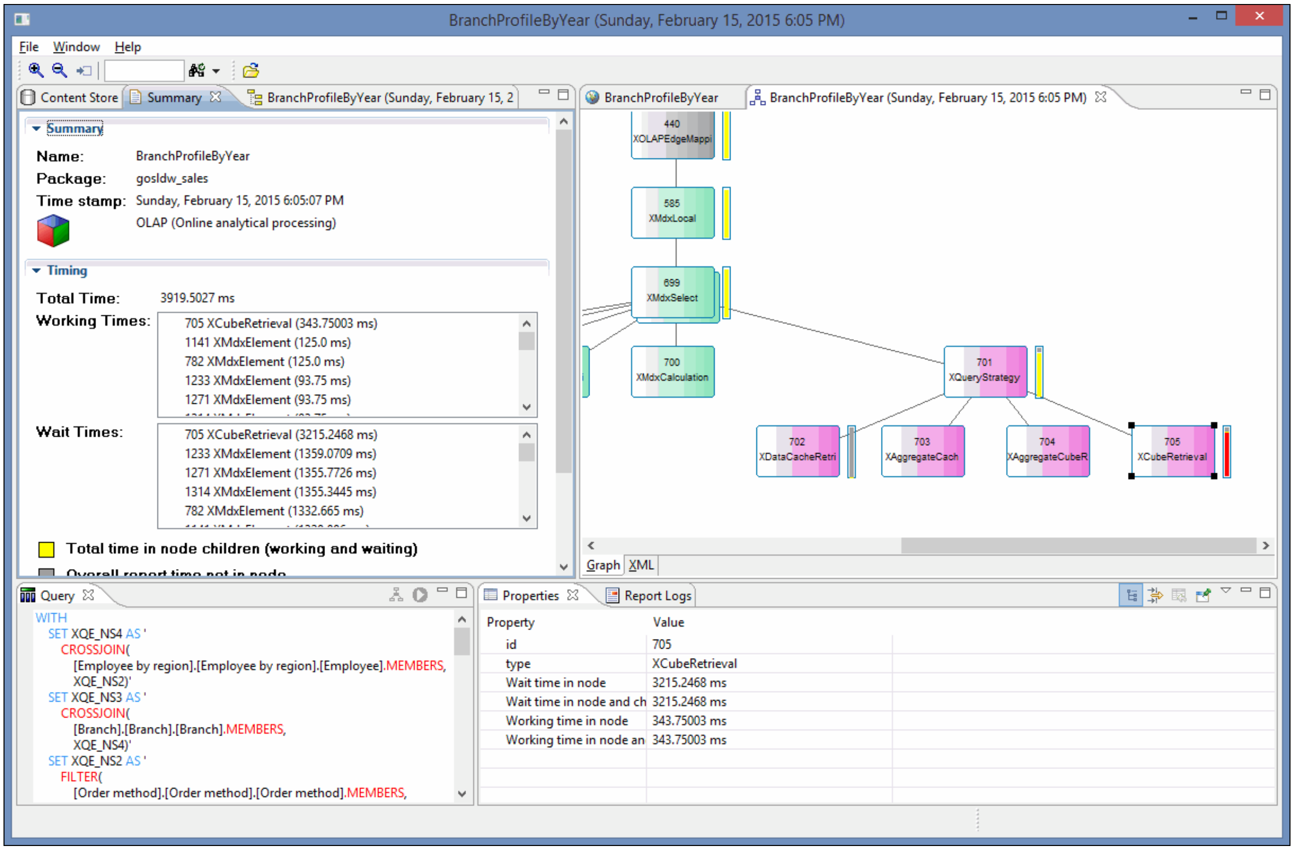


Figure 14-22 DQA BranchProfitByYear query execution trace

14.7.1 Reviewing the query execution trace

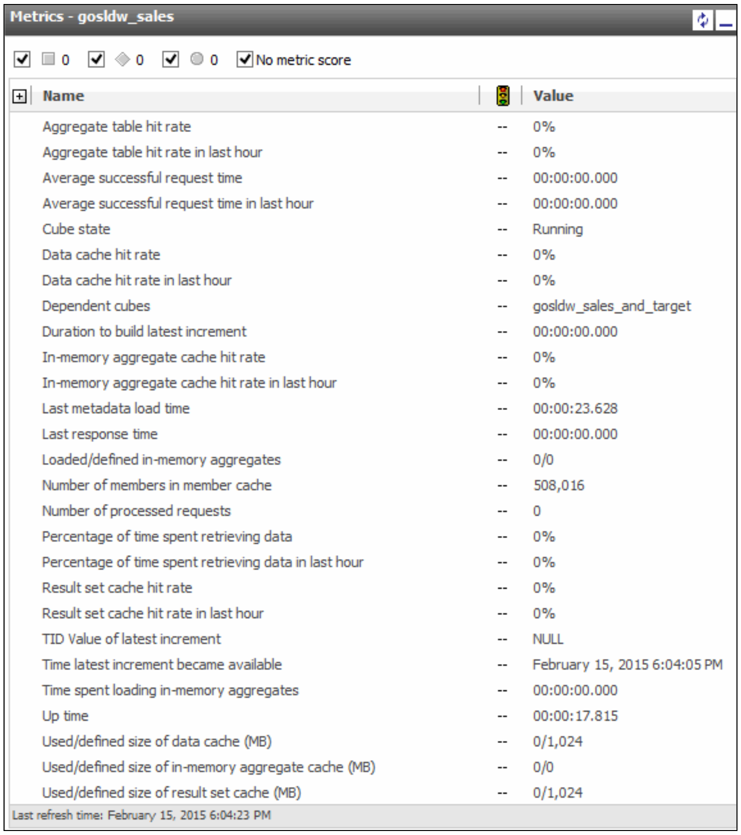
Chapter 2, “IBM Cognos Dynamic Cubes architecture” on page 13 describes details of the caching strategy of Cognos Dynamic Cubes and the data caches that are available to optimize performance.

Note: A query execution trace does not record entries against XQueryStrategy child nodes unless the dynamic cube data retrieval component they represent is used to satisfy the MDX data request for the report.

There is no result set cache or data cache available from which the results can be obtained because the report request for the BranchProfitByYear report is the first report to be requested. No aggregate table match is identified.

Reviewing the `golsldw_sales` dynamic cube monitor confirms the absence of a data cache hit ratio.

Figure 14-23 shows the `gosldw_sales` dynamic cube metrics.



Metrics - gosldw_sales

☑ 0 ☑ 0 ☑ 0 ☑ No metric score

Name		Value
Aggregate table hit rate	--	0%
Aggregate table hit rate in last hour	--	0%
Average successful request time	--	00:00:00.000
Average successful request time in last hour	--	00:00:00.000
Cube state	--	Running
Data cache hit rate	--	0%
Data cache hit rate in last hour	--	0%
Dependent cubes	--	gosldw_sales_and_target
Duration to build latest increment	--	00:00:00.000
In-memory aggregate cache hit rate	--	0%
In-memory aggregate cache hit rate in last hour	--	0%
Last metadata load time	--	00:00:23.628
Last response time	--	00:00:00.000
Loaded/defined in-memory aggregates	--	0/0
Number of members in member cache	--	508,016
Number of processed requests	--	0
Percentage of time spent retrieving data	--	0%
Percentage of time spent retrieving data in last hour	--	0%
Result set cache hit rate	--	0%
Result set cache hit rate in last hour	--	0%
TID Value of latest increment	--	NULL
Time latest increment became available	--	February 15, 2015 6:04:05 PM
Time spent loading in-memory aggregates	--	00:00:00.000
Up time	--	00:00:17.815
Used/defined size of data cache (MB)	--	0/1,024
Used/defined size of in-memory aggregate cache (MB)	--	0/0
Used/defined size of result set cache (MB)	--	0/1,024

Last refresh time: February 15, 2015 6:04:23 PM

Figure 14-23 Gosldw_sales dynamic cube metrics

14.7.2 Subsequent run requests: Taking advantage of the result set cache

As described in Chapter 2, “IBM Cognos Dynamic Cubes architecture” on page 13, running the BranchProfitByReport report again takes advantage of the query service result set cache, eliminating the need to request data from the dynamic cube, and minimizing the time that is recorded against the XQueryStrategy and child nodes.

To verify this behavior, use the following procedure to rerun the BranchProfitByYear report from DQA:

1. Select the **Content Store** view.
2. Expand the **Dynamic Cube** → **Reports** folder to explore the Cognos content view and locate the BranchProfitByYear report.
3. From the content object context, select **Run Report**.
4. Open the query execution trace generated for the most recent report run.

Figure 14-24 shows the BranchProfitByYear query execution trace.

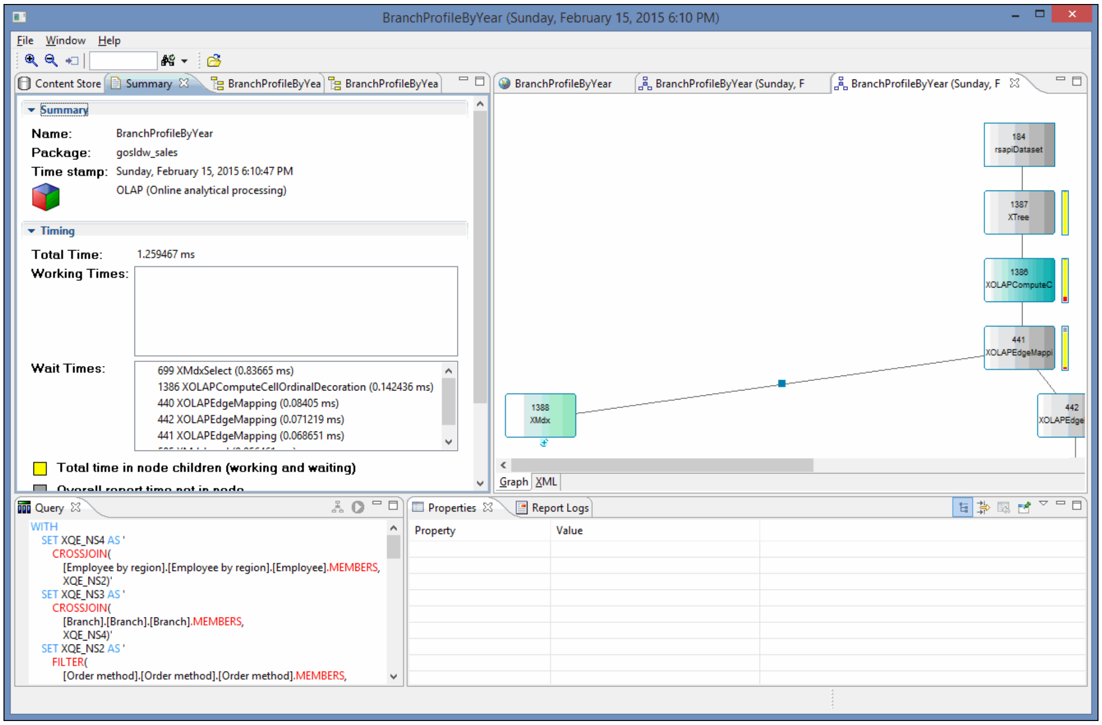


Figure 14-24 BranchProfitByYear query execution trace

The XMdxSelect node records a wait time of 0.83 milliseconds, and no values are recorded against the XQueryStrategy (or its child nodes).

The query result was taken from the available results in the result set cache.

14.7.3 Confirming the role of caching

To confirm the role of the result set cache and the presence of data in the data cache or the aggregate cache, the result set cache can be disabled and the report can be rerun.

The role of the result set cache can be confirmed by disabling the result set cache.

> **Note:** The dynamic cube must be restarted before the updated setting to disable the result set cache takes effect.

Perform the following steps:

1. Open IBM Cognos Administration.
2. Select **Status**.
3. Select **Dynamic Cubes**.
4. Select **All Server Groups**.
5. Select the server group that the server is in.
6. Select the dispatcher.
7. Select the cube.
8. In the **QueryService** menu, select **Set Properties**.

9. Select the **Disable result set cache** check box.

10. Click **OK** to apply the change.

> **Note:** Wait for a minute for the applied changes to take effect.

11. In the **QueryService** menu, select **Restart**.

Run the `BranchProfitByYear` report once. Then, rerun the `BranchProfitByYear` report in DQA and open the report's query execution trace.

Data held in the data cache is reused to satisfy the report query and the total time taken to obtain the data from the data cache is recorded in XDataCacheRetrieval node (9.148578 milliseconds).

The data required for the report could be fully satisfied from the data cache. Therefore, the `XAggregateCache`, `XAggregateCubeRetrieval`, and `XCubeRetrieval` nodes do not record values.

Figure 14-25 shows the `BranchProfitByYear`, the result set cache is disabled.

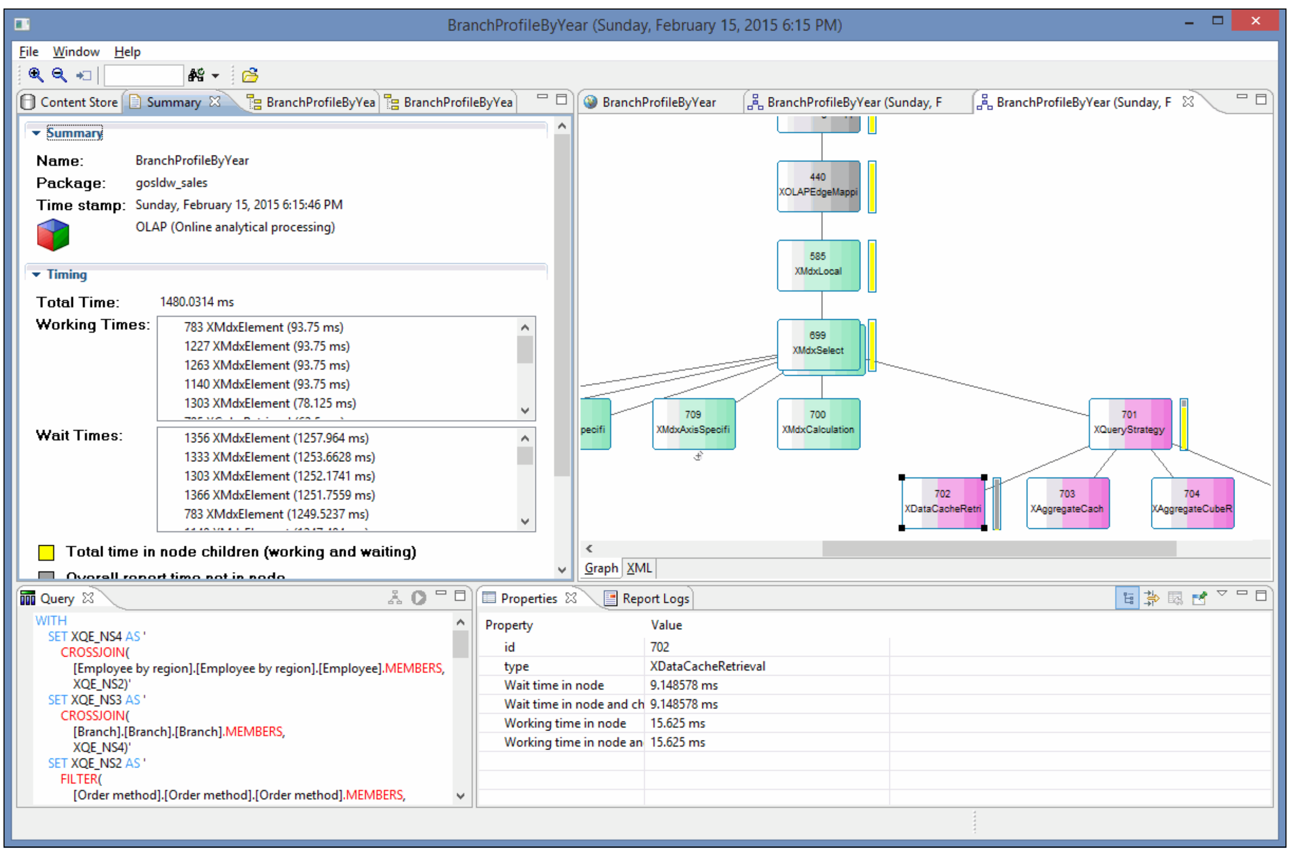


Figure 14-25 BranchProfitByYear, result set cache disabled

> **Note:** When you explore the role of caching on a report and the visualization in DQA, remember that manually clearing the dynamic cube member cache implicitly clears dependent caches, such as the data cache and aggregate cache.

14.7.4 Reviewing Report SQL

The SQL that is passed to the relational database can be viewed in DQA in these situations:

- When the data that is required to satisfy a report request is not available from one of the dynamic cube caches
- When, for optimization reasons, the query strategy determines that the request (or part of the request) should be managed by the relational database that supports the dynamic cube

The presence of working and wait time against the XQueryStrategy node indicates that data was retrieved from the relational database of the dynamic cube.

Perform the following steps to view the SQL requests that are passed to the relational database:

1. Right-click the **XMdxSelect** node to display the menu of the node. The XMdxSelect node is the parent node of XQueryStrategy.
2. Select **Open sub queries** from the menu to open all the sub query execution plans that are generated for the report execution. Each sub query opens in its own tab.

> **Notes:**
>
> - If the **Open sub queries** menu option is not available, no sub query requests have been recorded.
> - Sub query execution plans can also be opened from the Report Logs view. Sub Query execution plans are nested folders under the parent report execution plan folder. Individual sub query execution plans can be opened by selecting the sub query Profile node.

14.7.5 Verifying changes to the BranchProfitByYear report

In this scenario, the report development team extended the `BranchProfitByYear` report to help the business users understand employee sales performance and their interaction with the sales channels of the company. However, the revised report is slow.

You can use DQA to understand the performance difference and the report elements that are causing the report to be slow.

Use the following steps to review the report query execution plan:

1. Open DQA.
2. Enable query execution tracing.
3. From the Content Store view, locate the report to be traced, and from its menu select **Run Report**.
4. Open the query execution trace from the Report Logs view if the execution trace does not open automatically.

Figure 14-26 shows the query execution trace for the `BranchProfitByYearExtended` report.

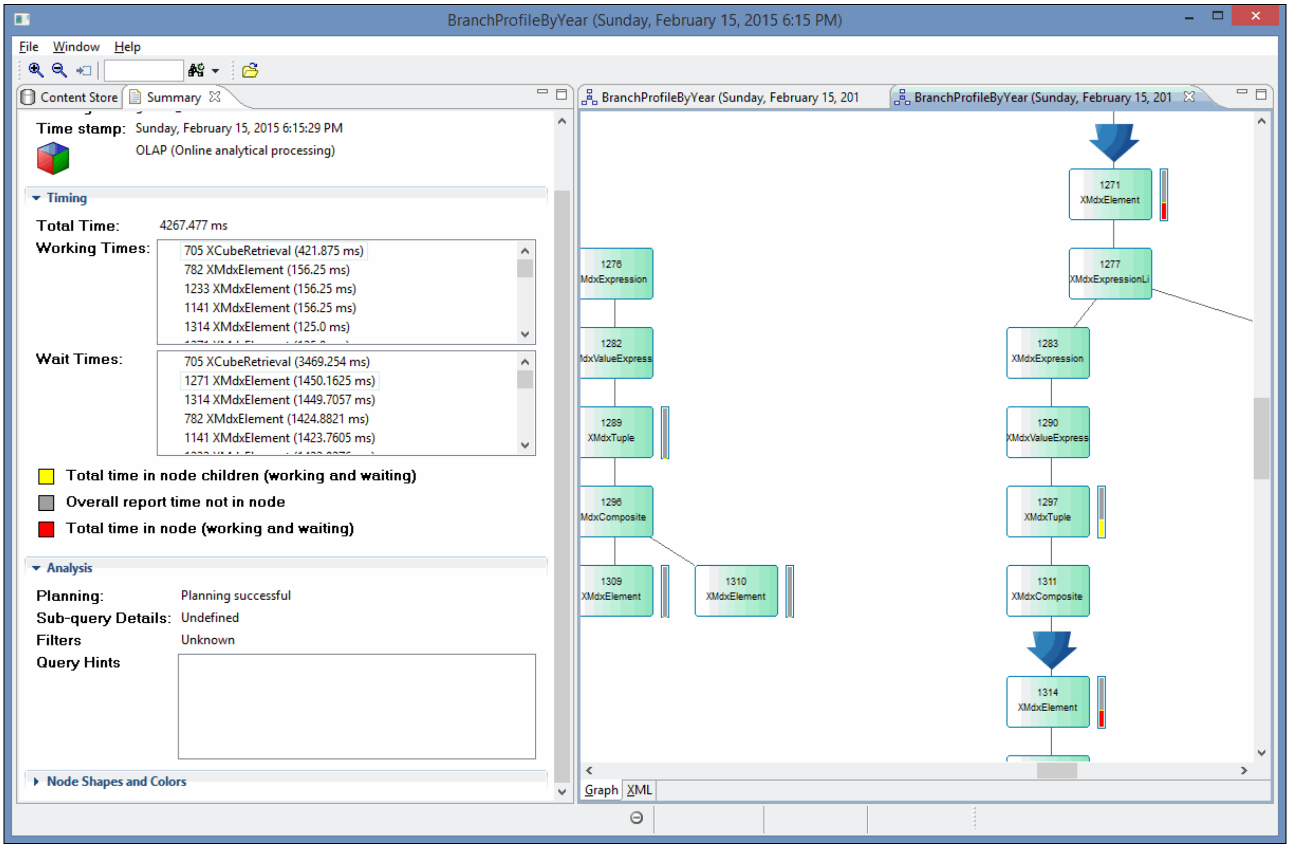


Figure 14-26 Query execution trace for report BranchProfitByYear

The report query execution trace highlights that most of the query execution time takes place within the XMdxElement nodes. Out of the 42670.477 milliseconds total time spent in query execution, 1449.7057 milliseconds wait time is spent determining appropriate values for the Gross Profit report measure.

To verify the report design, open the report in Cognos Report Studio, and explore the report expression definition for the Gross Profit Data item. The following filter is applied to the Gross Profit measure of the report:

```
[Gross Profit] >0
```

This filter was included to eliminate NULL values from the report. Although this filter expression (`[Gross Profit] >0`) is restricting the cells with NULL value, setting Suppress NULLs at the query or report level performs faster and is a better at removing empty cells.

By reviewing the MDX trace in the query window further, you can see that Order method data item is a member set that is restricted by a set filter or a data item with a filter expression as shown in Example 14-18.

Example 14-18 Order method

```
FILTER(
              [Order method].[Order method].[Order method].MEMBERS,
              [Order method].[Order method]
              .CURRENTMEMBER.PROPERTIES("MEMBER_CAPTION" )<> "E-mail" )'
```

The set filter is intended to remove the `E-mail` channel from the set of sales channels for which profitability is being assessed. However, the filter expression is applied to the caption of the current member of the Order method level. In Cognos Report Studio, the filter expression looks like the following example:

```
caption ( [gosldw_sales].[Order method].[Order method].[Order method] ) <>
'E-mail'
```

You can also use the `except` function shown in the following example to display all Order methods but `E-Mail` to restrict the report to show only order methods that do not include `E-mail`:

```
except ( [gosldw_sales].[Order method].[Order method].[Order method] , set (
[E-mail] ) )
```

However, this expression runs in a similar way to the caption function and takes around *4340 milliseconds* to run. Using a SET expression and including all the wanted Order methods results in a much faster execution time. The following example uses the SET function:

```
set([Fax],[Mail],[Sales visit],[Special],[Telephone],[Web])
```

Using the SET expression, the execution of the report takes around 1264 milliseconds to complete. Figure 14-27 shows the execution trace with modifications to use Suppress Nulls and the SET expression.

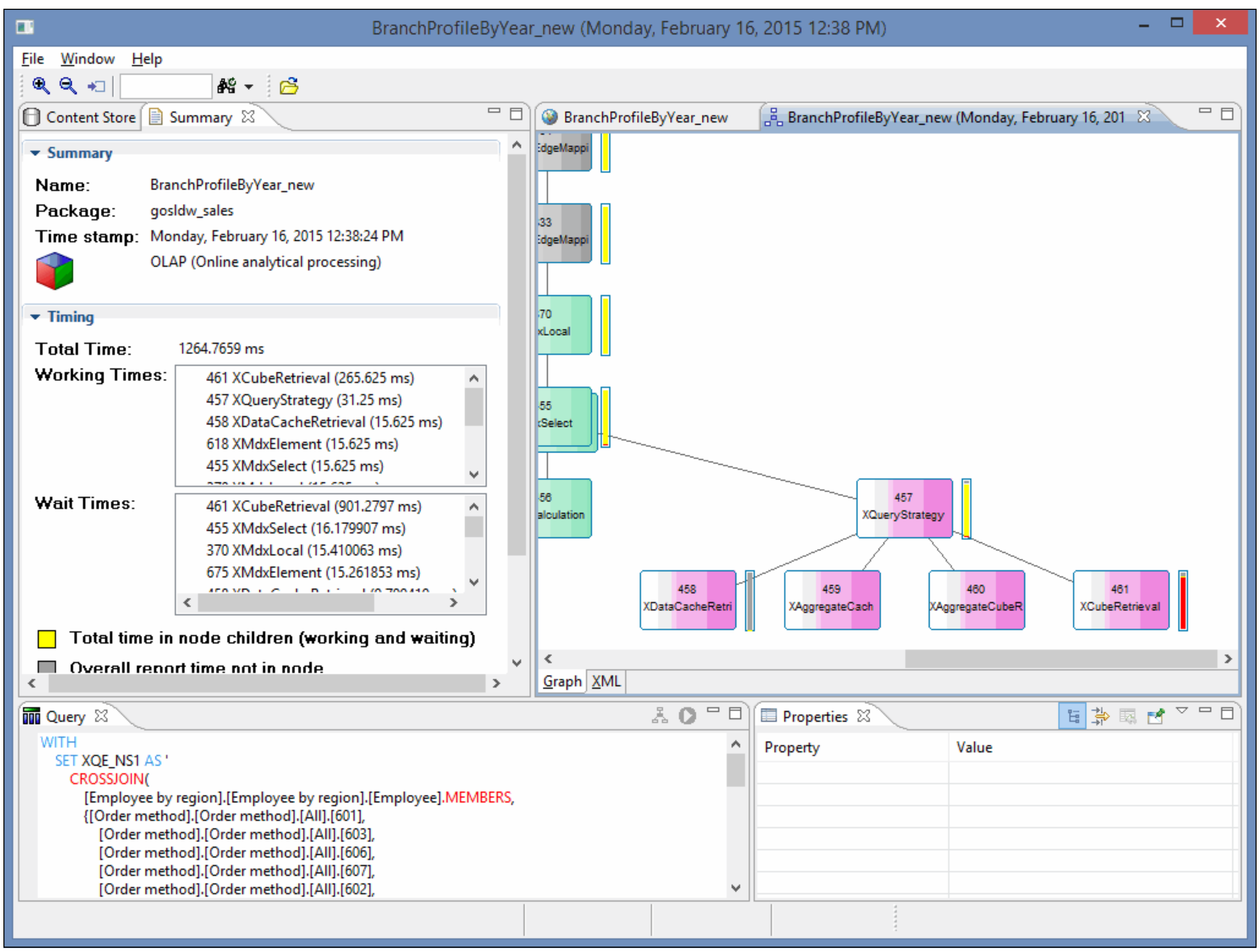


Figure 14-27 Query execution trace for report: BranchProfitByYear_new

14.7.6 Virtual cube query execution tracing

For a virtual cube, the XQueryStrategy and child nodes represent the total time spent to satisfy the MDX data request from the underlying cubes. The property values represent the cumulative totals across the dependent cubes.

Correspondingly, the virtual cube will not display any subquery nodes that represent the SQL queries that are passed to the underlying relational databases.

Query execution tracing on the dynamic cubes that comprise the virtual cube should be enabled to explore the query execution behavior for requests that are satisfied for the virtual cube.

14.8 Open query service log with DQA

Perform the following steps to view the query service log for a report in DQA:

1. Ensure that dynamic query logging is enabled.
2. Run a report from the Content Store view.
3. Open the query execution trace of the report from the Report Logs view, if the trace is not opened automatically.
4. Select **Window** → **Show View** to open the Show View window.
5. In the Show View window, select **Navigation** → **DQM Server Logs** to view the Dynamic Query log.

Figure 14-28 shows the dynamic Query log for `Report 2. Sales By Region`.

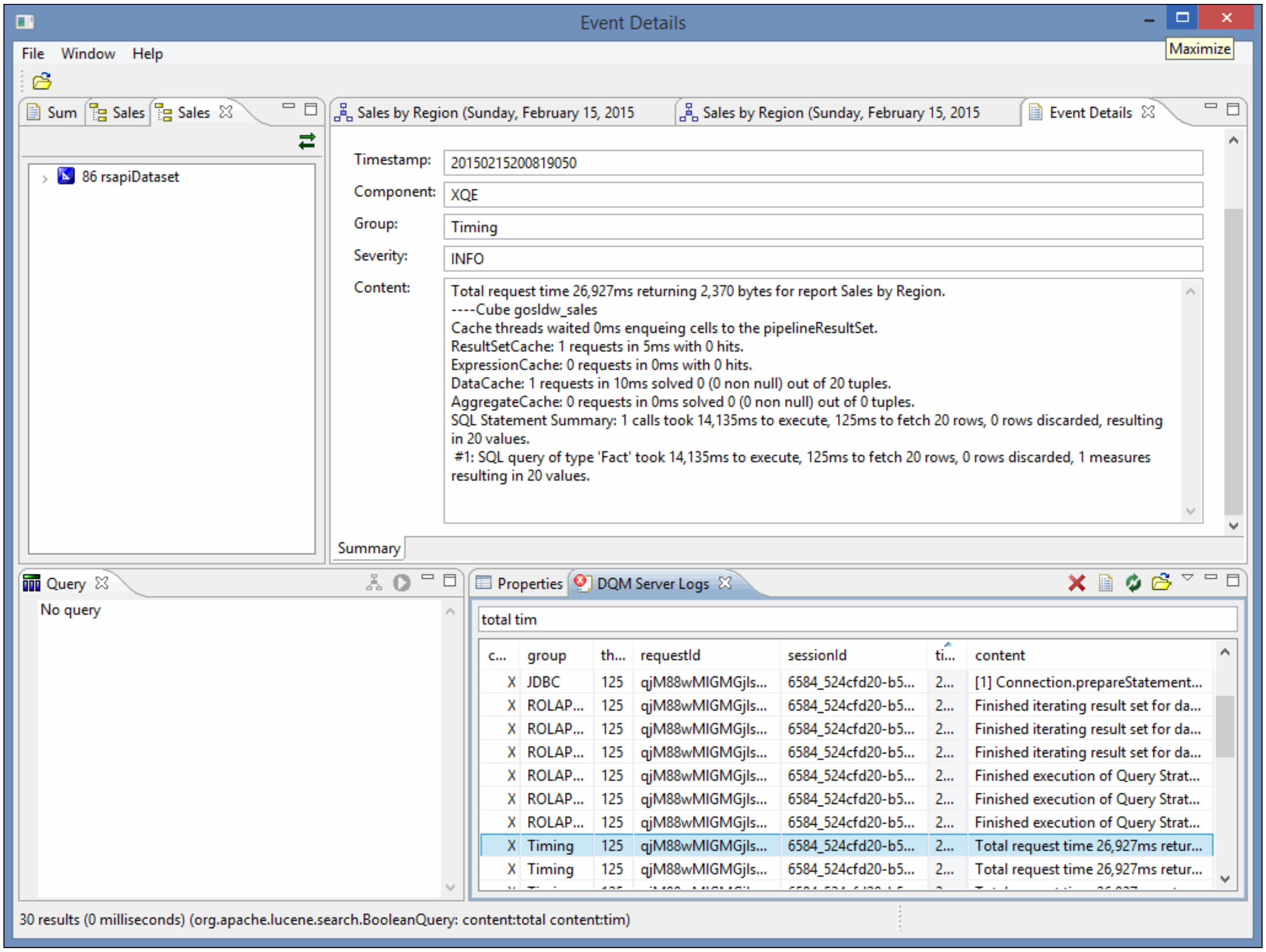


Figure 14-28 Dynamic Query log for Report 2. Sales By Region

14.9 IBM Support Assistant

The IBM Support Assistant (ISA) provides Cognos system administrators with rich problem determination tools and functions to help quickly troubleshoot problems related to the query service JVM. ISA also provides easy case management, file management, problem determination tooling capabilities, and automated analysis capabilities.

Additional Information about ISA is available at:

http://www-01.ibm.com/support/knowledgecenter/SSLLVC_5.0.0/com.ibm.isa.help.doc/html/overview/intro.html

You can download the latest version of ISA from this link:

http://www-01.ibm.com/software/support/isa/teamserver.html

14.9.1 Accessing ISA problem determination tools

Figure 14-29 shows the ISA user interface. You can start problem determination from the menu in the Files tab.

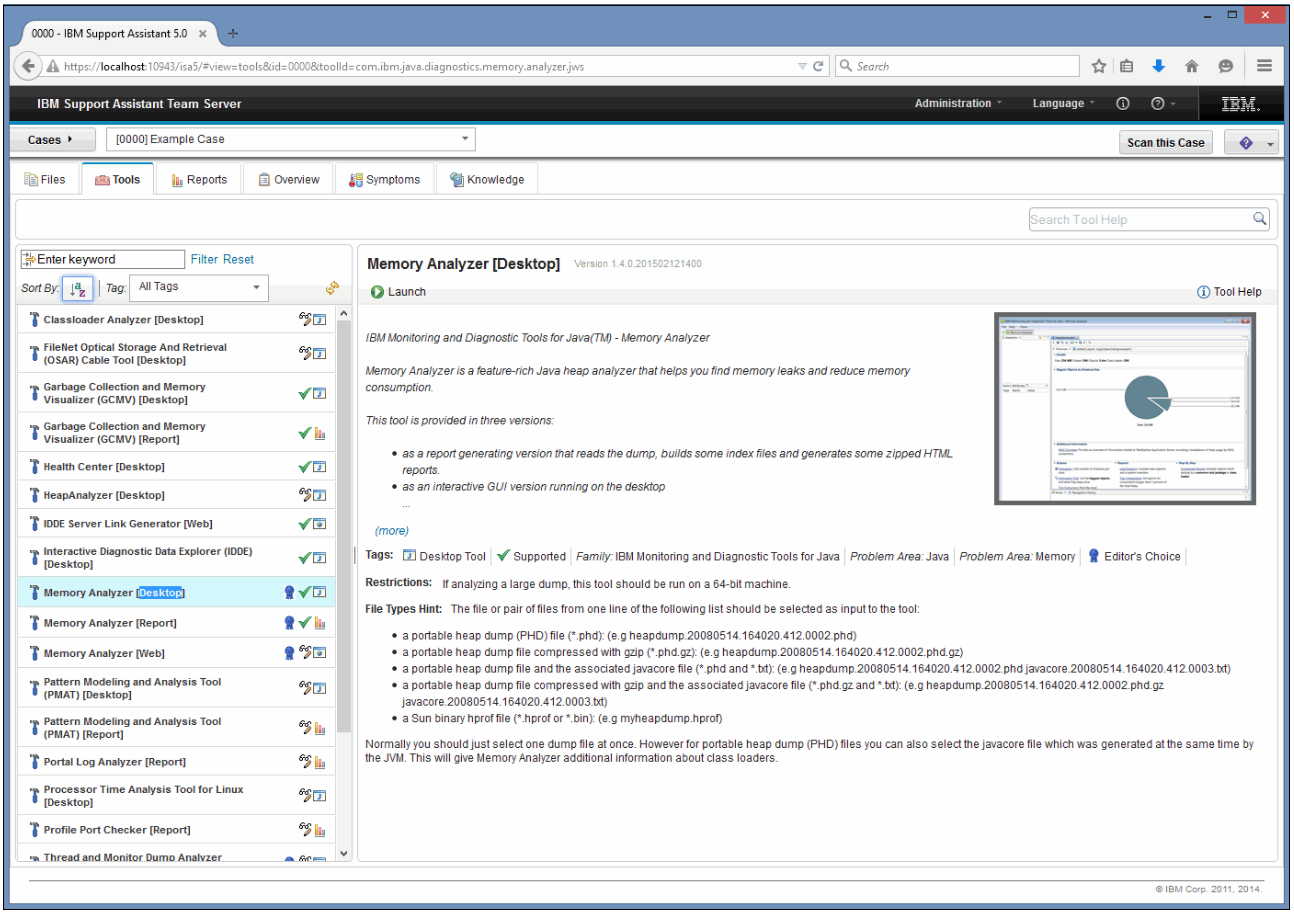


Figure 14-29 ISA Tools tab

There are several tools available in ISA. New tools and updates are released periodically. The following sections describe how to use some of these tools to troubleshoot problems with the query service JVM.

The complete list of available tools is available at.

http://www-01.ibm.com/support/docview.wss?uid=swg27036217

Garbage Collection and Memory Visualizer

The desktop version of the Garbage Collection and Memory Visualizer (GCMV) provides analysis and views of the verbose garbage collection output of your application. GCMV provides a clear summary and interprets the information to produce a series of tuning recommendations. GCMV enables you to analyze and visualize verbose GC logs generated by the query service JVM. These files are in the `logs/XQE` folder and are named `dq_verbosegc_*.log.001`.

To open a verbose GC file, click **File** → **Load File**.

The GCMV allows you to perform these tasks:

- Monitor and fine-tune Java heap size and garbage collection performance
- Identify possible memory leaks
- Size the Java heap correctly
- Select the best garbage collection policy

Figure 14-30 shows the GCMV recommendations based on the query service JVM verbose garbage collection (GC) file.

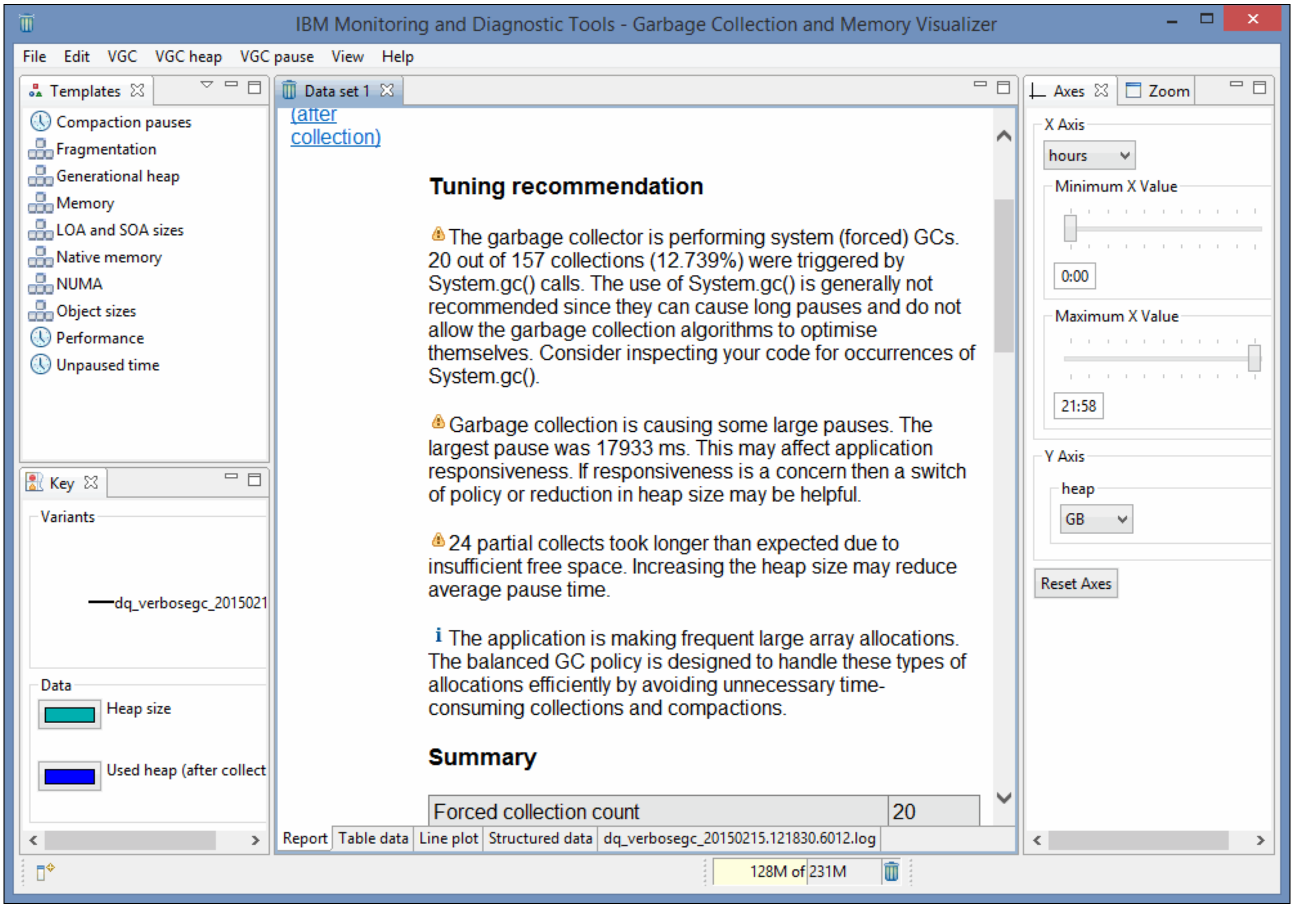


Figure 14-30 GCMV tuning recommendations

GCMV parses and plots data from the verbose GC logs, as shown in GCMV parses and plots data from the verbose GC logs in Figure 14-31.

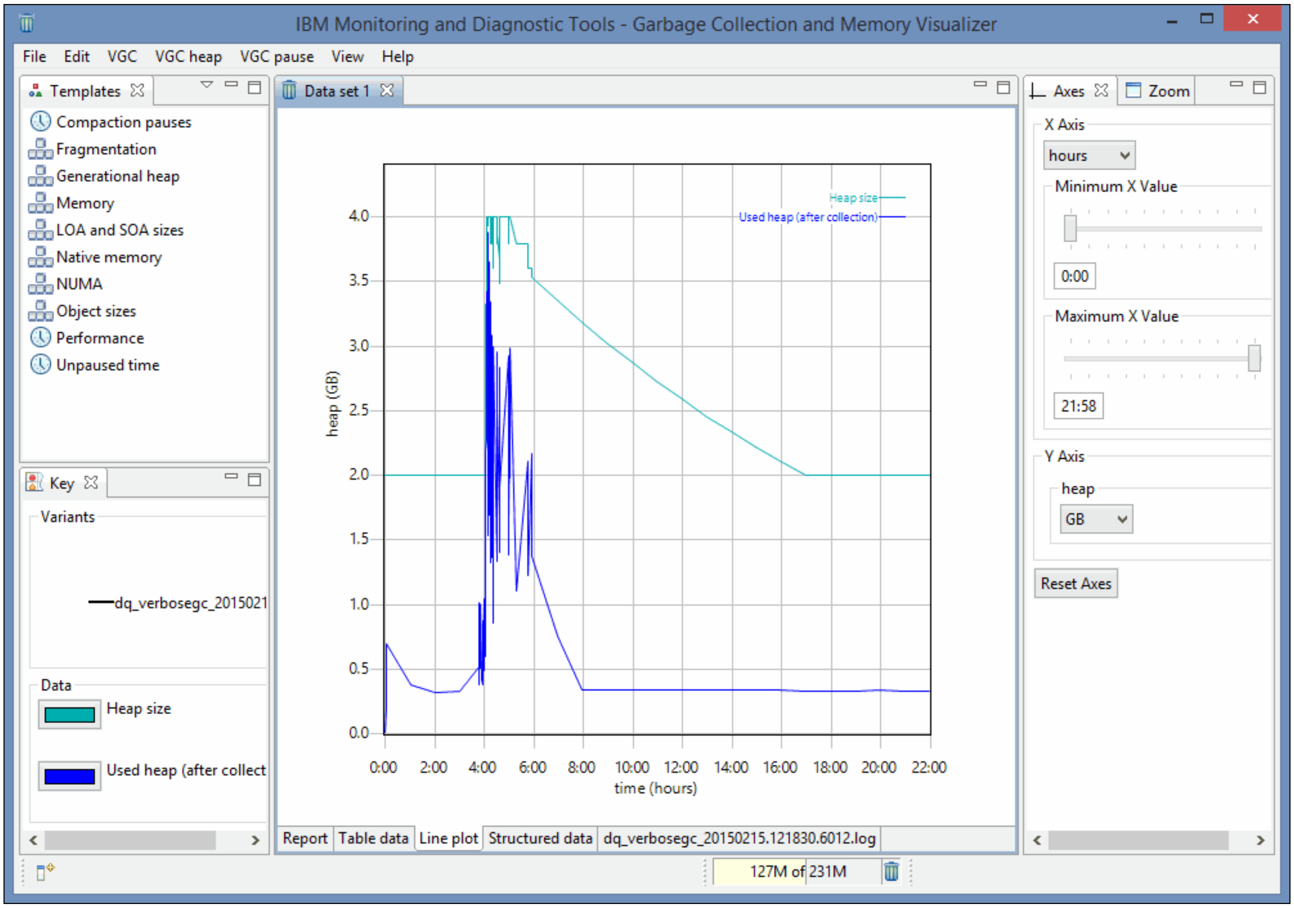


Figure 14-31 GCMV heap size

Memory Analyzer

The desktop version of Memory Analyzer enables you to analyze heap memory dumps with hundreds of millions of objects, quickly calculate the size of objects, see what is preventing the Garbage Collector from collecting objects, run a report to automatically extract leak suspects, and more. It is based on the Eclipse Memory Analyzer (MAT) project.

You can use this tool to analyze the `heapdump*.phd` files that are generated by the query service JVM as shown in Figure 14-32. These files are in the `bin64` folder. To open a verbose GC file, click **File** → **Open Heap Dump**.

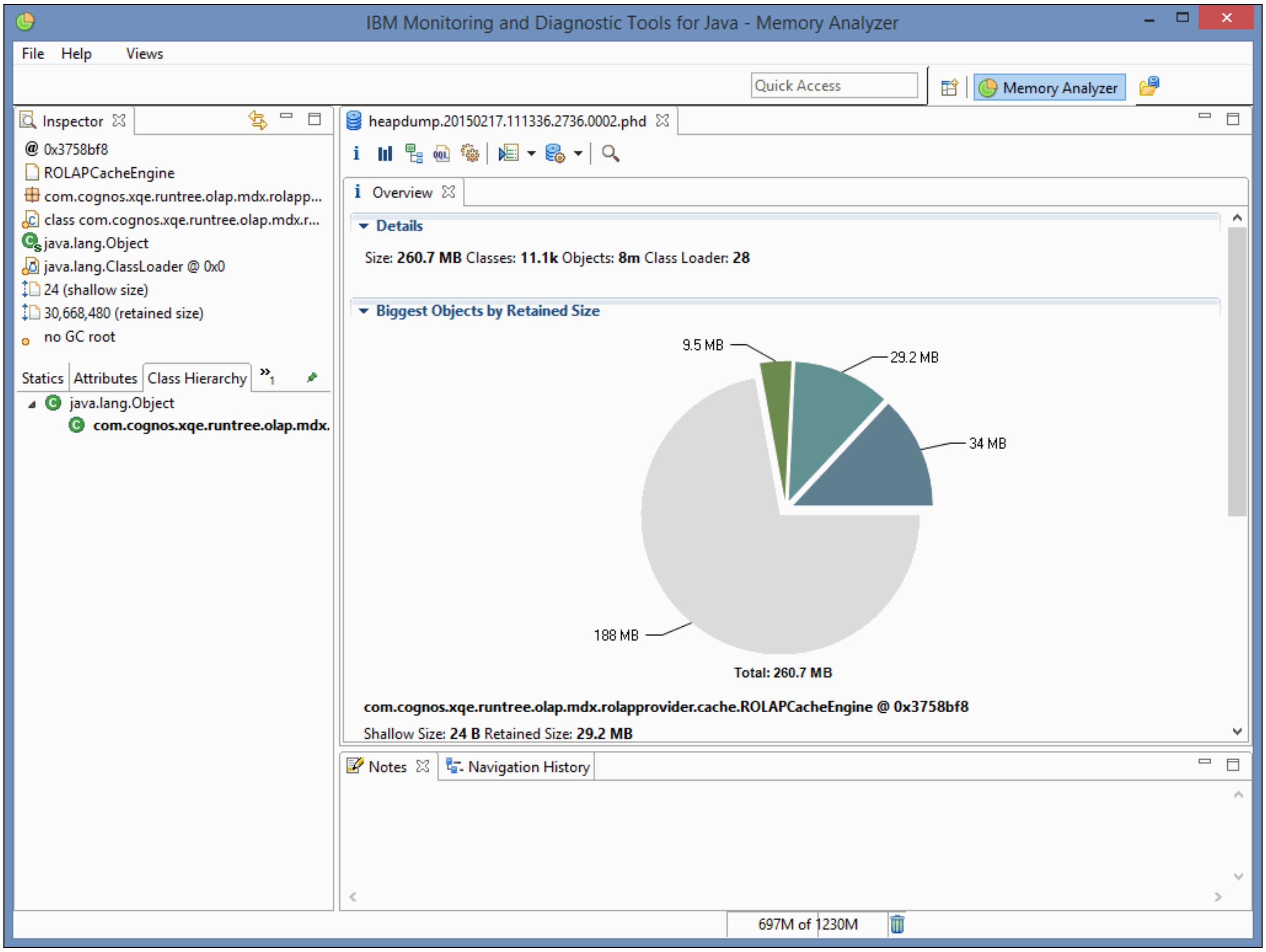


Figure 14-32 Memory Analyzer

15

Query service memory monitoring

This chapter discusses the memory monitoring mechanism introduced in Cognos V10.2.2 to improve stability of the query service.

This chapter contains the following sections:

15.1 Overview of query service memory

The query service is a Java-based process where objects are allocated on the Java virtual machine (JVM) heap. To ensure optimal performance and stability, the user must correctly compute and configure the size of the JVM heap. For more information, see Chapter 17, “Hardware sizing” on page 533 and 2.6.1, “Memory usage” on page 60.

Underestimating the size of the JVM Heap might result in poor performance due to prolonged garbage collection (GC) cycles when all application threads are halted while the JVM attempts to place orphaned objects back into the available heap space. The memory monitoring and query cancellation mechanism was designed to help avoid instances in which the heap space within the query service JVM is depleted. If the situation cannot be avoided, the query service is restarted to prevent problems caused by inadequate heap space.

15.2 Query service memory considerations

The JVM obtains memory from the underlying operating system as required. The query service JVM obtains an initial amount of memory for the heap as specified by the `Initial JVM heap size` for the query service configuration setting, up to the amount specified by the `JVM heap size limit` for the query service configuration setting. An attempt by the process to create objects that exceed the available heap space might cause the JVM to enter long garbage collection cycles, and eventually cause the JVM to raise an out of memory exception. As described in the *IBM Cognos Dynamic Cubes User Guide*, this exception causes the query service to restart.

For the query service, one approach is to set `Initial JVM heap size` equal to `JVM heap size limit` to avoid heap fragmentation for more efficient memory management. After the query service JVM reserves a slice of memory specified by Initial JVM heap size, it does not release this memory back to the operating system during the query service process lifespan.

The usage of the query service JVM heap can be monitored in the IBM Cognos Administration Console under **Metrics** → **QueryService** as shown in Figure 15-1.

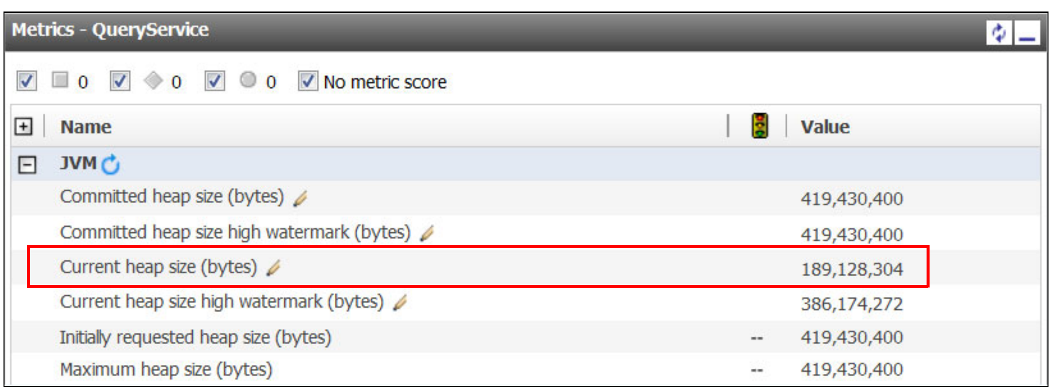


Figure 15-1 Metrics - Query Service

If you notice a degradation in an application performance, you should closely monitor the current heap size. If the current heap size approaches the maximum heap size and it is not substantially reduced after the global GC cycle, then your JVM heap memory allocation might not be sufficient for the current workload. You might need to increase it.

The query service is configured to use Generational Concurrent Garbage Collector (Generational) by default. A generational garbage collection strategy divides heap into different areas, and collects these areas at different rates. New objects are allocated in the *nursery* area to be collected more frequently. After objects survived for a while, these objects are moved into a *tenure* area. The collector examines these objects much less frequently, as these objects are less likely to become garbage.

Tip: You should not change default garbage collection policy unless the query service experiences prolonged garbage collection cycles. Changing it might cause other side effects such as degradation in overall performance or throughput depending on the workload characteristics.

Another option available is Balanced Garbage Collection policy. The Balanced Garbage Collection policy uses a region-based layout for the Java heap. These regions are individually managed to reduce the maximum pause time on large heaps. Because each region can be collected independently, it allows the collector to focus only on the regions that offer the highest yield. The goal is to reduce the long pauses by incrementally reducing fragmentation in the heap by only compacting a part of the heap in every collection. The query service might benefit from using the Balanced Garbage Collection Policy when the query service experiences long global collection pause times frequently enough to be disruptive and the query service JVM heap is relatively large (greater than 32 GB).

Configuring the query service JVM settings

To configure the query service JVM settings, complete the following steps:

1. In IBM Cognos Administration, select the Status tab and then select **Dynamic Cubes**.
2. In the Scorecard section, select the **All servers groups** view.
3. Click the **Server group** under System.
4. From the **Actions** menu for the `Query Service - dispatcher_name`, select **Set properties**.
5. Select the Settings tab.
6. For **JVM garbage collection policy (Requires QueryService restart)**, select **Balanced** as shown in Figure 15-2.

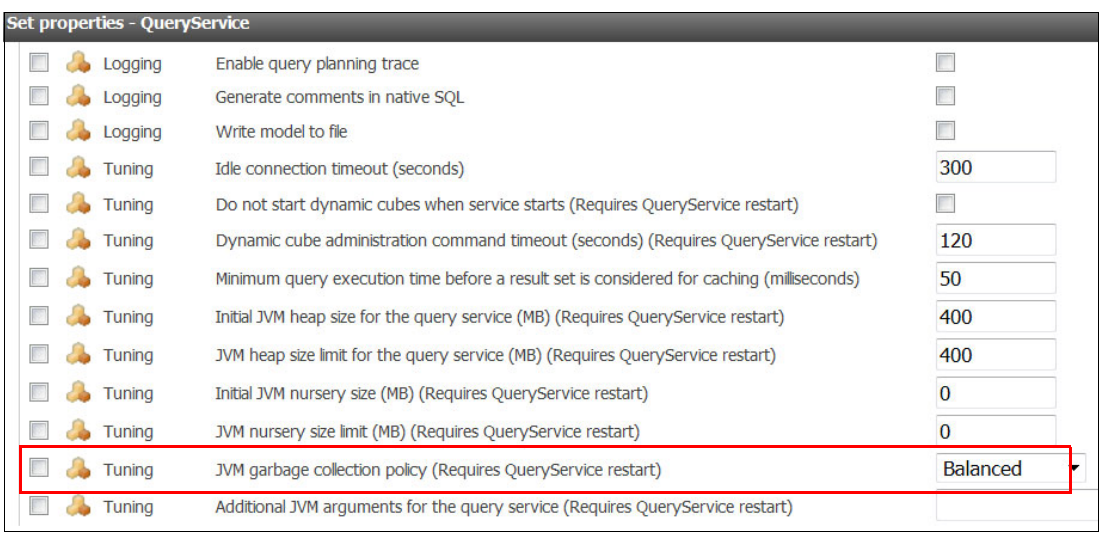


Figure 15-2 Configuring balanced garbage collection policy

7. Restart the query service.

15.3 Memory monitoring in query service

The query service continuously monitors its JVM heap to avoid running out of memory. When the available memory decreases to 10% or less, it enters an *overloaded* state and performs the following actions:

- Redirect any subsequent incoming queries to another dispatcher configured in the same server group.
- Prevent any subsequent incoming queries from starting if no other dispatchers are configured in the same server group.
- Cancel selected queries until available memory is 10% or more of the JVM heap and the query service reenters a *normal* state

15.3.1 Criteria used to cancel queries

Queries are selected for cancellation by analyzing the running time and size of each query in progress, and ranking them according to their effect on the server. The effect is determined by the following factors (listed in order of importance):

- The largest set created during the request.
- The number of data points added to the data cache.
- The request running time.

Example 15-1 is a log entry that shows request metrics used to select a query candidate to be canceled, in this example the `Heavy_Query _Demo` report.

Example 15-1 Request metrics to select a query candidate for cancellation

```
<event component="XQE" group="Resources.Monitor..." [CDATA[Successfully issued
cancel command for request...
Report name: Heavy_Query_Demo
Request metrics used:
   Biggest set size: 39221228168403501
   Number of values added to cubelets: 0
   Request duration (ms): 392983
]]></event>
```

For Cognos relational reports, the only criteria used to select a candidate for cancellationis the request running time.

15.3.2 Algorithm used to cancel queries

After ranking the queries based on the criteria outlined in 15.3.1, “Criteria used to cancel queries” on page 512, the dynamic query mode server cancels queries in the following order:

1. It cancels *the query with the highest rank* and waits 10 seconds.
2. If step 1 does not resolve the low memory issue (available memory is less than 10%), it *cancels the next 30% of the highest ranked queries* and waits 10 seconds.
3. If step 2 does not resolve the low memory issue (available memory is less than 10%), it *cancels all remaining queries*, waits 120 seconds, and makes an explicit GC call.

The query service stops canceling queries when it reenters *normal* state. For example, if after requesting to cancel 30% of the highest ranked queries in step 2 it reenters a normal state before step 3, it tries to stop canceling any queries still active.

Example 15-2 is a log entry that shows an example of canceling queries when entering an overloaded state:

1. Initially eight requests are available.
2. Cancels the heaviest request (1 out of 8 available requests).
3. Waits for 10 seconds.
4. Cancels the next 30% of the requests (2 out of 7 remaining requests).
5. Waits for 10 seconds.
6. Cancels all remaining requests (5 out of 5 remaining requests).

Example 15-2 Canceling queries when entering overload state

```
<event...[Entering OVERLOADED state (heap size 150,000 MB, used 135,185 MB (90%),
free 14,814 MB (9%))]
[Canceling requests due to OVERLOADED state (8 requests available)]
23:04:00.109 [Canceling the heaviest request]
[Successfully issued cancel command for request... due to low memory condition.]
...
23:04:10.112 [Canceling top 30% of remaining requests (2 of 7)]
...
23:04:20.115 [Canceling 5 remaining requests]]
</event>
```

If these actions do not resolve the available memory issue, the query service stops and restarts. The cubes start is delayed by 5 minutes by default to allow any orphaned database queries to be canceled on the database server.

For more information about memory monitoring feature, see section “Memory monitoring in the dynamic query mode server” in the *IBM Cognos Dynamic Cubes Version 10.2.2 User Guide*.

15.3.3 Analyzing Logs

Errors that are related to low available memory are saved to a log file under the `Resources.Monitor` group:

```
<component =”XQE” group =”Resources.Monitor” level=”ERROR”/>
```

You can view and analyze errors in the `Resources.Monitor` category of the `c10_location/logs/XQE` log file.

To log greater level of detail, edit the file `c10_location/configuration/xqe.diagnosticlogging.xml` and set logging to the `info` or `trace` level.

Example 15-3 shows how to set `Resources.Monitor` logging to `level="info"`.

Example 15-3 Setting Resources.Monitor logging to level="info"

```
<
<XQE>
   <components>
      <component name="XQE">
         <eventGroup name="Resources.Monitor" level ="info"/>
         ....
         ....
      </component>
   </components>
</XQE>
```

In Example 15-4, the log entry under `Resources.Monitor` shows entering an overloaded state due to a low memory condition. Requests are being canceled (two requests are available for canceling). The heaviest request (report name: `AggRec_Xtab_RO_RS`) is canceled based on metrics provided. After canceling the first request, the log shows reentering normal state and not canceling the remaining requests.

Example 15-4 Canceling requests after entering overload state and reentering normal state

```
<event component="XQE" group="Resources.Monitor" level="ERROR" ...[CDATA[Entering
OVERLOADED state (heap size 588 MB, used 545 MB (92%), free 42 MB (7%))]]></event>
<event component="XQE" group="Resources.Monitor" level="ERROR"..<![CDATA[Canceling
requests due to OVERLOADED state (2 requests available)]]></event>
<event component="XQE" group="Resources.Monitor" level="ERROR"..<![CDATA[Canceling
the heaviest request]]></event>
<event component="XQE" group="Resources.Monitor" level="ERROR" thread="42"
timestamp="2015-04-23 16:02:52.221"><![CDATA[Successfully issued cancel command
for request...
Report name: AggRec_Xtab_RO_RS
Request metrics used:
   Biggest set size: 51650610332
   Number of values added to cubelets: 133458
   Request duration (ms): 17118
<event component="XQE" group="Resources.Monitor" level="ERROR" thread="41"
timestamp="2015-04-23 16:02:53.019"><![CDATA[Entering NORMAL state (heap size 588
MB, used 511 MB (87%), free 76 MB (12%))]]></event>
<event component="XQE" group="Resources.Monitor" level="ERROR" thread="42"
timestamp="2015-04-23 16:02:53.221"><![CDATA[Finished canceling requests due to
OVERLOADED state (NORMAL state reached)]]>
```

15.3.4 Memory monitoring enhancements in Cognos BI V10.2.2 Fix Pack 1

In Cognos BI V10.2.2 Fix Pack 1, the following enhancements to the memory management capabilities of the query service were introduced:

- Added support for Oracle JVM.
- Added support for Balanced Garbage Collection policy.

- Queries to load hierarchies can be canceled due to insufficient memory.
- Queries can be routed to other servers within a server group for execution.

If a cube is refreshing its member cache or if it is restarting, notably on a system with another cube that is actively processing queries, loading members might push the query service beyond its available memory. Starting with Cognos BI V10.2.2. Fix Pack 1, if this situation occurs, the query service cancels the queries to protect the availability of the cubes that are already active.

The cancellation of the member loading queries in turn causes the cube start or refresh to fail. Therefore, a subsequent refresh or start of the cube is required, ideally when more memory is available.

Before Cognos BI V10.2.2 Fix Pack 1, if a cube was available on multiple servers as part of a server group and a query running on one server was canceled due to insufficient memory, the overall report was canceled.

Starting with Cognos BI V10.2.2 Fix Pack 1, in the presence of a query being canceled on one server due to insufficient memory, the original report or analysis is now routed to another server within the server group. This process continues until either the report (query) runs successfully or all servers cancel the query due to insufficient memory. At this point, an error is returned to the user.

15.3.5 Memory monitoring limitations

The following memory monitoring limitations still exist in Cognos BI V10.2.2 Fix Pack 1:

- The member in-memory aggregate and data caches are not cleared on low memory.
- Queries posed by the query service itself, such as those used to load in-memory aggregates and apply near real-time incremental updates, are not canceled in low memory situations.

15.4 Configuring the query service monitoring settings

You can change the default behavior of the memory monitoring feature (the Resource Monitor) within the query service when the server becomes overloaded.

To configure the Resource Monitor, add the Java command-line argument **`-D`** to the **`Additional JVM arguments`** for the query service property in the query service settings, then append the appropriate Resource Monitor setting.

For example, to change the maximum JVM heap in use that is considered normal to `95` (default is `90`), add the following string: **`-DresourceMonitor.overloadedPercent=95`**. This setting can be changed to a higher value as shown in Figure 15-3 on page 516 when working with a very large JVM heap.

To configure the query service memory monitoring settings, perform the following steps:

1. In IBM Cognos Administration, select the Status tab, and then select **Dynamic Cubes**.
2. In the Scorecard pane, select the **All servers groups** view.
3. Click the **Server group** under System.
4. From the **Actions** menu for the `QueryService - dispatcher_name`, click **Set properties**.
5. Select the Settings tab.

6. In **Additional JVM arguments for the query service**, add following string **`-DresourceMonitor.overloadedPercent=95`** (see Figure 15-3).

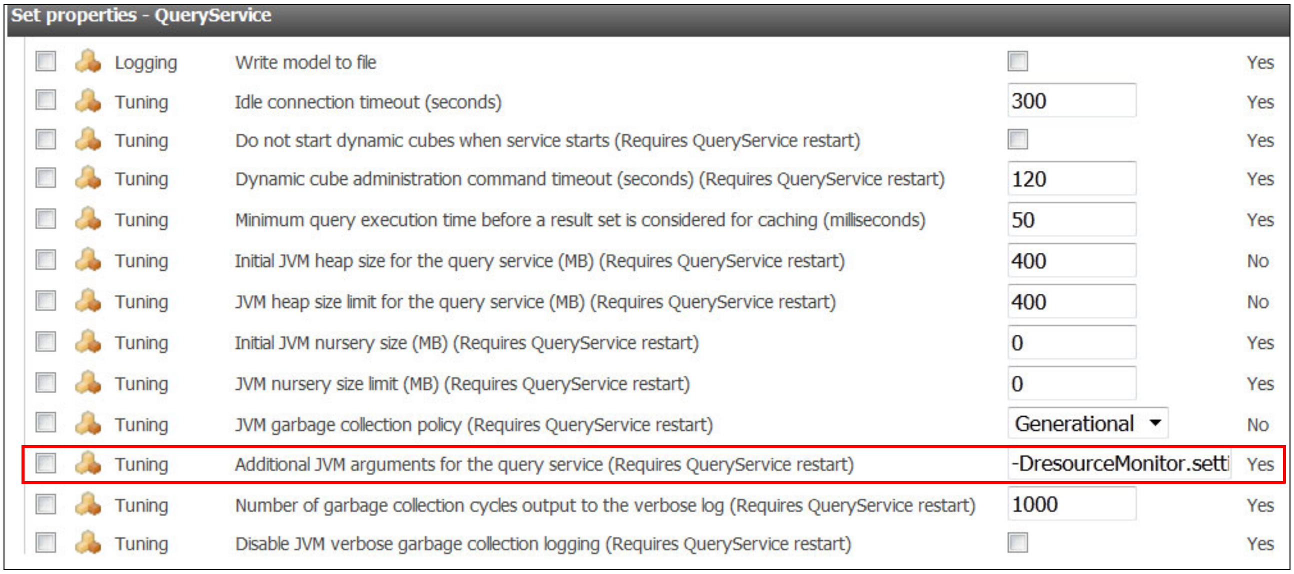


Figure 15-3 Configure Additional JVM arguments

7. Restart the query service.

You can also configure the Resource Monitor by copying the file `c10_location/configuration/ xqe.config.xml`, and renaming it to `xqe.config.custom.xml` for editing. You can then add the appropriate Resource Monitor setting to `xqe.config.custom.xml`. For example, to change the maximum JVM heap size in use that is considered normal to `95`, add the following section:

```
<resourceMonitor> <overloadedPercent>95</overloadedPercent> </resourceMonitor>
```

> **Tip:** If the `Additional JVM argument for the query service` differs from the corresponding setting in the `xqe.config.custom.xml` file, the Additional JVM argument takes precedent

See the *Dynamic Cubes User Guide* for more configuration settings available for the Resource Monitor.

More settings can be configured in the Advanced settings property in the query service. For example, to change the number of seconds to delay starting the dynamic cubes after a critical failure to 100 seconds, add the string **`qsCubeStartDelayOnRecovery`** to the Advanced settings property. You can change this setting to a lower value from the default 300 seconds as shown in Figure 15-4 on page 517 and Figure 15-5 on page 517 if you want to minimize the cube start delay after a query service critical failure.

Configuring advanced query service settings

To configure advanced query service settings, complete the following steps:

1. In IBM Cognos Administration, select the Status tab and then select **Dynamic Cubes**.
2. In the Scorecard section, select the **All servers groups** view.
3. Select the **Server group** under System.
4. From the **Actions** menu for the `QueryService - dispatcher_name`, click **Set properties**.

5. Select the **Settings** tab.
6. Click **Edit** for the **Advanced settings** property (see Figure 15-4).

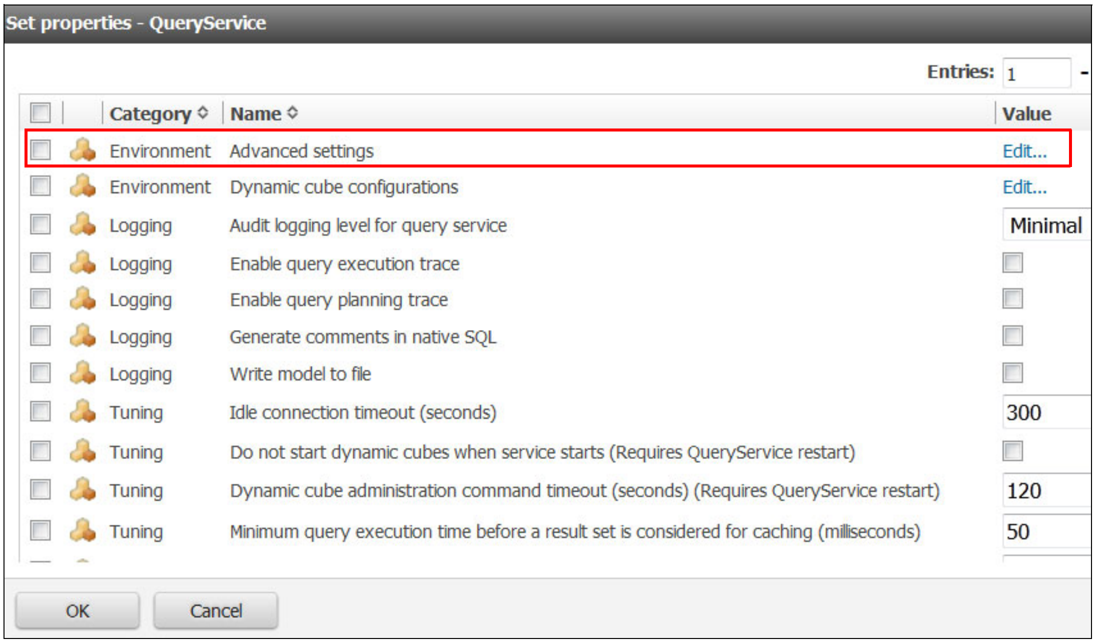


Figure 15-4 Set Advanced settings for the query service

7. Select **Overwrite the settings acquired from the parent entry**, set the parameter to `qsCubeStartDelayOnRecovery`, and set value to `100` (see Figure 15-5).

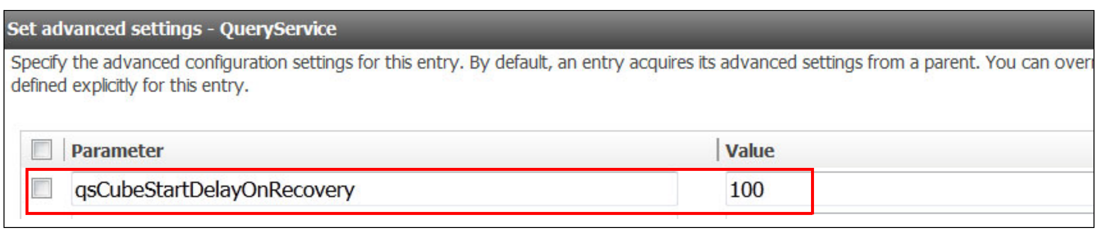


Figure 15-5 Setting advanced settings for the query service - Adding a parameter

8. Restart the query service

See the *Dynamic Cubes User Guide* for more related advanced settings that are available.

16

Using Framework Manager models

This chapter provides an overview of the IBM Cognos Cube Designer feature that allows you to import metadata from a Framework Manager package.

This chapter contains the following sections:

- Introduction
- Cube Designer and Framework Manager paradigms compared
- Process to import Framework Manager models

16.1 Introduction

The ability to import an IBM Cognos Framework Manager model into IBM Cognos Cube Designer allows you to take advantage of your existing IBM Cognos BI modeling investments when you are modeling dynamic cubes. Additional modeling is typically required after a Framework Manager package is imported into Cognos Cube Designer before a functional dynamic cube is created.

This feature is intended for cases where a relational or dimensionally modeled relational data (DMR)-based solution has not provided the required performance or functionality, but the wanted performance or functionality is available with Cognos Dynamic Cubes.

The feature allows a modeler to use their existing models, especially in the case of DMR models that contain dimensional information that is directly translated into equivalent dynamic cubes structures.

It is important to understand that this feature does not alter the requirements for Cognos Dynamic Cubes. It is still a requirement that a dynamic cube be modeled on a relational star or snowflake schema. The parts of the imported Framework Manager model that you model as dynamic cubes should be based on either a star or snowflake schema. No amount of editing within Cognos Cube Designer can overcome this restriction.

The import process creates Cognos Cube Designer metadata objects that are analogous to the Framework Manager objects that you choose to import. For more information, see 16.3.4, “Generation of the dynamic cube metadata” on page 526.

During the process of importing a Framework Manager model, Cognos Cube Designer attempts to unwind the model objects and their relationships. It attempts to replicate the source Framework Manager model in Cognos Cube Designer in a manner that conforms to the IBM Cognos preferred practices. The business and data layers of the model are created internally. Your layers are refactored based on the preferred practices.

After the Framework Manager metadata is imported into Cognos Cube Designer, you need to complete these tasks:

- Perform additional modeling
- Correct issues that are identified by Cognos Cube Designer

For more information, see 16.3.5, “Model cleanup and refinement” on page 529.

The ability to import a Framework Manager model into Cognos Cube Designer does not support the conversion of existing reports to allow them to run on the new model. Any such reports must be authored again because all model references such as dimensions, hierarchies, levels, and members will be different from those in the source Framework Manager DMR model. The model references will be entirely different from all metadata such as query subjects and query items in a source Framework Manager relational model.

16.2 Cube Designer and Framework Manager paradigms compared

Much of the fundamental modeling concepts in Cube Designer and Framework Manager are similar, but the underlying technologies and approaches to modeling are different. These differences mean that not all of a Framework Manager model can be mapped to an analogous object in a dynamic cube.

In Framework Manager, the recommended practice is to create several layers in your model:

- Query layer

  This is the layer where you define the query plan in which you can do the following tasks, among others:

  – Create aliases for objects that are used in multiple contexts.

  – Define unambiguous relationship paths so that the query engine can generate the appropriate query to match the role that the object is playing

- Business layer

  This is the layer where you add the business logic to the query layer objects.

- Dimensional layer

  Add this layer if you are working with a DMR model.

Cube Designer collapses the modeling experience into a cleaner, more compact approach. When you define a dimension, you are also defining the relationships between the underlying tables used in the dimension. Similarly, when you define a relationship between a cube measure dimension and a dimension, you are defining the query layer relationship between the underlying fact table and the tables of the dimension. The model that you define resolves to a query layer plan that conforms to preferred practice, which then generates queries in a predictable manner.

When you model multi-fact scenarios in Framework Manager, you create multiple measure dimensions and include them in a package that is different from Cognos Cube Designer.

Each dynamic cube consists of a single fact table and its related dimensions. If you want to work with multiple fact tables, you must model a dynamic cube for each fact table and use them in virtual cubes that allow you to create reports that use those fact tables. For more information, see 6.5, “Multiple fact and multiple fact grain scenarios” on page 181 and Chapter 8, “Virtual cubes” on page 223.

16.3 Process to import Framework Manager models

This section describes the stages that are involved while importing Framework Manager models.

16.3.1 Process overview

The process to import Framework Manager models begins with a published package in Cognos Connection. The import process extracts the model from the package and brings it into Cube Designer. You then choose what objects you want to transform into Cube Designer objects.

The process has several stages:

1. Preparation
2. Retrieving the Framework Manager model metadata
3. Generation of the dynamic cube objects
4. Model cleanup and refinement

16.3.2 Preparation

The preparation stage precedes the actual importing of the Framework Manager model.

The most critical aspect of the preparation state entails identifying whether your Framework Manager models are appropriate candidates for migration to Cognos Dynamic Cubes.

In addition, it requires you to educate the relevant employees about the Cognos Dynamic Cubes technology so that they can succeed. Before proceeding, review the IBM Cognos modeling guidelines to help you understand what is going to happen. It is also useful to review how Cognos Cube Designer works and how it differs from modeling in Framework Manager.

If your existing DMR or relational application is working satisfactory, it might not make sense to attempt migration unless there are features in Cognos Dynamic Cubes that you want to use or you want to improve performance.

It is crucial to determine whether the structure of the database underlying the Framework Manager model is one that allows Cognos Dynamic Cubes to function well. Dynamic cubes require a data warehouse star schema or snowflake schema.

In addition, each measure dimension in the Framework Manager model must be from a single fact table of a single grain. If a Framework Manager measure dimension has measures from more than one source fact table, Cognos Cube Designer will identify the issue during validation. You must edit the measure dimension of the cube so that only one fact table is used. You must create cubes for each of the other fact tables that you want to use. You can reuse dimensions in multiple cubes.

16.3.3 Retrieving the Framework Manager model metadata

You retrieve Framework Manager metadata from packages published in Cognos Connection.

To connect to Cognos Connection, from IBM Cognos Cube Designer click **Get Metadata Select Framework Manager Package** as shown in Figure 16-1.

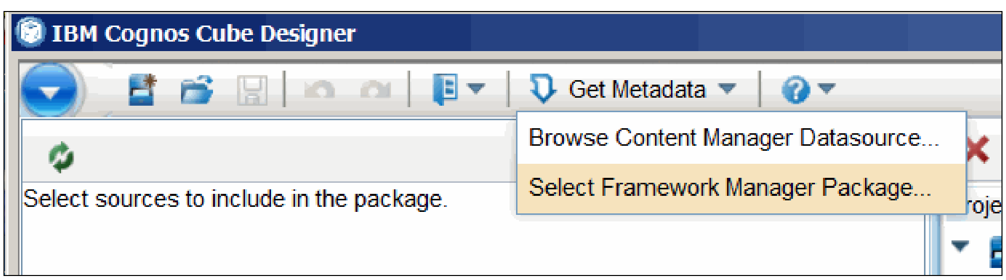


Figure 16-1 Framework Manager package import menu item

The Select a Package window is displayed with the list of the packages that are in the public folders location and in your personal `My Folders` location as shown in Figure 16-2.

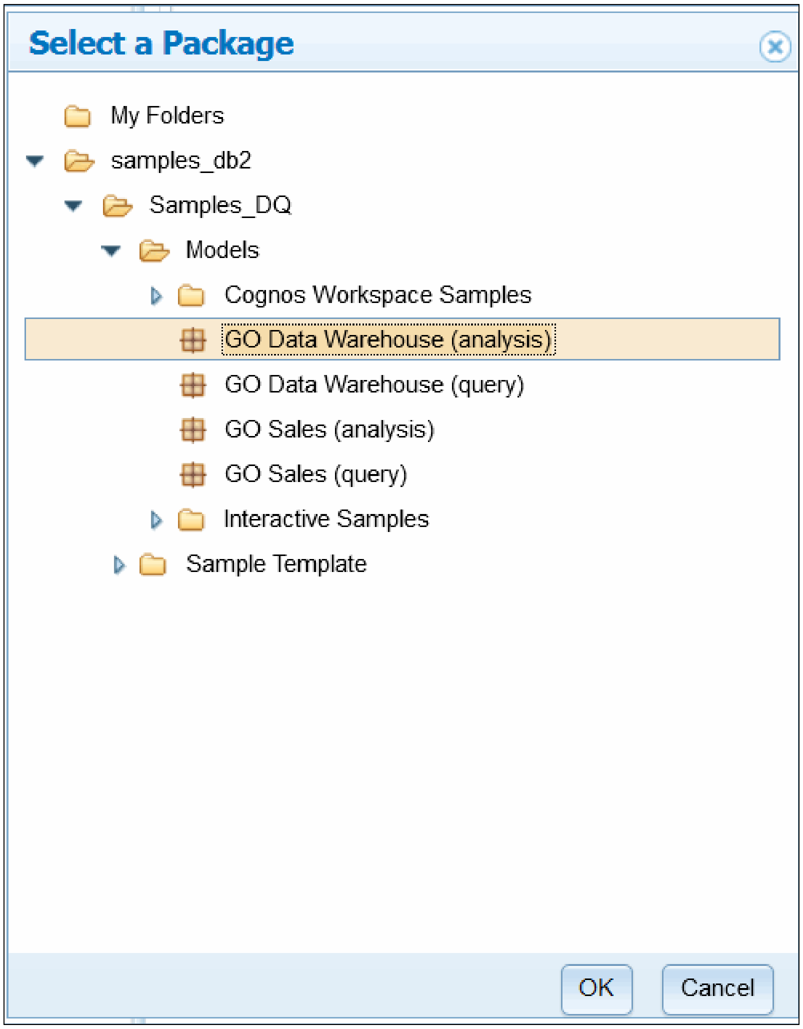


Figure 16-2 Select a Package window for importing Framework Manager models

When the import process is complete, you will see several accordion panes in the metadata view. The pane that is visible at the bottom of the list of metadata contains the Framework Manager model itself. The pane can be expanded to display some of the metadata in the Framework Manager model. Objects of this type have the familiar structure of namespaces, folders, dimensions, and query subjects that exist in the Framework Manager model. The other panes contain the data source metadata that is used to create the Framework Manager model.

Figure 16-3 shows one of the packages from the sample Framework Manager model imported into Cognos Cube Designer. There are two accordion panes in the metadata tree. The Project Explorer is emptyuntil you import the Framework Manager metadata during the dynamic cube generation phase.

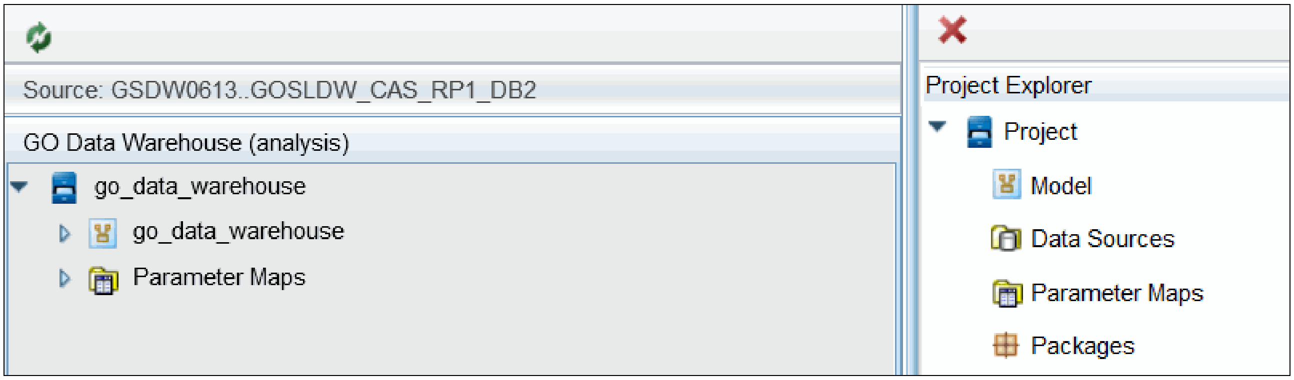


Figure 16-3 The metadata view after importing a Framework Manager model.

All objects that were included in the model are visible, including objects that were hidden in the package.

Figure 16-4 shows the Framework Manager model partly expanded.

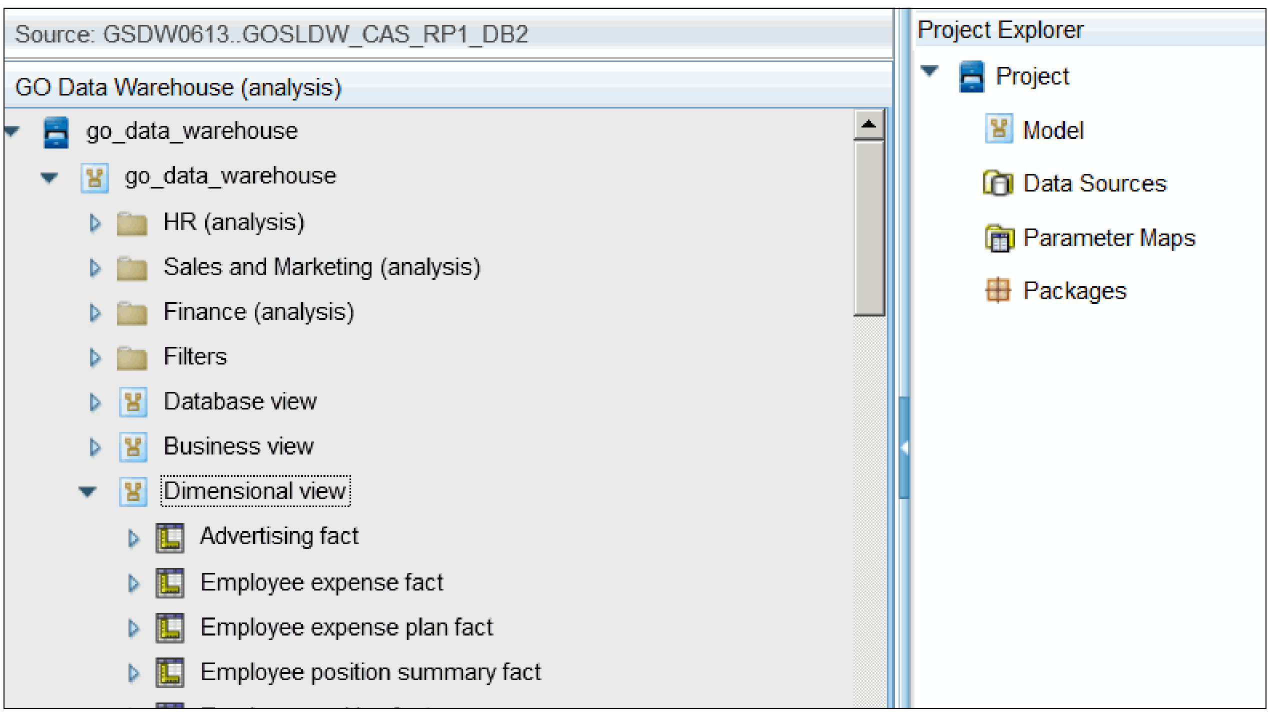


Figure 16-4 Partly expanded Framework Manager model view

Each of the data source metadata panes is derived from one of the data sources that are used in the Framework Manager model. These metadata panes look identical to metadata from data sources fetched into Cognos Cube Designer directly. Figure 16-5 shows the metadata.

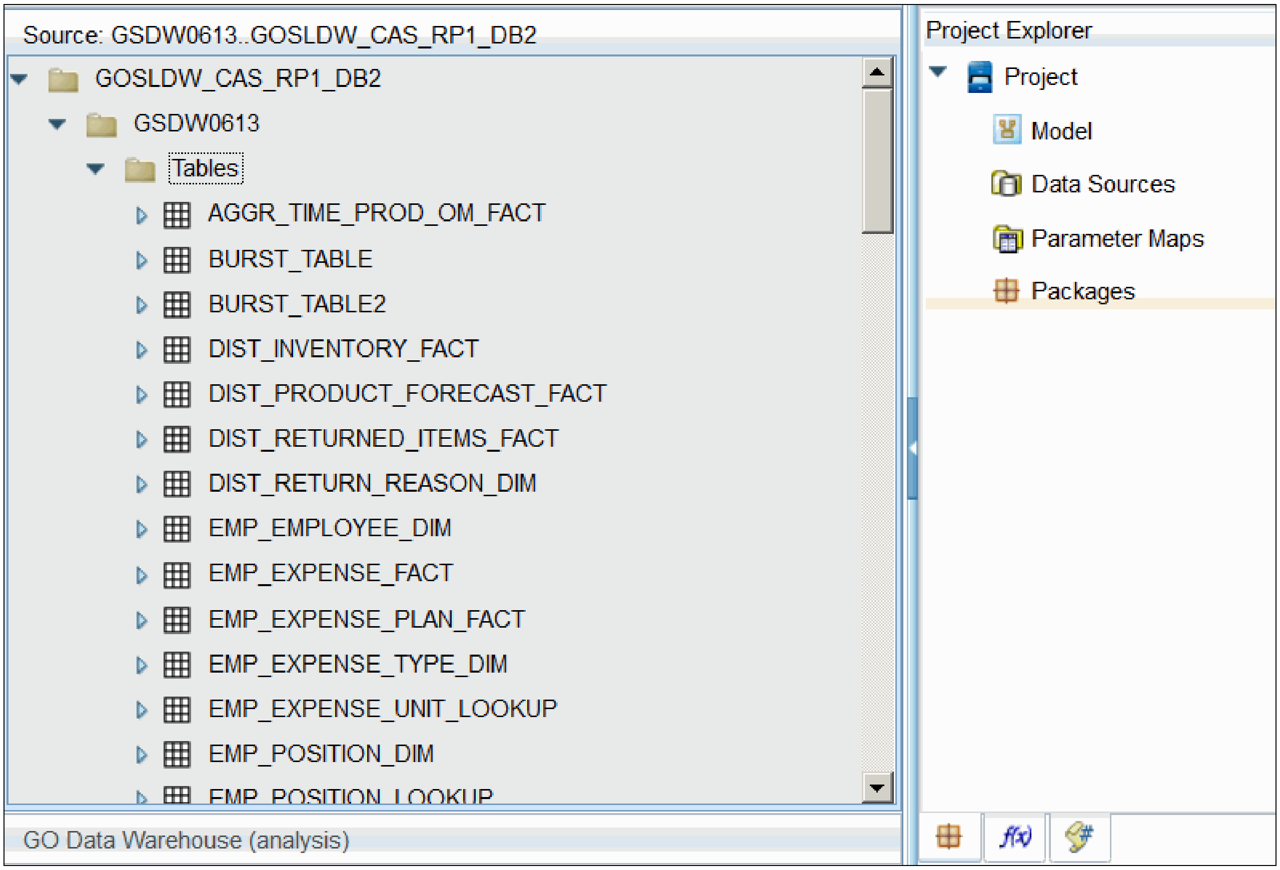


Figure 16-5 Relational database metadata from an imported Framework Manager model

You can save and close your model, and retrieve it and the metadata by clicking the appropriate menu item in the most recently used (MRU) list. The Framework Manager model metadata can be distinguished from Cognos Cube Designer model files by the absence of the `.fmd` file extension in the name.

Figure 16-6 shows an MRU list with an imported Framework Manager model and a Cognos Cube Designer model. If the imported Framework Manager model does not appear in the MRU list, you need to reimport it.

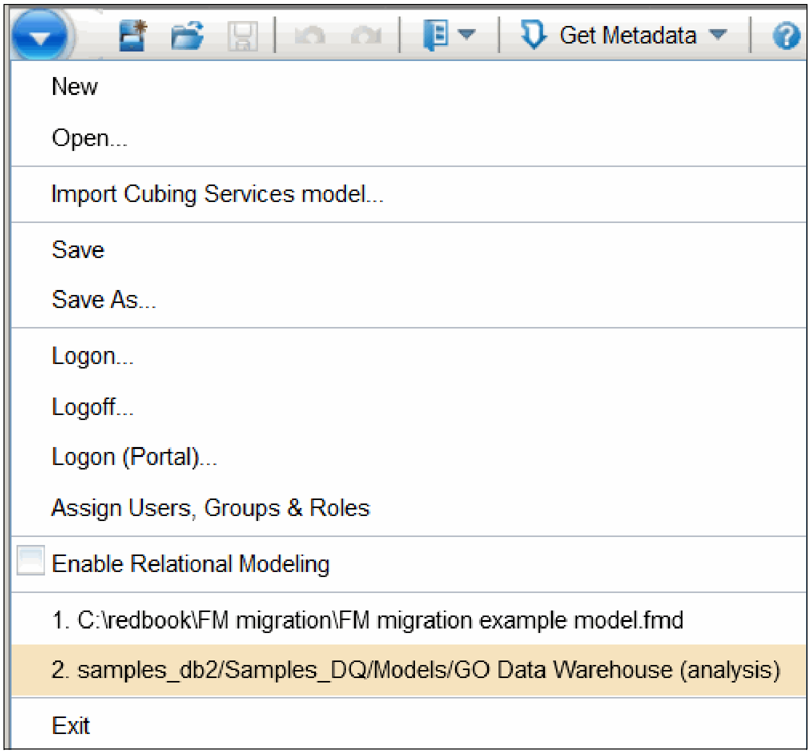


Figure 16-6 Cognos Cube Designer model and Framework Manager package in MRU list

If a package does not contain DMR or relational metadata, then the only accordion pane created is for the package model. It will be empty.

16.3.4 Generation of the dynamic cube metadata

The actions that you can perform to generate dynamic cube objects from Framework Manager depend on what object you choose to work with and what you want to do.

There are several workflows, and each of which is performed on an object of a different type and produces different objects in the Cognos Cube Designer model. These workflows give you more flexibility to migrate your Framework Manager model to Cognos Dynamic Cubes. For example, you can import a measure dimension and automatically generate its associated dimensions. You can also import a dimension.

Figure 16-7 shows one workflow where every measure dimension in a namespace is imported into Cognos Cube Designer. Table 16-1 on page 528 shows the actions that you can perform on objects you work with when generating dynamic cubes.

As part of the cleanup stage, review the generated model.

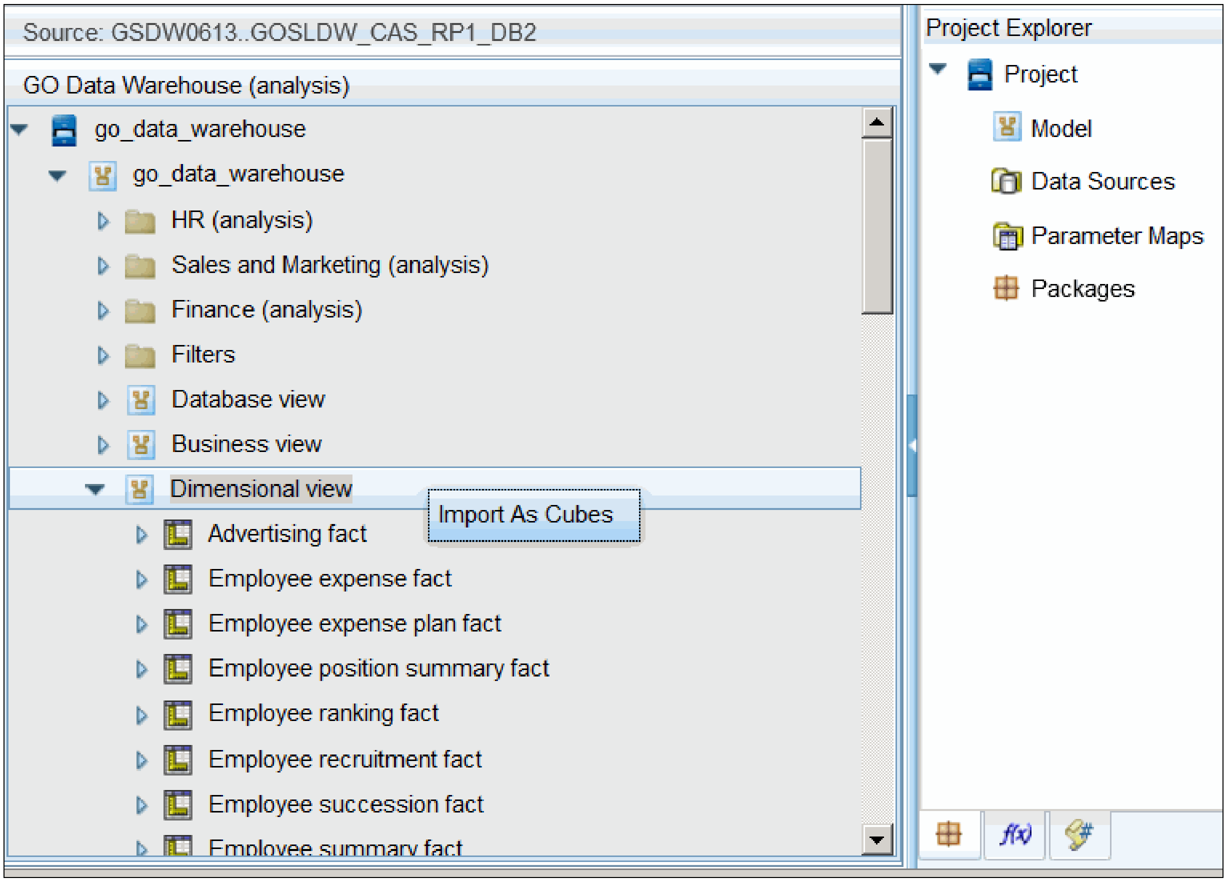


Figure 16-7 Importing all the objects of a namespace as objects in cubes

You can import multiple objects simultaneously if they are of the same type. An example is shown in Figure 16-8. Two measure dimensions are selected in the figure.

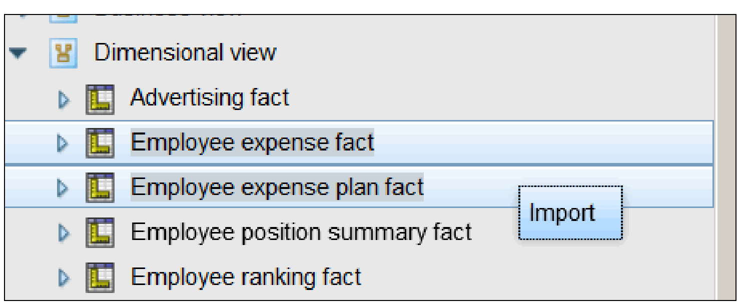


Figure 16-8 Importing two measure dimensions simultaneously.

When an object is imported, Cognos Cube Designer attempts to create an analogous Cognos Cube Designer metadata object. You should familiarize yourself with Cognos Dynamic Cubes n order to understand how Cognos Cube Designer converts the Framework Manager metadata into Cognos Cube Designer metadata.

For example, in Figure 16-9 a dimension is imported. On the left is the metadata tree displaying the source dimension from the Framework Manager model. On the right is the dynamic cube dimension. Notice that both dimensions have a hierarchy and four levels. Notice also a Levels folder in the dynamic cube dimension. You can reuse levels in multiple hierarchies in Cognos Cube Designer.

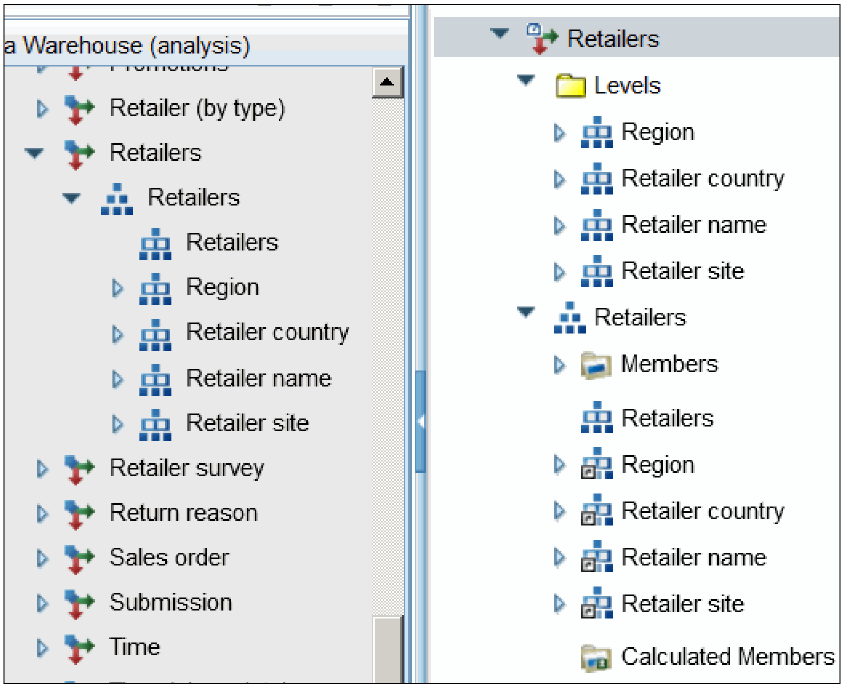


Figure 16-9 Source Framework Manager dimension and resulting Cognos Cube Designer dimension

Table 16-1 lists the Framework Manager model objects that you can work with when generating dynamic cube objects, the possible actions you can perform on them, and the expected results.

Table 16-1 Object import available actions

Object	Action	Comment
Query subjects and shortcuts to query subjects	Import as a dimension	Creates a dimension of a single level. Each query item in the query subject is generated as an attribute. It is possible that intermediate attributes are created if the expression defining the query item is sufficiently complex.
	Import as the measure dimension of a new cube	Creates a cube and puts the numeric columns that are not either primary key or foreign key columns of the query subject into the measure dimension as measures. All other columns are ignored.
Dimensions and shortcuts to dimensions	Import	Creates a dimension. The hierarchies, levels, and attributes of the source Framework Manager dimension are replicated. The business key of each level is replicated as the level key. The member caption of each level is replicated as the member caption.

Object	Action	Comment
Measure dimensions and shortcuts to measure dimensions	Import	Creates a cube, putting the measures of the source Framework Manager measure dimension into the measure dimension of the cube and creating dimensions for each dimension that has a scope relationship to the source Framework Manager measure dimension. This import action walks through the model to find the associated dimensions no matter where in the Framework Manager model the dimensions are located. Each dimension created exists outside of the cube. For this reason, you can reuse it in other cubes if you want. You can perform this action on multiple measure dimensions simultaneously. If you have inadvertently created a scope relationship in the Framework Manager model that does not have a backing relationship in the query layer, the relationship will not be created fully as there is not sufficient information to do so.
Namespaces and folders	Import as cubes	The action attempts to find measure dimensions, shortcuts to measure dimensions and shortcuts to dimensions in the namespace or folder for measure dimensions. Cognos Cube Designer attempts to create a cube for each measure dimension or shortcut to measure dimension. Each dimension in the Framework Manager model that has a scope relationship to any of the measure dimensions is created. Every dimension that has a scope relationship to a measure dimension is put into the dynamic cube of the measure dimension. A conformed dimension is put into each cube that it belongs to. The Import as Cubes function only examines the children of the namespace or folder. It does not examine nested namespaces or folders. If the namespace does not contain the necessary objects, you are informed by a message similar to the following: `BMT-MD-7117 There are no dimensions or measure dimensions in {namespace/folder name}. Select a namespace or folder with dimensions or measure dimensions in their list of direct children.`

16.3.5 Model cleanup and refinement

After you have imported Framework Manager metadata into a Cognos Cube Designer cube model, you should review the model for correctness and confirm that it reflects the intent of the original Framework Manager metadata.

The starting point is to validate the entire model and to correct any of the validation errors and warnings that are identified.

After that, you need to review the model to confirm that what has been generated conforms to what you want.

Some properties and settings are not imported so you must manually assign values to them.

For example, the concept of scope relationship does not exist in Cognos Dynamic Cubes. As a consequence, you must ensure that no dimension is modeled below the fact grains of the cubes that it participates in. For more information about multi-fact modeling, see 6.5, “Multiple fact and multiple fact grain scenarios” on page 181.

If the relationship between query subjects in the Framework Manager model is sufficiently complex, the relationship cannot be created in Cognos Cube Designer during the import process. In that case, you must create the relationships manually. All tables that are used in the underlying objects in the Framework Manager model are included in the model. Validation errors might result due to invalid relationship definitions after importing Framework Manager metadata.

In Framework Manager, if a level key is not unique, click to clear its unique level check box. Doing so brings in all the keys from the higher levels to uniquely identify the members of the level. This particular Framework Manager property is ignored during the import process in Cognos Cube Designer. As a result, you must define the additional level keys in Cognos Cube Designer to ensure the uniqueness of the members of the level.

Parameter maps in a Framework Manager model are imported. However, parameterized expressions are not resolved but are imported as strings. A list of the expressions is put into a text file in the `..\logs` directory of your Cognos Cube Designer installation location. The expressions must be resolved. In many cases, the expressions contain references to objects that do not exist in the new Cognos Cube Designer model. This situation is shown in Figure 16-10.

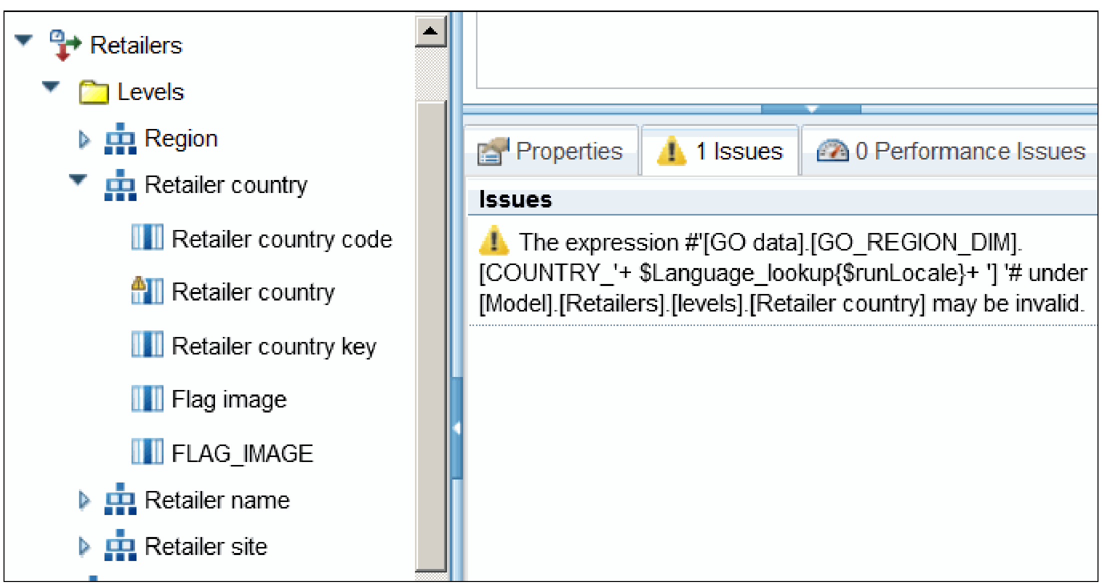


Figure 16-10 Attribute with unresolved parameterized macro expression

If you are familiar with the Framework Manager sample model, you will recognize that the expression is referring to a query item in a namespace that only exists in the Framework Manager model.

Figure 16-11 shows the expression after it is resolved. It is necessary to create an attribute in the Retailer country level with the column COUNTRY_EN from the GO_REGION_DIM table and to replace the parameterized expression with that attribute.

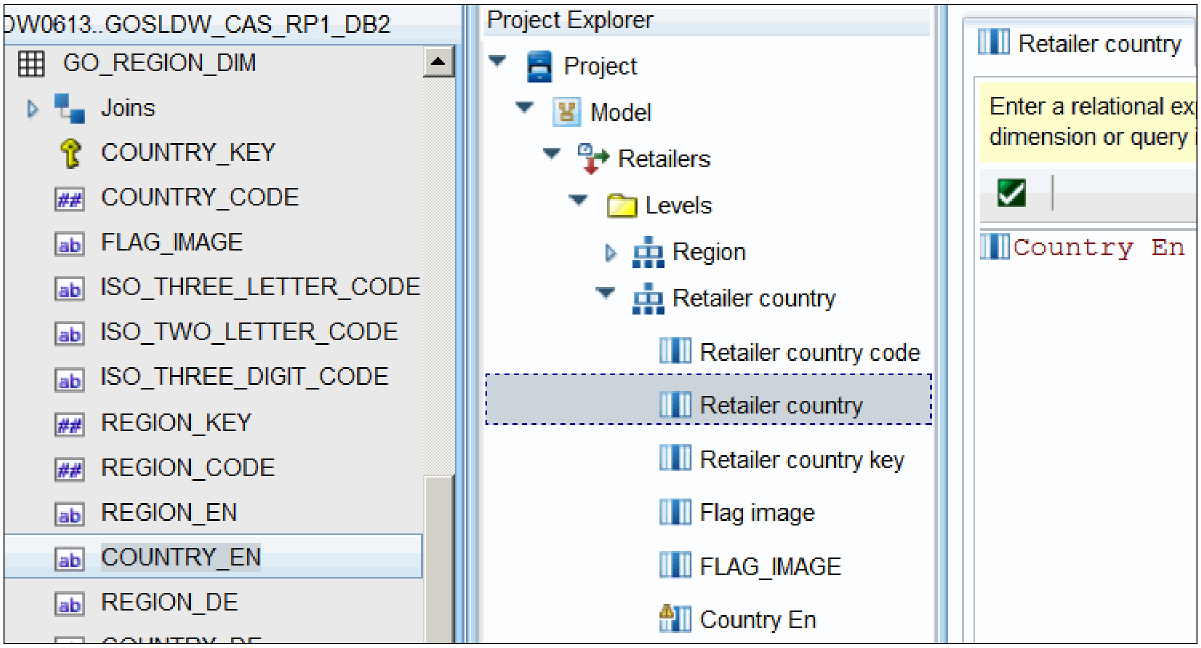


Figure 16-11 Resolved expression of a formerly parameterized macro expression.

If you import individual measure dimensions and then import others, you will cause the generation of duplicate dimensions. They will have names such as `Copy of` .... You can drag the original dimension into the new cubes and delete the copies. It is probably best if you can import all or most of your measure dimensions in one action to avoid working with the dimension copies.

You might see intermediate-expression objects created in some levels. For example, an object in the dimension might be based on an expression created in the business layer. Cognos Cube Designer must bring this object into the dimension layer. Thus, you might see objects that contain expressions that would not exist in the original Framework Manager dimension layer.

The following objects are not imported from a Framework Manager model:

- Filters and calculations. If they are used in a query subject or dimension, then their expression is incorporated into the imported dimension.
- Security. The security models in Cognos Dynamic Cubes and Framework Manager are sufficiently different as to preclude a conversion from one form to another.
- Determinants.
- Measure aggregate rules.
- The Unique level setting.

All measures are imported. However, some complex measures must be re-created as calculated measures for the timing of the aggregation of the elements of the measures to conform to the Framework Manager behavior. For example, any measure that has as part of its expression other measures that use aggregate rules must be re-created as a calculated measure.

17

Hardware sizing

This chapter provides information about estimating hardware requirements (CPU, memory, and disk) for one or more dynamic cubes. This chapter discusses the available methods, formulas, and considerations for estimating hardware requirements.

For more information, see *IBM Business Analytics Proven Practices: Dynamic Cubes Hardware Sizing Recommendations* at the following website:

http://www.ibm.com/developerworks/library/ba-pp-infrastructure-cognos_specific-page681/index.html

This chapter refers to IBM Cognos Dynamic Query Analyzer and Aggregate Advisor. For information about installing and configuring Cognos Dynamic Query Analyzer and running Aggregate Advisor, see Chapter 3, “Installation and configuration of IBM Cognos Cube Designer and IBM Cognos Dynamic Query Analyzer” on page 65.

This chapter contains the following sections:

- Using Proven Practices documentation
- Using the hardware sizing calculator in IBM Cognos Cube Designer
- Applying estimated values to the cube configuration
- Hardware requirements additional considerations

17.1 Using Proven Practices documentation

The document *IBM Business Analytics Proven Practices: Dynamic Cubes Hardware Sizing Recommendations* covers both a high-level and a detailed approach to estimating hardware requirements for dynamic cubes. This document is available from IBM developerWorks, Business analytics proven practices at:

http://www.ibm.com/developerworks/library/ba-pp-infrastructure-cognos_specific-page681/index.html

This section prepares the reader for using the *Dynamic Cubes Hardware Sizing Recommendations* document to size one or more dynamic cubes within a single query service. It provides an overview of the process to estimate the hardware size for a dynamic cube and explains the applicability of the proven practices document for more advanced sizing efforts.

In most cases, estimating hardware sizing requirements is best handled by the hardware sizing calculator in Cognos Cube Designer (Estimate Hardware Requirements feature) described in 17.2, “Using the hardware sizing calculator in IBM Cognos Cube Designer” on page 534.

In some cases, the proven practices document can provide for extra considerations or serve as a preliminary estimate before installing Cognos Cube Designer and modeling the cube. Also, if you want to have a deeper understanding of the basis for the estimates provided by the hardware sizing calculator (Estimate Hardware Requirements feature), the proven practices document is an excellent resource.

The proven practices document begins with a high level sizing recommendation and then expands on the detailed information about sizing. The high level sizing section might be useful for audiences that are simply looking for general guidance based on small, medium, large, or extra-large configurations before installing any software or modeling the cube. The detailed sizing section covers details explaining the various caches, and calculations for estimating CPU, disk, and memory requirements.

Although it might be interesting, and sometimes necessary, to understand the details behind the calculations, it is by far easier and more straightforward to use the hardware sizing calculator in Cognos Cube Designer. Situations where you might need to use the *Dynamic Cubes Hardware Sizing Recommendations* document are described in 17.4, “Hardware requirements additional considerations” on page 538.

Whether using the Estimate Hardware Requirements feature or the formulas presented in the proven practices document, after the estimated size of the cube (core, memory, and disk) is calculated, see 17.3, “Applying estimated values to the cube configuration” on page 537 for instructions on where to apply the results.

17.2 Using the hardware sizing calculator in IBM Cognos Cube Designer

When creating the cube model, estimate the hardware requirements for your cube. IBM Cognos Cube Designer provides a special function, the *hardware sizing calculator,* to help you estimate memory, CPU, and disk requirements based on the cube size and the expected load. This function is available in IBM Cognos release 10.2.1 Fix Pack 3 and later as the Estimate Hardware Requirements feature.

17.2.1 Starting the hardware sizing calculator

To start the hardware sizing calculator, right-click the cube instance in the Project Explorer and select **Estimate Hardware Requirements** (see Figure 17-1).

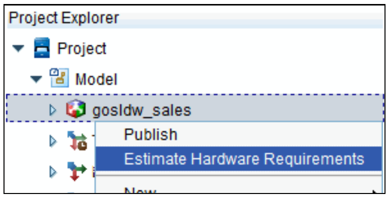


Figure 17-1 Starting the hardware sizing calculator

A new window opens where you can modify cube size and load parameters and receive guidance on the hardware that the cube requires (see Figure 17-2).

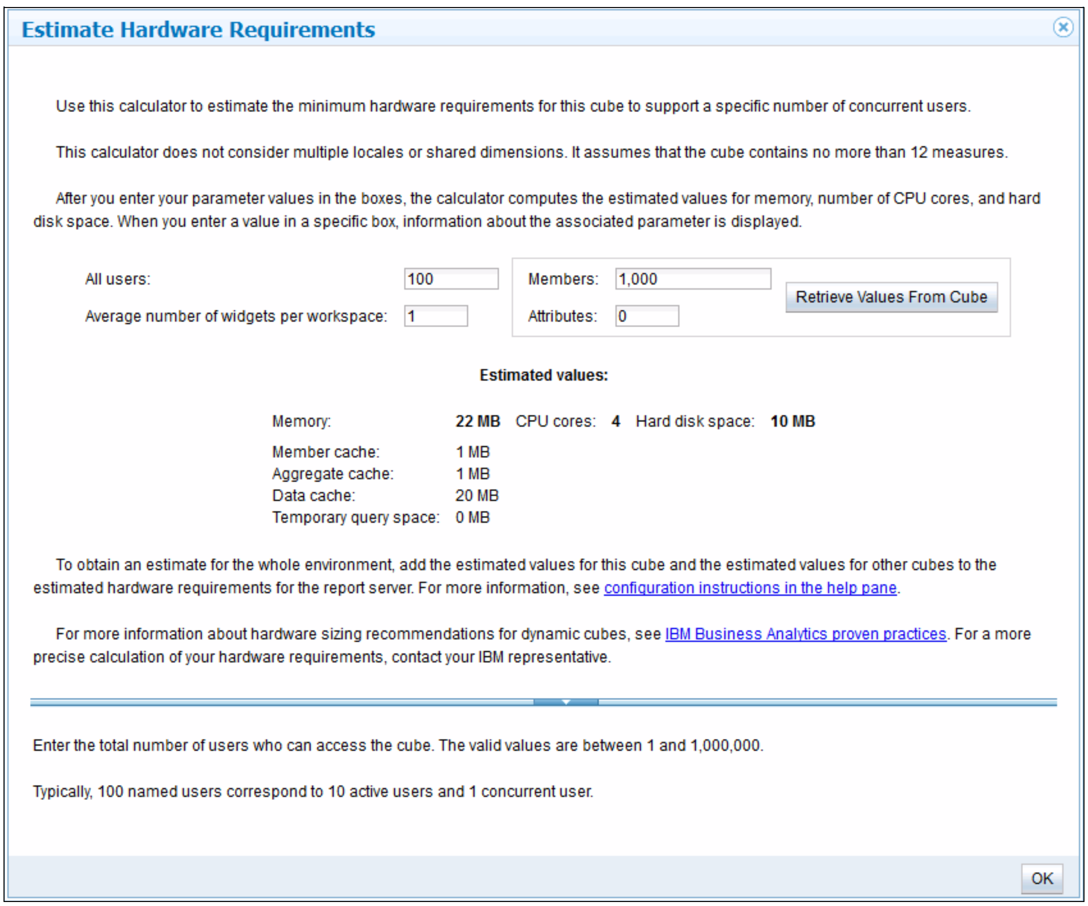


Figure 17-2 Hardware sizing calculator

Note:

- The hardware sizing calculator should only be used for rough initial estimation of hardware requirements. For a more detailed description of hardware estimation methodology, see the *Dynamic Cubes Hardware Sizing Recommendations* document. If you require help with a more precise calculation of your hardware requirements, contact your IBM representative.
- Hardware sizing calculator currently does not support virtual cubes, and does not consider scenarios of multiple locales or shared dimensions. Virtual cubes are discussed in the *Dynamic Cubes Hardware Sizing Recommendations* document. The hardware sizing calculator also assumes that the cube contains no more than 12 measures. The effect of defining a larger number of measures is described in 17.4.3, “Number of measures in a single cube” on page 539.

You can modify these four parameter fields:

- *All users*. Indicates the total number of users in the range 1 - 1,000,000 that are expected to access the cube. From this number of total users, a number of concurrent users is calculated, using the assumption that 100 named users correspond to 10 active users, which in turn corresponds to one concurrent user.
- *Average number of widgets per workspace*. Indicates the number of queries (1 - 100 queries) in the report that an average user is likely to run in parallel. This parameter helps to estimate the concurrent load on the system. Several concurrent queries run by one user generate load on the system equivalent to that many concurrent users running one query each. By default multiple queries in a complex report are run sequentially unless configured otherwise or unless the report is an IBM Cognos Workspace report.
- *Members*. Indicates the approximate number of members in the cube in the range 1 - 1 billion. It can be approximated by adding up the number of members in the two largest hierarchies.
- *Attributes*. Indicates the average number of member attributes (in the range 0 - 100). This number should be a *weighted average*, which is the number of attributes in each level weighted by the number of members in that level. Default attributes (such as caption) should not be included. This number can be approximated by calculated the weighted average number of attributes in the leaf levels of two largest hierarchies.

As you click any field, a help text for that parameter is displayed in the help pane in the bottom part of the window.

Below the parameter value fields, the current estimated values for the following hardware parameters are provided:

- Memory. The combined size of the following four types of storage:
  - Member cache
  - Aggregate cache
  - Data cache
  - Temporary query space (memory area used for query analysis and processing)
- CPU cores. The number of CPU cores that are needed to support the cube.
- Hard disk space. The disk space that is required for temporary files.

17.2.2 Changing parameter values

Hardware size values can be estimated based on the current cube model and the underlying data. To request estimates, click **Retrieve Values From Cube**.

The hardware sizing calculator analyzes the hierarchy sizes and the defined attributes, and provides estimates for members and attributes based on that analysis. See Figure 17-3 for an example.

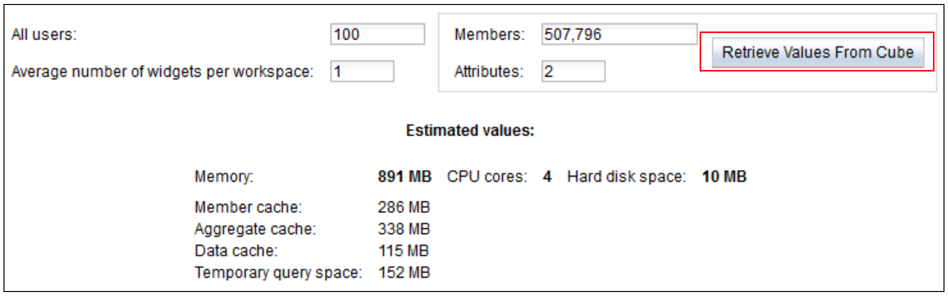


Figure 17-3 Estimated values for the gosldw_sales cube.

Estimating the number of members and attributes involves running queries against the database on which the model is based. If the size of this database differs in scale from the database to be used in production, then instead of leaving it up to the hardware sizing calculator to estimate the values, enter them manually.

You can modify any field value and the estimated size values are automatically recalculated and adjusted. A warning describing valid value ranges is displayed if an incorrect value is entered.

Note: If the parameter values that you enter cause the memory estimate to increase above 300 GB, the hardware sizing calculator recommends setting up multiple instances of this cube on different servers to avoid performance degradation.

17.3 Applying estimated values to the cube configuration

Regardless of which method was used to calculate the size estimates, you can apply the values through IBM Cognos Configuration. The values apply to the JVM for the query service and also to cache size limits for individual cubes.

Note: The estimates that are provided are for a single cube scenario. If you are hosting multiple cubes on the same IBM Cognos instance, then estimates for all the cubes should be considered together. Report server hardware requirements should be considered in addition to cube size requirements.

The estimated `Memory` value can be used to adjust JVM memory sizes in the query service configuration. The fields that can be adjusted with this value are `Initial JVM heap size for the query service` and `JVM heap size limit for the query service` (see Figure 17-4).

Figure 17-4 JVM memory properties in query service configuration

You can choose to leave the initial JVM heap size at the default 1024 value and allow the JVM to manage its own expansion. Alternatively, you can set the initial JVM heap size close to your estimated value to avoid the JVM having to increase its size during start.

In addition to the overall JVM heap size for the query service, each individual cube has properties to govern the size limits for caches:

- The `Aggregate cache` value can be entered in the field `Maximum space for in-memory aggregates`.
- The `Data cache` value can be entered in the field `Data cache size limit`.
- The `Hard disk space` value can be entered in field `Maximum amount of disk space to use for result set cache` (see Figure 17-5).

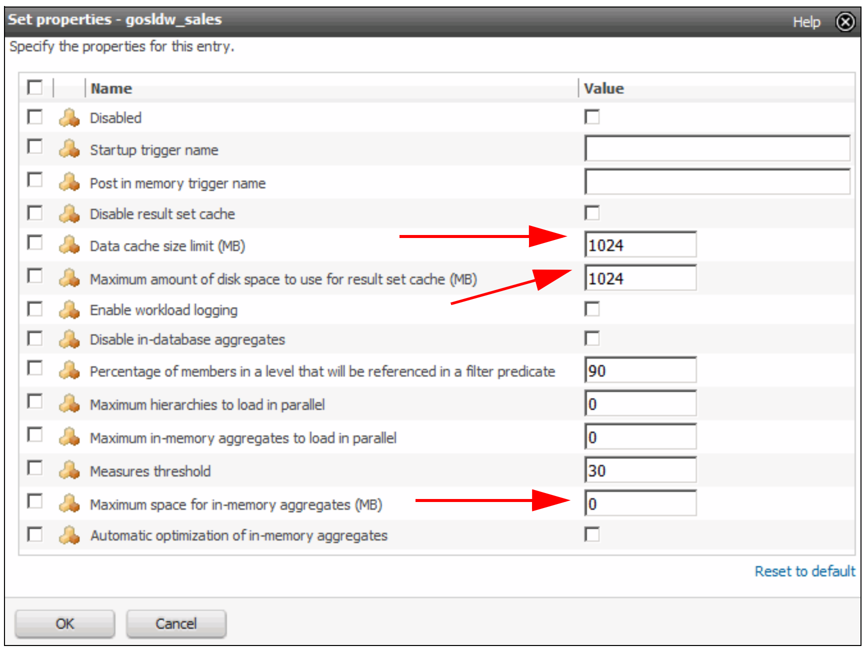


Figure 17-5 Cube property fields that can be adjusted based on estimated values

> **Note:** Setting the cache size limits to a large number does not increase or waste memory. Only the amount of memory that is required to hold the defined aggregates is used. For example, if it takes 35 MB to hold the aggregates for `gosldw_sales`, and the aggregate caches size is set to 1 GB, only 35 MB of memory is used. Over time, if the underlying fact tables grow, the aggregates are allowed to grow to the specified maximum of 1 GB.

17.4 Hardware requirements additional considerations

In some cases, additional considerations must be taken into account when estimating the hardware requirements for a dynamic cube. These considerations arise when your cubes do not conform with the base use case described in 17.2, “Using the hardware sizing calculator in IBM Cognos Cube Designer” on page 534. Not all possible circumstances are covered in this chapter, but the most common scenarios are addressed. Keep in mind that these considerations are in addition to the information about hardware sizing already provided in the previous sections of this chapter.

17.4.1 Multiple locales and hardware implications

Dynamic cubes support multiple locales. This feature allows you to display dimension members, attributes, and measure names in multiple languages. As a result of loading multiple values for these elements, the size of the member cache increases. Generally speaking, each additional locale increases the base member cache size by approximately 6%. A more accurate estimate can be calculated by using formulas provided in the *Dynamic Cubes Hardware Sizing Recommendations* document.

17.4.2 Shared dimensions and hardware implications

Dimensions can be shared across multiple cubes within the same query service instance. Because a shared dimension is loaded once and then shared across multiple cubes, the time that it takes to load multiple cubes is reduced. In addition, the amount of memory that is used for member cache is reduced. This situation is particularly relevant when working with large dimensions.

To understand the impact of using shared dimensions, use the *Dynamic Cubes Hardware Sizing Recommendations* document to calculate individually the size of the member cache for each shared dimension. Subtract this value from your member cache estimate for each additional cube that uses this shared dimension.

As a simple example, consider an environment with three cubes. Each cube has an estimated member cache of 6 GB. Each cube has a product dimension of 3,000,000 members.

Using the formula `<# of members in largest hierarchies> * 550` bytes to calculate the estimated member cache of the product dimension, you can estimate that the product dimension requires approximately 1.65 GB of memory.

By sharing the product dimension across all three cubes, instead of using 3 x 1.65 GB for the product dimension, the member cache requires only 1 x 1.65 GB for the product dimension.

Keep in mind that this applies only to the member cache in this context. The size of the data cache, aggregate cache, and temporary query cache are not affected by the presence of shared dimensions.

17.4.3 Number of measures in a single cube

The formulas presented in the *Dynamic Cubes Hardware Sizing Recommendations* assume that any single cube has 10 - 12 measures. In the case where a significantly larger number of measures exist, and they are heavily used by multiple reports, extra memory will be required for the data cache. Adjusting the cache to the optimal size due to many measures might require an iterative approach of increasing memory and observing performance. Setting the cache size too large can have a negative effect on performance due to the processor burden associated with managing a larger than necessary JVM.

17.4.4 User-defined in-memory aggregates and hardware implications

User-defined in-memory aggregates are defined in Cognos Cube Designer. After the in-memory aggregates are defined, the cube must be published so that the in-memory aggregate definitions can be read from the content store by the Aggregate Advisor in IBM Cognos Dynamic Query Analyzer. The Aggregate Advisor results display the estimated space required (in bytes) for each in-memory aggregate whether they are user-defined or generated by the Aggregate Advisor.

Figure 17-6 shows the General tab of the Aggregate Advisor Results. There are four user-defined in-memory aggregates appearing at the beginning of the list of In-Memory Recommendations.

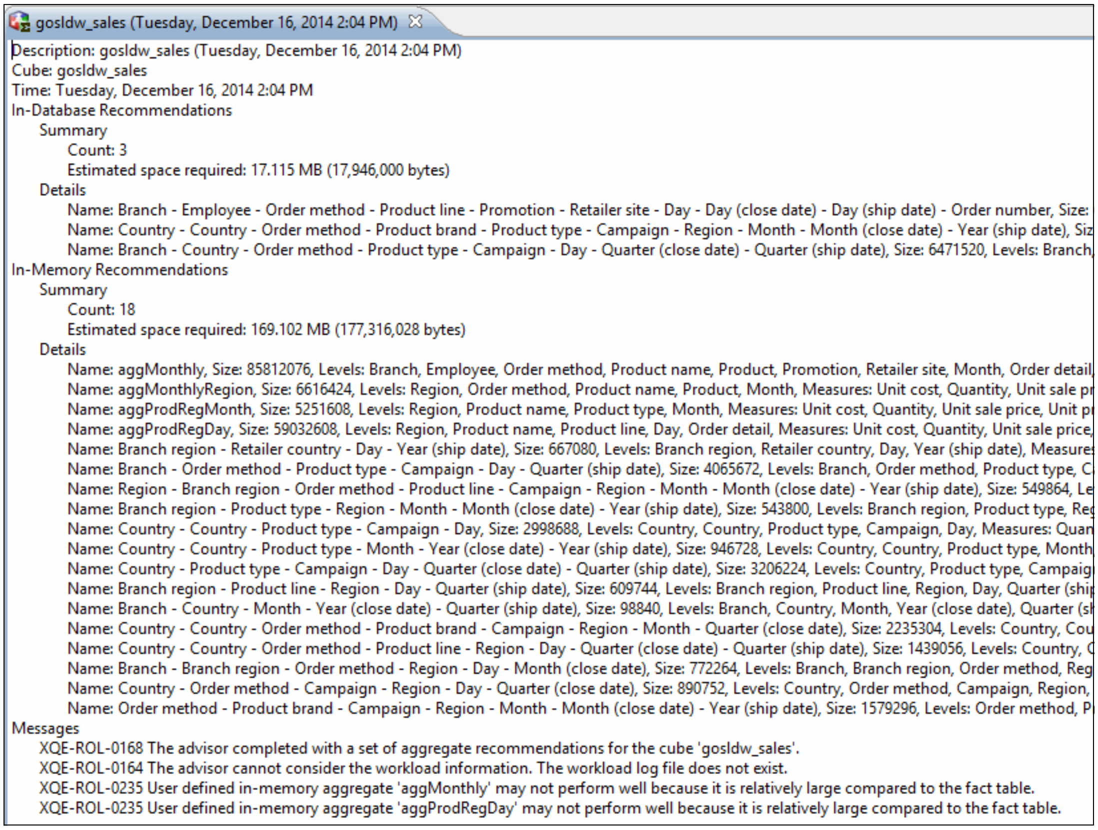


Figure 17-6 Aggregate Advisor Results, General tab

In the Advisor Results view, select **Apply Selected In-Memory Recommendations** from the **File** menu. The Select the aggregates to apply window is displayed (see Figure 17-7). The Estimated Space Required is updated based on the current recommendations selected.

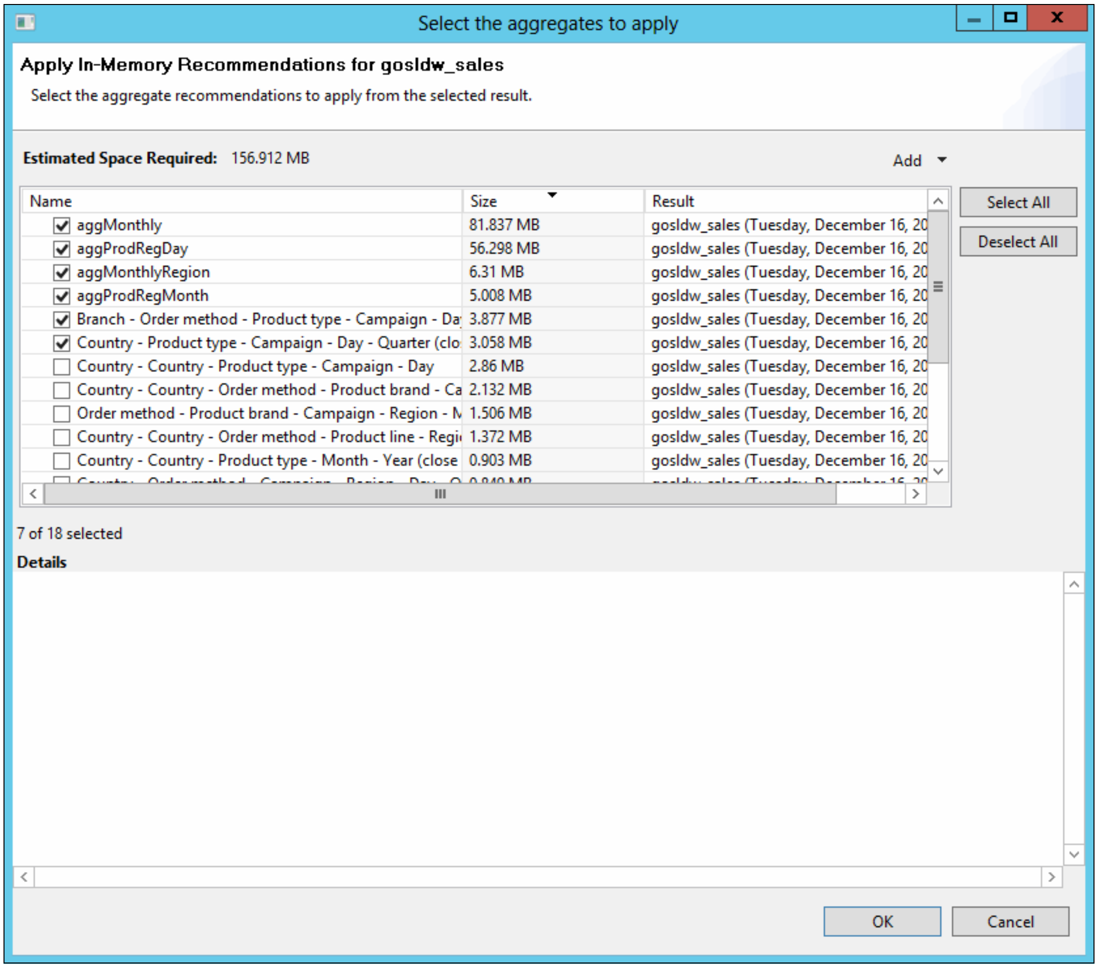


Figure 17-7 Select the aggregates to apply

The amount of memory to allocate to aggregate cache should consider the estimated space that is required for in-memory aggregates. If there is not enough memory assigned for the aggregate cache, in-memory aggregates will not be loaded.

> **Note:** There is a relationship between data cache and aggregate cache that must be understood when adjusting the JVM heap size. Briefly, as the aggregate cache increases, the need for data cache should decrease. To fully understand this relationship, see the data cache and aggregate cache section of the *Dynamic Cubes Hardware Sizing Recommendations* document.

Set the `Maximum space for in-memory aggregates` cube property to a value that is greater than the advisor-estimated size so that all the in-memory aggregates can be loaded. The steps to apply this property are described in 11.4, “In-memory aggregates” on page 367.

17.4.5 Attributes and hardware implications

Each additional attribute adds approximately 20 bytes to each member. When multiple locales are used, each additional locale adds an extra 20 bytes per attribute per member. Example 17-1 shows an example of calculating additional memory requirements to accommodate attributes.

Example 17-1 - Calculating memory requirements for attributes

```
Assume a dimension of 100,000 members. To calculate the estimated increase in
memory to accomodate 4 attributes (only 1 locale), the formula is:

#members * #attributes * #bytes * #locales = estimated additional memory

100,000 * 4 * 20 * 1 = 8,000,000bytes = 8 MB
```

You can also use the hardware sizing calculator in Cognos Cube Designer to calculate the impact of attributes on member cache. However, the interface does not allow for calculating multiple dimensions of various sizes with various numbers of attributes.

In a real world environment, many dimensions have multiple attributes. It is important to remember to calculate the memory requirements for sets of attributes for each dimension, then sum those results to determine the total increase in memory requirements.

18

Dynamic cubes in a multi-dispatcher environment

This chapter describes how to deploy, manage, and work with dynamic cubes in an IBM Cognos Business Intelligence (BI) system that is scaled out with multiple IBM Cognos BI servers. A dynamic cube can be deployed with an instance on multiple Cognos BI servers to increase processing capacity. The Cognos BI server dispatcher can automatically balance the load across all instances of a cube.

This chapter contains the following sections:

- Overview of a multi-dispatcher environment
- Deploying and managing cubes in a multi-dispatcher environment
- Database impacts in a multi-dispatcher environment
- Aggregate Advisor in a multi-dispatcher environment
- Cube Designer in a multi-dispatcher environment

18.1 Overview of a multi-dispatcher environment

This section establishes context for the remainder of the chapter by providing an overview of an IBM Cognos BI multi-dispatcher environment. For more information, see the following documentation:

- *IBM Cognos Business Intelligence Version 10.2.2 Architecture and Deployment Guide*

  http://public.dhe.ibm.com/software/data/cognos/documentation/docs/en/10.2.2/crn_arch.pdf
- *IBM Cognos Business Intelligence Version 10.2.2 Installation and Configuration Guide*

  http://public.dhe.ibm.com/software/data/cognos/documentation/docs/en/10.2.2/inst_cr_winux.pdf
- *IBM Cognos Business Intelligence Version 10.2.2 Administration and Security Guide*

  http://public.dhe.ibm.com/software/data/cognos/documentation/docs/en/10.2.2/ug_cra.pdf
- *IBM Business Analytics Proven Practices: IBM Cognos BI Dispatcher Routing Explained*

  http://www.ibm.com/developerworks/data/library/cognos/infrastructure/cognos_specific/page510.html

18.1.1 Cognos BI server topology

The IBM Cognos BI server has a multitiered distributed architecture. The Cognos BI server consists of three separately installable components:

- Gateway
- IBM Content Manager
- Application Tier Components

The gateway component runs on a web server in the web tier and is the entry point for user requests. The gateway is configured with an ordered list of dispatchers and relays requests to the first available dispatcher in the list. The first dispatcher in the list is referred to as the *primary*, *preferred*, or *front* dispatcher. The front dispatcher acts as a single entry point for all requests in the application tier and load balances report requests across other dispatchers.

The Content Manager component is a service that manages system and application data. The Content Manager component stores information in a repository called the *content store database*. You can install multiple Content Managers, but only one is active at any time. Other Content Managers are on standby for failover. There can only be one content store database.

An Application Tier Components installation includes one dispatcher and several services, such as the report service, batch report service, and query service. When there is more than one Application Tier Components installation, the system is referred to as a *multi-dispatcher environment*.

This chapter focuses on the application tier, scaled out with multiple Application Tier Components installations across multiple computers with multiple instances of a dynamic cube. This chapter assumes a single gateway and an application tier with multiple dispatchers using built-in load balancing.

Deploying multiple instances of a dynamic cube has these benefits, among others:

- Improved performance by distributing the query workload over multiple servers.
- Improved fault tolerance through failover.

Interactive and scheduled report requests are handled by the report service and batch report service. The report service and batch report service are implemented by an external process known as the report server (RS). The RS process name is `BIBusTKServerMain`. The report service and batch report service can each be configured to run multiple instances of the RS. By default, the number of RS processes for both the report and batch report service is two. Each instance of the RS is multi-threaded and can run multiple reports in parallel.

The query service processes requests from the report and batch report service. Dynamic cubes are implemented within the query service. There can only be one instance of the query service per Application Tier Components installation.

The dispatcher manages the services and routes requests. Requests are routed to either a local service or to another dispatcher running the required service. Dispatchers register with the Content Manager at start and are continually made aware of the presence and state of other dispatchers and services by interacting with the Content Manager.

Figure 18-1 shows an example of multi-dispatcher environment with multiple Application Tier Components.

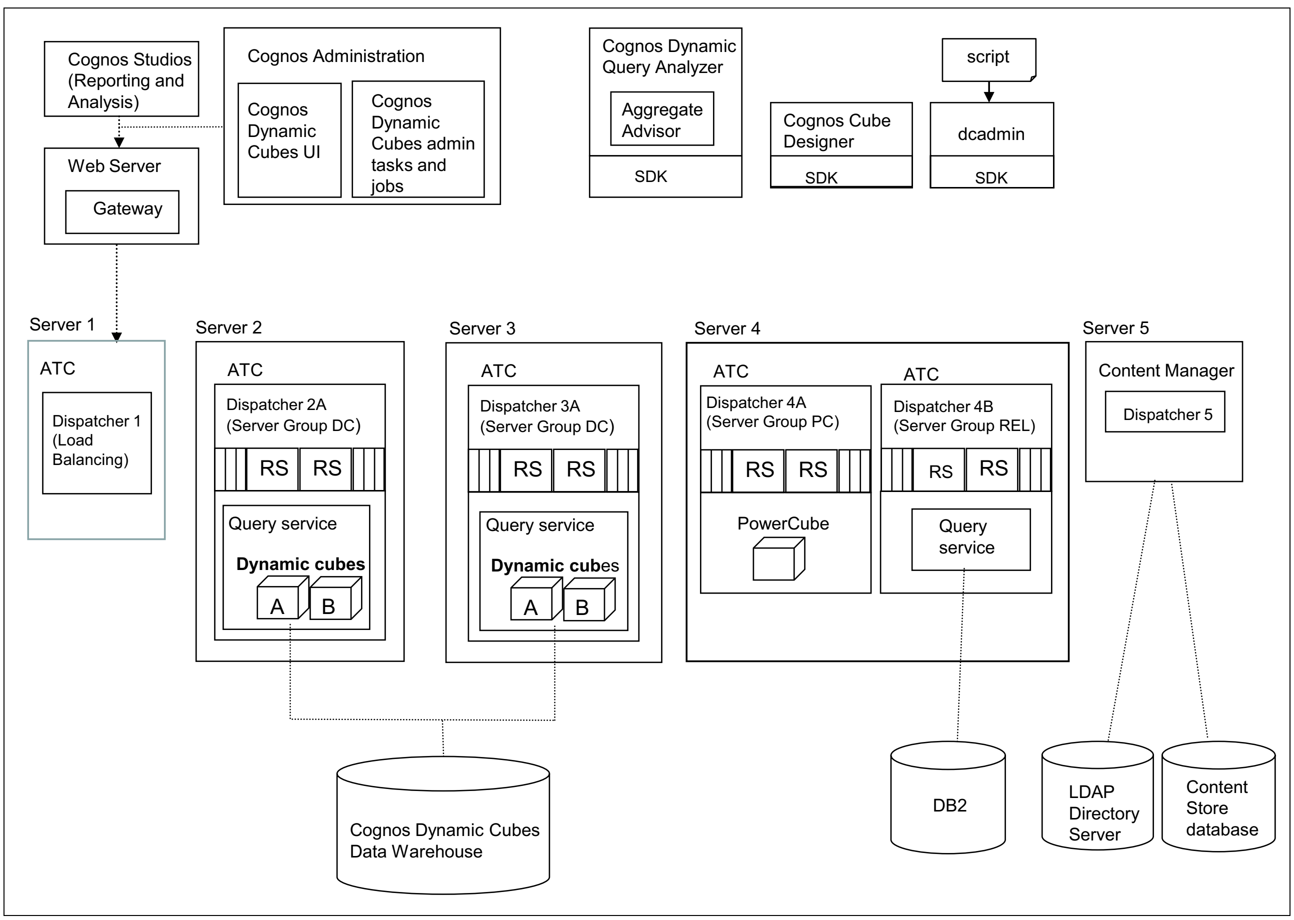


Figure 18-1 IBM Cognos BI multi-dispatcher environment

18.1.2 Dispatcher routing

Dispatchers route requests to services and perform load balancing and failover. The gateway relays user requests to a single application tier entry point referred to as the *front dispatcher*. This dispatcher in turn either services the request by using a local service, or routes it to a service on another dispatcher. There are two dispatcher load balancing modes: *Weighted round-robin* (which is the default) and *cluster compatible*. This document focuses on weighted round-robin algorithm.

The front dispatcher load balances requests, and regardless of which dispatcher is configured as the front dispatcher, load balancing only occurs once. Every dispatcher has information about all the other dispatchers, their server groups, and the services running on them. When a request reaches the front dispatcher, it is routed to a service running on a dispatcher in the correct server group. Only the front dispatcher incurs the routing processor burden.

If a certain dispatcher is running the report service, but has a relatively light reporting load, it might be a good choice as the front dispatcher. A more common approach is to dedicate the front dispatcher exclusively to load balancing by disabling the report service and other services unrelated to load balancing. With the *load balancing dispatcher* approach, a dispatcher that is fielding report requests does not incur routing processor burden that could affect report throughput.

Note: For simplicity, in the context of this chapter, a *report request* refers to any request associated with the authoring and running of a report or analysis.

By default after system installation, report requests are balanced equally across all dispatchers in the system that are running a report service. The underlying assumption is that a report request can be processed by any report service on any dispatcher. If this assumption does not hold because certain report requests can only be serviced by certain dispatchers, or if the assumption does hold but you want to explicitly route requests to specific dispatchers, *routing rules* must be defined. For example, if some dispatchers are in the 32-bit report server execution mode for compatible query mode data sources, and other dispatchers are in the 64-bit report server execution mode for memory-intensive dynamic cube reporting, routing rules are required to ensure that compatible query mode requests are only routed to 32-bit report servers, and dynamic cube requests are only routed to 64-bit report servers. A routing rule is defined in terms of server groups and routing sets. Therefore, server groups and routing sets must be defined before specifying routing rules.

Server groups

By default after installation, a dispatcher is in a built-in server group named `DefaultServerGroup`. All dispatchers must logically be in a server group and the `DefaultServerGroup` acts as a catch-all group. The `DefaultServerGroup` is a logical system entity only and cannot be subject to user-defined routing rules.

The dispatchers in a system can be organized into user-defined *server groups*, where a server group is a set of dispatchers. Each dispatcher can only be in one server group at a time. User-defined server groups allow for advanced routing by using routing rules where requests specific to a package, role, and group can be directed to a specific server group. This configuration allows different servers and dispatchers to be dedicated to different purposes. Perhaps one or more server groups are dedicated to dynamic cubes, whereas another server group is dedicated to some other data source or to a specific user group with servers in a different geography that is physically located close to the users.

Note: Only assign dispatchers that are running a report service to an explicit user-defined server group. To do otherwise might not necessarily cause an immediate problem, but it does not make sense.

A dispatcher is assigned to a server group by entering a user-defined string in the dispatcher `Server group` property by using Cognos Administration, as shown in Figure 18-2. An empty server group string causes the dispatcher to be assigned to the built-in `DefaultServerGroup`.

Explicit server groups are automatically created and deleted based on the current value of the dispatcher Server group property. A server group is automatically created when a Server group name that is not currently in use is specified. A server group is automatically removed when a name is no longer used by any dispatcher.

Figure 18-2 The dispatcher Server group property

Routing sets

Before creating a routing rule, you must create routing sets. A routing set is created by assigning a keyword string to a package, group, or role object. The same keyword can be assigned to multiple objects. For example, multiple packages can be included in the same routing set by assigning the same keyword to each package. Define separate routing sets for packages, groups, and roles.

By default, the only routing sets are the built-in wildcards: (Any package), (Any group), and (Any role) (see Figure 18-4 on page 549). You must explicitly create other routing sets on package, group, and role objects as required. A routing set is created by setting the properties on a package in Cognos Connection, or setting properties on a group or role in Cognos Administration, and specifying a keyword string on the Assign routing sets page. After they are defined, the routing set keyword values appear in the picklists on the Specify routing rules window.

For example, use the following procedure to create a routing set, or to assign an existing routing set, to a package:

1. Navigate to the package in Cognos Connection and in the Actions column, click the **Set properties** icon, and go to the Advanced routing pane on the General tab.
2. Select **Override the routing sets acquired from the parent entry**, and select **Set** under Routing Sets.

3. Type a keyword string in the **Type routing sets** field, for example, `Dynamic Cube Sample Packages` and click the **Add new routing sets** arrow. Figure 18-3 shows the Assign routing sets window. Click **OK** to save

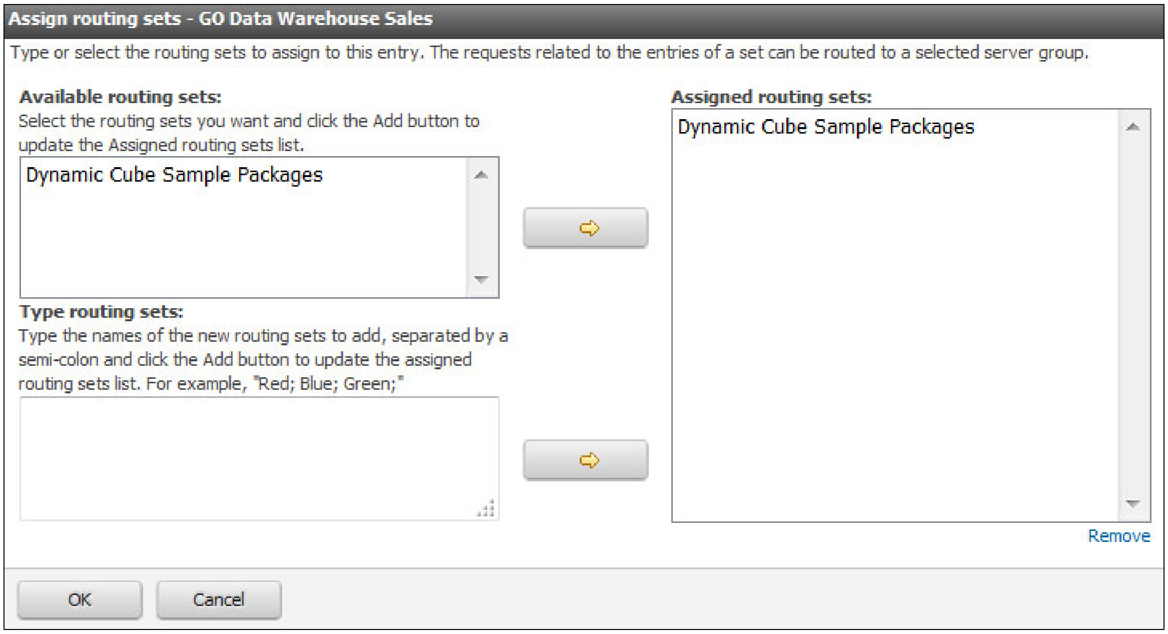


Figure 18-3 Assign routing sets page

Follow a similar procedure to create a role or group routing set.

Routing rules

The dispatcher uses *routing rules* to route report requests to specific server groups. A routing rule is a combination of a package routing set, group routing set, role routing set, and a server group. A rule is resolved by ANDing the package, group, and role conditions. If the package, group and role of a specific report request match a rule, the request is dispatched to the server group specified in the rule.

Routing rules are ordered. The rules are processed in sequence and the first matching rule is used. When there is not a matching rule, or when no rules are defined, report requests are load-balanced across all dispatchers in the system that are running a report service.

To create a routing rule, complete these steps:

1. Open Cognos Administration, click **Configuration**, select **Dispatchers and Services,** and click the **Specify Routing Rules** icon to open the **Specify the routing rules** window.
2. Click **Add a rule** and select the routing sets and server group that you want.

Figure 18-4 shows the Specify the routing rules window with two routing rules defined. As soon as a routing rule is saved, the Content Manager is updated immediately, and the rule takes effect for new requests.

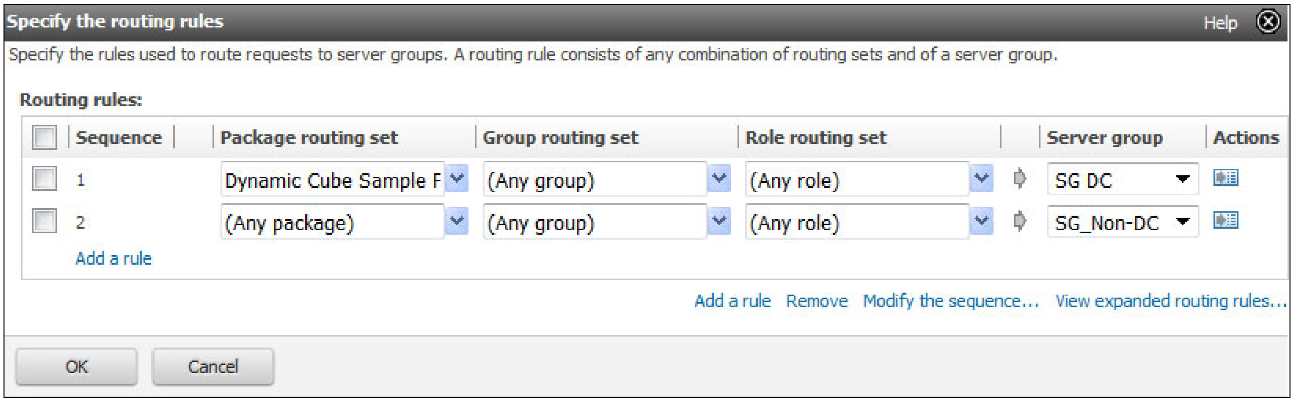


Figure 18-4 Specify the routing rules page with two routing rules defined

The first rule in Figure 18-4 routes a request to server group SG DC if the report package has a routing set of Dynamic Cube Sample Packages, regardless of user group or role. The second rule is a catch-all rule that routes all other requests to server group SG_Non-DC. To see the underlying packages, groups, and roles of a routing set, and which dispatchers belong to the server group, select **View expanded routing rules** on the Specify Routing Rules window. The expanded routing rules view is shown in Figure 18-5.

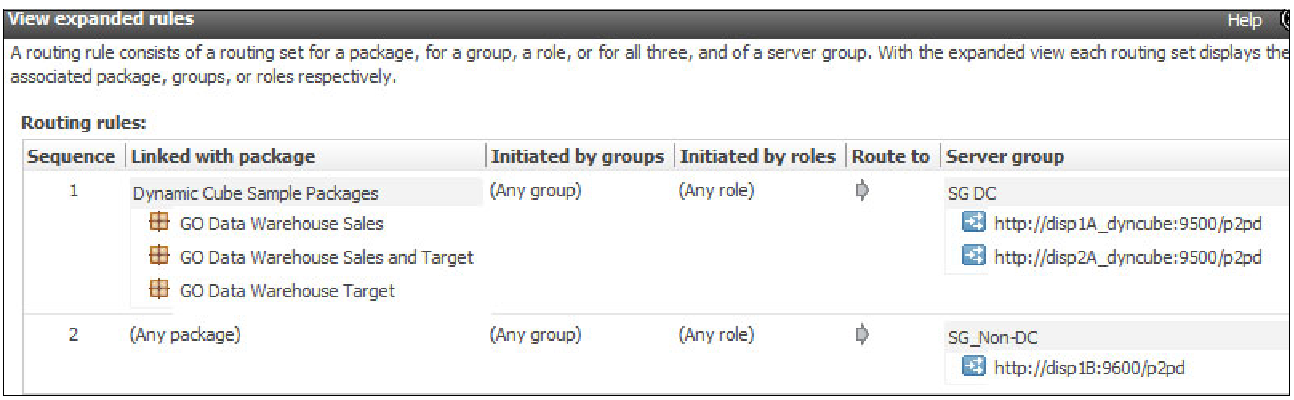


Figure 18-5 View expanded rules window

Only explicitly defined server groups are included in the server group picklist on the Specify routing rules window. The system does not allow the built-in `DefaultServerGroup` to be used in a routing rule. The `DefaultServerGroup` is a logical container for all dispatchers that are not explicitly placed in a user-defined server group.

> **Tip:** Design routing rules such that there is a matching rule for every report request. Every report request should be tagged with an explicit server group. When there is not a matching rule, the system can route the request to any dispatcher in the system running a service of the required type, including dispatchers in the `DefaultServerGroup`. If a request is delivered to a dispatcher that is not equipped with the required data source or is not running a compatible report server execution mode, the request fails. It might be useful to define a *catch-all* rule last in the list, which maps `(Any package)`, `(Any group)`, and `(Any role)` to a specific server group.

The following are the general steps for deploying a dynamic cube in a multi-dispatcher environment:

1. Create server groups.
2. Add the dynamic cubes to server groups. This creates a separate autonomous instance of each dynamic cube on every dispatcher in the server group.
3. Configure and start all instances of the cube.
4. Set up advanced routing by specifying routing rules that map report requests associated with a package, user group, and user role to a specific server group.

Use Case: Introducing dynamic cubes into an existing installation

This section discusses a common use case that describes a scenario for the introduction of dynamic cubes into an existing IBM Cognos BI system.

Scenario

The following are the main characteristics of this scenario:

- The system has a dedicated load-balancing dispatcher on server 1, with the report services disabled.
- There is an Application Tier Components installation on server 2 and server 3, each with the report server execution mode set to 32-bit.
- The system has pre-existing compatible query mode (CQM) and dynamic query mode (DQM) data sources.
- The Content Manager is installed on server 4.
- No routing rules are defined.

  When packages are opened and reporting or analysis is performed, the requests are distributed across all dispatchers in the system that are running a report service (that is servers 2 and 3) for load balancing.
- You want to introduce dynamic cubes.

  Initially you will use the default 32-bit report server execution mode, but in the future you can change the report server execution mode to 64-bit.

  To be prepared for a possible change of the report server execution mode to 64-bit, and to restrict the traffic on dynamic cube dispatchers to dynamic cube requests only, use dedicated dynamic cube servers and set up explicit routing rules.

Approach

The following points summarize the solution approach:

- Two new Application Tier Components are installed on servers 5 and 6, for exclusive use by dynamic cubes.
- The two new dispatchers are assigned to a server group named DynamicCube Server Group.
- Dynamic cubes are added to the server group, and a package-based routing rule is defined so that dynamic cube requests only route to server group DynamicCube Server Group.

  At this point, with only one routing rule, non-dynamic cube requests are also routed to the dynamic cube dispatcher for load balancing.

  Non-dynamic cube requests are open to be dispatched to any dispatcher running a report service because there are no specific routing rules for those requests.

- The non-dynamic cube CQM requests will run successfully on the new dispatchers if the report server execution mode is not changed to 64-bit, and if the required client software is installed on the servers.
- A catch-all routing rule is defined last in the list to route (`Any package`), (`Any role`), and (`Any group`) to a new server group called Non-DynCube Server Group.

  This routing rule is defined because the report server execution mode can change to 64 bit in the future, and to prevent non-dynamic cube traffic from loading dynamic cube dispatchers.

The resulting routing rules are as shown in Figure 18-6.

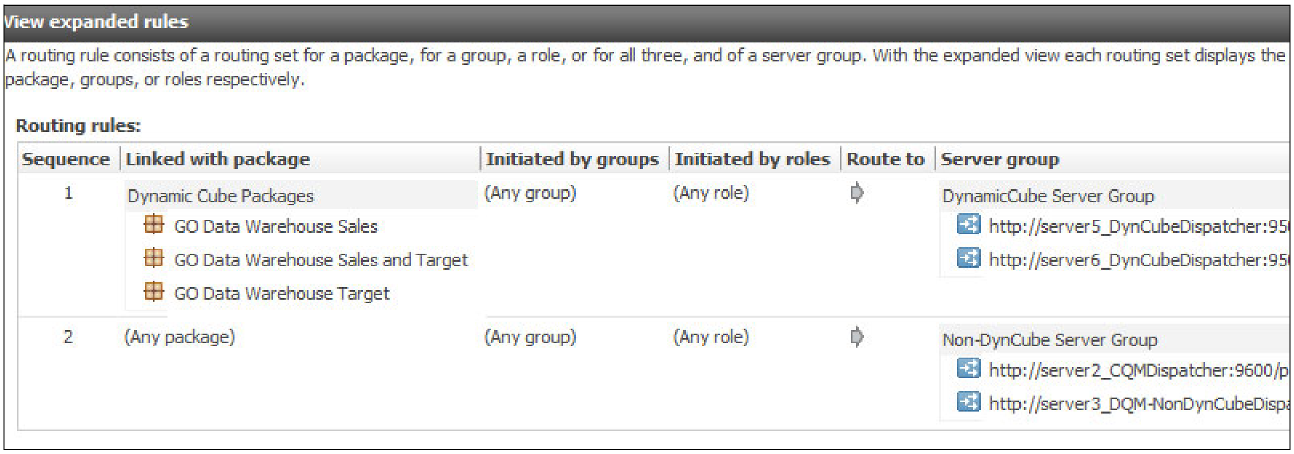


Figure 18-6 Expanded view of routing rules

18.1.3 Load balancing

Load balancing is the distribution of report requests across report services on dispatchers in the same server group. In the absence of user-defined routing rules, or when there are routing rules but a request does not match any rule, such requests are load balanced across all dispatchers in the system that are running a report service.

The load balancing mode is a dispatcher property. The default is *weighted round-robin*, where the dispatcher processing capacity is taken into account.

> **Note:** The load balancing mode should be identical on all dispatchers in a server group.

The header of all requests includes a *request affinity* value, where the affinity indicates where a request can, should, or must be routed. Reporting and analysis involve related sequences of requests and responses called *conversations*. The initial request *can* be load balanced. Most subsequent interactions, such as HTML paging, should be routed to the same dispatcher for performance. Some subsequent interactions, such as cancel, *must* be routed to a specific dispatcher. In the absence of a fault condition, only the initial request in a conversation is load balanced. Requests associated with an analysis session in IBM Cognos Workspace Advanced are typically routed to the same report service.

> **Note:** All dispatchers in a server group must have the same set of dynamic cubes and all instances of a cube must be available. If this is not true, a request for a cube might fail (with a message like `The dynamic cube “XXX” either does not exist or has not been added to a query service instance`).

18.1.4 Failover

All dispatchers and services are registered with Content Manager. When dispatchers and services go in and out of service, this information eventually is propagated to all dispatchers in the system so that requests are routed to dispatchers with available services.

> **Note:** It can take up to one minute for a dispatcher or service state change to completely propagate through the system.

Failover is the rerouting of report requests to an available instance of the report service in the same server group when an instance of the report service becomes unavailable. If a report request is destined for a server group, and no report service is available in that server group, the request fails.

Failover of report requests is triggered by the availability of the report service. If report services fail on one dispatcher, report requests that are normally load balanced to the failed service are rerouted to a report service on another dispatcher in the same server group.

> **Note:** Dispatcher routing does not take the availability of the query service into account. If the query service becomes unavailable, report requests continue to be routed to the query service and fail.
>
> Similarly, dispatcher routing does not take the availability of a dynamic cube into account. If a dynamic cube in an active server group becomes unavailable, requests continue to be routed to the cube and fail.
>
> In the case where the query service is available but running low on memory, the query service might enter an overloaded state. When in an overloaded state, the query service rejects new incoming report requests with a *server busy* error. The dispatcher responds by rerouting the request to another report service on another dispatcher in the same server group if one is available.

18.2 Deploying and managing cubes in a multi-dispatcher environment

This section describes how to perform various dynamic cube administrative procedures in a multi-dispatcher environment such that there are no report request failures.

Consider the following key points:

- Every dispatcher in a server group must have the same set of cubes. Otherwise, the report requests will fail. Before a dispatcher is added to an active server group, ensure that the same set of cubes is available on it.
- Cube instances are autonomous. Each cube instance has separate properties. In general, within a server group configure each instance of a cube the same, including cache sizes. Note, however, that the Automatic optimization of in-memory aggregates property should only be enabled on one instance of a cube.
- The query service should be configured the same on each dispatcher in a server group.
- Dispatchers are not aware of the existence of dynamic cubes. To prevent requests from being dispatched to an unavailable cube, before stopping a dynamic cube, reassign the dispatcher to an inactive server group.

18.2.1 Bringing a new dynamic cube in-service

This procedure brings a new dynamic cube in-service on a server group with multiple dispatchers, such that report requests are properly dispatched. The cube is for the exclusive use of members of the Sales Exec role.

This procedure involves the following steps:

1. Assign two dispatchers `DispA` and `DispB` to server group `SG Sales Exec`.
2. Configure the query service (for example, the heap size). Restart the query service to activate the properties.
3. Add the dynamic cube to server group `SG Sales Exec`.
4. Configure cube properties on `DispA` and `DispB`. Set the same property values on `DispA` and `DispB`.
5. Start the cube on `DispA` and `DispB`. Wait until both cube instances are available, and in-memory aggregates are loaded.
6. Assign routing set `Sales Exec Package` to the dynamic cube package.
7. Assign routing set `Sales Exec Role` to the `Sales Exec` role.
8. Specify a routing rule, for example:

   ```
   Package routing set = Sales Exec Package
   Group routing set = (Any group)
   Role routing set = Sales Exec Role
   Server group = SG Sales Exec
   ```

The cube is now ready for use by members of the `Sales Exec` role.

18.2.2 Adding a dispatcher to an existing active server group

This procedure adds a dispatcher to an existing server group that is actively processing dynamic cube requests, such that report requests are properly dispatched and do not fail.

Server group `SG1` currently has a single dispatcher, `DispA`. Dynamic cubes `CubeA` and `CubeB` are available and processing requests on `DispA`. The goal is to add another dispatcher, `DispB`, to `SG1` for load balancing. To avoid report request failures, as soon as `DispB` is added to `SG1`, it must be ready to process requests.

The system has routing rules in place and all requests are subject to a routing rule. Therefore, a new dispatcher can be installed and started and no requests are routed to it until it is explicitly included in a routing rule.

This procedure involves the following steps:

1. Add the new dispatcher `DispB` to an inactive server group `SG_INACTIVE`. This server group is not used in any routing rules.
2. Configure query service properties on `DispB`. They should match `DispA`. If required, restart the query service to activate properties.
3. Add all cubes currently deployed in `SG1` to `SG_INACTIVE`.
4. Set properties for each cube in `SG_INACTIVE`. In general, the property values should match the cubes on `DispA`.

5. Start all cubes in `SG_INACTIVE`. Wait for all cubes to become available, and for in-memory aggregates to load.
6. Reassign DispB to SG1.

The dispatcher becomes part of SG1 and the cubes remain available. New incoming report requests are load balanced across DispA and DispB.

18.2.3 Removing a dispatcher from an active server group

This procedure removes a dispatcher from an existing server group that is actively processing dynamic cube requests, such that report requests are properly dispatched and do not fail.

A server group `SG1` has two dispatchers: `DispA` and `DispB`. Dynamic cubes are available on both dispatchers.

This procedure involves the following steps:

1. Reassign `DispB` to an inactive server group `SG_INACTIVE` that is not used in any routing rules.
2. Wait one minute for other dispatchers to become aware of the reassignment so that new requests are no longer dispatched to `DispB`.
3. Stop the cube by running the **`Stop after active requests complete`** command after active requests complete on the cubes in `SG_INACTIVE`.
4. When all cubes are in the `Unavailable` state, remove the cube from the server group by running the **`Remove data store from server group`** command.

`DispB` is ready to be redeployed to another server group or decommissioned.

18.2.4 Moving a cube from one active server group to another

This procedure moves a cube from one server group to another, with no interruption to reporting against the cube.

The goal is to reshape the query load by moving `CubeA` from server group `SG1` to server group `SG2`. The dispatchers and services in `SG2` are already configured to handle the additional load from `CubeA`. `SG1` and `SG2` each have two dispatchers.

This procedure involves the following steps. Do not disturb SG1 because it is actively processing requests.

1. Add `CubeA` to `SG2`. Configure and start both cube instances in `SG2`. Wait until both instances are available, and in-memory aggregates are loaded.
2. Edit any routing rules associated with `CubeA` and change the server group from `SG1` to `SG2`. New requests for the cube should now route to `SG2`.
3. Stop the cube by running the **`Stop after active requests complete`** command after active tasks complete on `CubeA` in `SG1`.
4. When the cube state changes to unavailable in `SG1`, remove the cube from the server group by running the **`Remove data store from server group`** command.

18.2.5 Near real-time incremental update

Incremental update applies new fact rows to the data cache and aggregate cache without stopping or pausing the cube. When the incremental update completes, all new queries are directed to the updated caches.

Consider a server group where queries are load balanced across multiple instances of a cube. The incremental update command is issued per dispatcher (per cube instance). Separate requests can be issued to all dispatchers in a server group at roughly the same time. However, the requests do not complete at the same time and there is a period of cache state inconsistency across dispatchers.

During the period of cache state inconsistency, requests routed to different servers can return different fact data during the period where each cube instance is at a different active transaction ID (TID). Alternate report executions for a given interactive user can return different results.

To minimize the period of cache state inconsistency across dispatchers, follow these tips:

- Issue the incremental update command simultaneously to all dispatchers.
- Process smaller increments to reduce the variation in completion time.
- Use equally sized servers with equal load.

If a period of cache state inconsistency cannot be tolerated, use one of the following techniques:

- Perform the incremental update in a maintenance window.
- Apply the pause command to all cube instances before running the update.
- Remove all but one dispatcher from the server group, leaving one dispatcher to service requests while the other dispatchers are updated. Then, add the updated dispatchers back into the server group while removing and updating the remaining dispatcher.

18.2.6 Cache refresh

Each cube instance has an autonomous set of caches. Starting a cache refresh on a server group will not complete at exactly the same time on each instance of a cube because the operations are independent. If there are active reporting sessions, a user can potentially see different member and fact data because report requests are load balanced during the period when the caches are out of sync.

In general, to ensure cache consistency across instances of a cube within a server group, perform the following operations in a maintenance window:

- Refresh member cache, if the underlying dimension tables have changed.
- Refresh security settings, if the published security views, database security lookup tables, or security view access permissions have changed.

The content of a data cache depends on the requests processed by a cube instance. With load balancing, a workload is spread across cube instances, which results in some degree of difference in the content of the data cache.

18.3 Database impacts in a multi-dispatcher environment

When there are multiple dispatchers per server group, each cube in the server group has one instance per dispatcher. Each cube instance is autonomous, with its own set of caches. Starting multiple instances of a cube concurrently places an increased load on the underlying relational database proportional to the number of cube instances in the server group.

For each cube instance on start, there are database queries to populate the member cache, and after cube start, database queries to populate the aggregate cache. In addition, there might be data cache priming queries. The increased concurrent load must be taken into account during database performance planning.

To reduce the concurrent database load, use a staged approach where one cube or set of cubes on one dispatcher is started at a time.

18.4 Aggregate Advisor in a multi-dispatcher environment

This section provides guidance on working with the Aggregate Advisor when a cube is deployed to a server group with multiple dispatchers.

18.4.1 Installation

The Aggregate Advisor is part of the IBM Cognos Dynamic Query Analyzer (DQA) installation. IBM Cognos Configuration is used to configure a gateway URI and a dispatcher URI for the external applications property.

The gateway URI is used by DQA to run reports to generate query logs for profiling. It is not used by the Aggregate Advisor. Aggregate Advisor is dependent solely on the dispatcher URI for external applications property, for accessing workload information, running Aggregate Advisor queries, saving and fetching results, and applying and clearing recommendations.

The workload log used by the Aggregate Advisor, and the results that are displayed by the Aggregate Advisor, depend on which dispatcher the Aggregate Advisor connects to because the workload log and results are hard disk drive files within the configured dispatchers' Application Tier Components installation.

There might be a need to connect to different dispatchers to access different workload logs (and view different results). Although it is possible to install DQA in the same folder as a Cognos BI server, it is best to install it in a separate location so that DQA can be configured independent of the Cognos Configuration of the Cognos BI server. If co-located with the Cognos BI server, the configuration is shared, and the dispatcher URI for external applications cannot be changed without affecting the Cognos BI server.

Tip: Install DQA in a dedicated folder so that if required, you can freely change the dispatcher URI for external applications to point at a different dispatcher.

18.4.2 Workload logging

Workload logging is enabled per cube instance. A cube that is deployed in a server group with multiple dispatchers has a separate cube instance per dispatcher. As queries are load balanced across dispatchers, the number of queries processed by each instance varies and the workload log content is different. If the query workload for a cube is run several times, the content of the workload log across cube instances converges, although differences might remain.

Tip: Aggregate Advisor recommendations are highly dependent on the content of the workload log. To ensure that all queries for a workload are captured in a single workload log file, capture the workload logs while the cube is deployed to a server group with a single dispatcher only. This approach can be done on a single-dispatcher staging system before moving into a multi-dispatcher production environment.

If the workload log can only be captured while the cube is deployed with multiple instances, enable workload logging on all instances, then manually merge the individual workload log files into a single file on one dispatcher before running the Aggregate Advisor.

The workload log is essentially a trace file without consolidation of duplicate queries. The file is structured with an opening tag `<olap:aggregateWorkloads>`, followed by `<olap:report>...</olap:report>` entries for each query. By design, there is not a closing `</olap:aggregateWorkloads>` tag. The file grows as more `<olap:report>...</olap:report>` sections are added. Two separate workload logs from different instances of the same cube can be merged by copying all the `<olap:report>...</olap:report>` content from one file and pasting it to the end of the other file.

18.4.3 Aggregate Advisor results

Aggregate Advisor results (recommendations) are saved as files on the file system of the dispatcher that the Aggregate Advisor is connected to at the time it runs. Results created while connected to dispatcher A are not shown in the Aggregate Advisor results window if the configured dispatcher is changed to dispatcher B. The Aggregate Advisor only shows the results from the file system of the dispatcher it is configured against.

Tip: To avoid having result files on different Application Tier Components installations and having to track where advisor results are located, always run the advisor against the same dispatcher.

18.4.4 Saving and clearing in-memory aggregates

There is a single set of in-memory aggregate definitions per cube. Saving and clearing aggregate recommendations is applied to the content store. Therefore, saving new or clearing existing aggregates affects all instances of a cube the next time that the cube instance starts.

Saving and clearing aggregates is independent of the dispatcher configured for DQA because the aggregate definitions are in the content store.

18.4.5 Connecting to a different dispatcher

To connect the Aggregate Advisor to a different dispatcher, complete these steps:

1. Close DQA.
2. Start DQA Cognos Configuration.
3. Change the Dispatcher URI for external applications and save the configuration.

Relaunch DQA, and run the Aggregate Advisor.

18.4.6 Automatic optimization of in-memory aggregates

The automatic optimization of in-memory aggregates feature runs and selects aggregates separately per cube instance. There is currently no coordination among cube instances.

Enabling automatic optimization of in-memory aggregates on multiple instances of the same cube can result in independent and inconsistent recommendations depending on the queries processed by the cube instance. Given that the dispatcher balances the query workload across cube instances, each cube instance has a different workload log and automatic optimization of in-memory aggregates can potentially produce a different set of recommended aggregates per instance.

With a single set of aggregate definitions in the content store shared by all instances of a cube, running automatic optimization of in-memory aggregates on more than one instance of a cube results in each instance changing the aggregate definitions saved by the other instance. In addition, running automatic optimization of in-memory aggregates on more than one instance of a cube adds more burden on the system.

> **Note:** Given the independent nature of cube instances, only enable the automatic optimization of in-memory aggregates feature on one instance per cube. Periodically synchronize the aggregates in other cube instances by restarting the cube.

The autonomic optimization of in-memory aggregates feature is described in 11.4.5, "Automatic optimization of in-memory aggregates" on page 375.

18.5 Cube Designer in a multi-dispatcher environment

The Cognos Cube Designer application interacts with the *configured* dispatcher, which is whichever dispatcher is specified by the Dispatcher URI for the external applications property in Cognos Configuration. During the publishing process, there is an option to *Add* the dynamic cube to a dispatcher. If this option is selected, the cube is added to the configured dispatcher.

The Cognos Cube Designer application is not aware of server groups. If the configured dispatcher is in a server group with multiple dispatchers, during the publishing process the cube is only added to the configured dispatcher. Later, before reporting, the cube must be manually added to all dispatchers in the server group because otherwise requests load balanced to dispatchers in the server group that do not have the cube fails.

> **Tip:** The cube development phase of a project is typically an iterative process that involves modeling, deployment, and reporting. During cube development, set up a server group that contains one dispatcher only, with support for dynamic query mode, and configure Cognos Cube Designer to use that dispatcher. This approach avoids the need to manually add the cube to other dispatchers (in the server group) to be able to do reporting for testing purposes.

> **Note:** The dispatcher configured for cube designer must be running the dynamic query mode services. A typical load balancing dispatcher with dynamic query-related services disabled is not suitable as the Cognos Cube Designer dispatcher.

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this book.

IBM Redbooks

The following publications provide additional information about the topic in this document:

- *IBM Cognos Business Intelligence V10.1 Handbook*, SG24-7912
- *IBM Cognos Dynamic Query*, SG24-8121

You can search for, view, download or order this document and other Redbooks, Redpapers, Web Docs, draft and additional materials, at the following website:

ibm.com/redbooks

Online resources

These websites are also relevant as further information sources:

- Business Intelligence 10.2.2 documentation

  http://ibm.co/1zZb7hP

- Dynamic Cubes Installation and Configuration Guide 10.2.2

  http://ibm.co/1FdEVI0

- Dynamic Cubes User Guide 10.2.2

  http://ibm.co/1QIQVoG

- Dynamic Query Analyzer Installation and Configuration Guide 10.2.2

  http://ibm.co/1bOVzBU

- Dynamic Query Analyzer User Guide 10.2.2

  http://ibm.co/1H37JQm

- Framework Manager User Guide 10.2.2

  http://ibm.co/1IAFpsD

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Redbooks

IBM Cognos Dynamic Cubes

SG24-8064-01

ISBN 0738440833

(1.0” spine)
0.875”<->1.498”
460 <-> 788 pages

SG24-8064-01

ISBN 0738440833

Printed in U.S.A.