Information Management software

IBM

Multidimensional Analytics: Delivered with InfoSphere Warehouse Cubing Services

- Getting more information from your data warehousing environment
- Multidimensional analytics for improved decision making
- Efficient decisions with no copy analytics

Chuck Ballard
Silvio Ferrari
Robert Frankus
Sascha Laudien
Andy Perkins
Philip Wittann

Redbooks

ibm.com/redbooks

International Technical Support Organization

Multidimensional Analytics: Delivered with InfoSphere Warehouse Cubing Services

April 2009

SG24-7679-00

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

First Edition (April 2009)

This edition applies to IBM InfoSphere Warehouse Cubing Services, Version 9.5.2 and IBM Cognos Cubing Services 8.4.

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 give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.

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

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

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

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

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:

The following terms are trademarks of other companies:

Cognos, and the Cognos logo are trademarks or registered trademarks of Cognos Incorporated, an IBM Company, in the United States and/or other countries.

NOW, and the NetApp logo are trademarks or registered trademarks of NetApp, Inc. in the U.S. and other countries.

Oracle, JD Edwards, PeopleSoft, Siebel, and TopLink are registered trademarks of Oracle Corporation and/or its affiliates.

SAP, and SAP logos are trademarks or registered trademarks of SAP AG in Germany and in several other countries.

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

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

Balanced Warehouse™
ClearCase®
Cognos®
DataStage®
DB2®
IBM®
InfoSphere™
Rational®
Redbooks®
Redbooks (logo) ®
RequisitePro®
System z®
WebSphere®

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

Preface

To be successful and grow in today's business environment you need an advantage. Most often that advantage is found in the form of more current information. That current information, which goes by the term business intelligence, is becoming more complex to analyze because it is defined by multiple dimensions rather than the two dimensions typically used in the past. In most situations, simply using analyses based only on two dimensions no longer provides that required advantage.

This IBM® Redbooks® publication is the third of a series of IBM Redbooks publications that comprise what we call the *multidimensional trilogy*. In this series of books, we describe how to develop a multidimensional environment and analytics capabilities from start to finish.

IBM provides end-to-end technologies, services, and solutions for generating Business Intelligence, which includes information integration, master data management, data warehousing, industry models, BI tools, and industry frameworks. Supporting all of these capabilities requires a robust data warehousing infrastructure, which IBM also provides in the form of the InfoSphere™ Warehouse offering.

InfoSphere Warehouse enables a unified, powerful data warehousing environment. It supports structured and unstructured information and operational and transactional data, in real-time. However, our primary objective in this book is to discuss and describe how InfoSphere Warehouse Cubing Services can help you develop and use the required multidimensional analysis and how you can use the functions and features of InfoSphere Warehouse to solve those business problems. In particular, we discuss how you can use InfoSphere Warehouse Cubing Services (IWCS) to develop those solutions. For detailed information about InfoSphere Warehouse and the supported client interfaces, reference the Redbooks publication *InfoSphere Warehouse: Cubing Services and Client Access Interfaces*, SG24-7582.

IWCS provides a multidimensional view of data, which is stored in a relational database, to dramatically improve the performance of online analytical processing (OLAP) queries. Using IWCS, you can create, edit, import, export, and deploy cube models over the relational warehouse schema. It also provides optimization techniques to dramatically improve the performance of OLAP queries, which is a core component of data warehousing and analytics, and all occurs without copying the data from the relational databases to another store. We call that *No Copy Analytics*.

To do this, IWCS uses dimensional models. These models make it easier to pose business questions that are related to a particular business process or business area. You can analyze these business questions from multiple perspectives by selecting an appropriate set of criteria. When the data is organized in dimension hierarchies, the structure is intuitively more familiar to you and facilitates your understanding of the relationships among data categories.

Cubing Services works with business intelligence (BI) tools, such as Alphablox, Cognos®, and Microsoft® Excel®, through the client access interface to accelerate OLAP queries from many data sources. It works by using cube metadata to design specialized summary tables (DB2® materialized query tables or MQTs) that contain critical dimensions and levels, or slices, of the cube. The Cubing Services cube optimization process recommends summary tables that you can create to improve the performance of OLAP queries. The DB2 optimizer transparently rewrites incoming queries and routes eligible queries to the appropriate summary tables for significantly faster query performance. These Cubing Services recommend summary tables that can accelerate all SQL-based queries into the data warehouse, not just those queries submitted on behalf of Cubing Services.

The team that wrote this book

This book was produced by a team of specialists from around the world working with the International Technical Support Organization, in San Jose California. In this section, we list the team with a short biographical sketch of each.

Chuck Ballard is a Project Manager at the International Technical Support organization, in San Jose, California. He has over 35 years of experience, holding positions in the areas of Product Engineering, Sales, Marketing, Technical Support, and Management. His expertise is in the areas of database, data management, data warehousing, business intelligence, and process re-engineering. He has written extensively on these subjects, taught classes, and presented at conferences and seminars worldwide. Chuck has both a Bachelors degree and a Masters degree in Industrial Engineering from Purdue University.

Silvio Ferrari is a senior IT specialist in the IBM Sales and Distribution Organization in Sao Paulo, Brazil. For the past eight years he has worked in the IBM Information Management area as a Business Intelligence consultant, providing support for customers who are planning and implementing data warehousing environments and OLAP systems.

Robert Frankus is a Consultant IT Specialist for the BI Best Practices Team focused on Analytics. He is based in San Francisco, CA. His areas of expertise are in analytics, data warehousing, and information integration. Over the last seven years, he architected and developed a number of analytical applications, dashboards, and corporate performance management solutions for Fortune 500 clients, in industries, such as retail, high tech, and financial services. He teaches extensively on dimensional modeling and building business intelligence applications. He has a Masters of Management Information Systems from the University of Cologne, Germany and was a Visiting Fellow at the Massachusetts Institute of Technology Sloan School of Management in Boston, Massachusetts.

Sascha Laudien works as an IT Specialist for the IBM Software Group, Germany. His areas of expertise are in administration of DB2 databases, data warehousing, information integration, and analytical application development. As a part of the IBM Information Management services team he currently supports customer projects in Germany that are based on a wide variety of Data Warehouse and Business Intelligence topics. He has a Bachelors degree in Computer Science and is an IBM Certified Associate and an IBM Certified Solution Designer.

Andy Perkins is a Data Warehousing Specialist for the IBM Silicon Valley Lab Data Warehouse on System z SWAT team, focusing on InfoSphere Warehouse on System z. He lives in Dallas, Texas and has over 26 years of experience with customer data warehouse and BI projects, providing support in consulting, architecture design, data modeling, and implementation of data warehouses and analytical solutions. Over the years Andy worked with the majority of the IBM Information Management products portfolio on platforms ranging from the PC to the System z.

Philip Wittann is an Advisory Software Engineer for Cognos Business Intelligence Reporting with IBM, Australia. He has worked with business intelligence reporting for seven years and through this time created and identified some of the key approaches to successful implementation of relational and dimensional reporting applications with IBM Cognos products. He also taught courses on data design, report design, metadata modeling, and understanding queries from both SQL and MDX query perspectives across multiple relational and OLAP providers. Phil has a degree in Physics from the University of Waterloo, Canada.

Other Contributors:

In this section, we thank others who either contributed directly to the content of the book or to its development and publication.

For their significant contributions and ongoing support in the development of this Redbooks publication, we give special thanks to:

Skyla Loomis	Manager of InfoSphere Warehouse Cubing Services
David Wilhite	InfoSphere Warehouse Product Manager

Thanks also to the following people for their contributions to this project, from IBM locations worldwide:

Vagner Bettarelli	Software Sales Manager, Information Management, Cognos Solution, Sao Paulo, Brazil.
Nathan Gevaerd Colossi	Senior IT Specialist, Software Sales, Sao Paulo, Brazil.
Daniel DeKimpe	Software Developer, Cubing Services Development, Silicon Valley Lab, San Jose, CA.
Robert Kickhofel	Technical Support Professional, Client Resolution Specialty, Silicon Valley Lab, San Jose, CA.
Kevin Low	Software Developer, InfoSphere Warehouse Cubing Services, Silicon Valley Lab, San Jose, CA.
Jose Roberto de Oliveira Neto	Information Management IT Specialist, Software Sales, Sao Paulo, Brazil.
Warren Pettit	Information Management Instructor/Developer for Data Warehouse courses, Menlo Park, CA.
John Poelman	Information Developer, InfoSphere Warehouse Cubing Services, Silicon Valley Lab, San Jose, CA.

Velimir Radanovic Software Developer, InfoSphere Warehouse Cubing Services, Silicon Valley Lab, San Jose, CA.

Cheung-Yuk Wu Software Developer, Common AD Tooling, Silicon Valley Lab, San Jose, CA.

From the International Technical Support Organization, San Jose Center, we thank:

Mary Comianos Publications Management
Deanna Polm Residency Administration
Emma Jacobs Graphics
KaTrina Love Abram Editor

Become a published author

Join us for a two- to six-week residency program! Help write a book dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You will have the opportunity to team with IBM technical professionals, Business Partners, and Clients.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you will develop a network of contacts in IBM development labs, and increase your productivity and marketability.

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® in one of the following ways:

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

  ibm.com/redbooks

- Send your comments in an e-mail 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

1

Introduction

Our objective in this IBM Redbooks publication is to illustrate how to implement the various multidimensional constructs using the Cubing Services component of InfoSphere Warehouse. Cubing Services provides the ability to analyze warehouse data at the speed of thought by using OnLine Analytical Processing (OLAP) techniques.

In this book, we describe how to effectively use Cubing Services to deliver these OLAP style analytics. To do this, first requires a solid dimensional data model for the base relational data in addition to the Cubing Services product knowledge. To help you define, develop, and deliver these analytics capabilities, we created a series of three books. This is the third book in that series, that we call *the multidimensional trilogy.*

The first Redbooks publication in the trilogy, *Dimensional Modeling: In a Business Intelligence Environment*, SG24-7138, describes and demonstrates how to develop a multidimensional relational database using a star-schema model. In this book, the authors describe how to start with an E/R database model and create from it a dimensional relational database model. It illustrates how to solve many of the challenges inherent in creating a dimensional relational database.

The second Redbooks publication in the trilogy, *InfoSphere Warehouse: Cubing Services and Client Access Interfaces*, SG24-7582, has more of a product-level focus. In this book, the authors describe the details of using the product and include information and examples of how to access the data by using Cognos BI

and Microsoft Excel. In addition, it includes a detailed and informative chapter about the MDX language.

This publication completes the trilogy by starting with the relational multidimensional concepts presented in the Dimensional Modeling book and showing how they are implemented in Cubing Services. To best make use of this book, use it in conjunction with the two Redbooks publications that we previously mentioned. We do not cover the relational modeling concepts that were presented in the Dimensional Modeling book nor do we cover the Cubing Services product usage that is in the Cubing Services book. In this book, we assume that you are familiar with the other two Redbooks publications that are in the trilogy or that you can access them for reference. To get those two books, go to the ITSO Web site and download them for free:

http://www.ibm.com/redbooks

1.1 Multidimensional Business Intelligence: The Destination

Delivering the right information to the right people at the right time is the ultimate goal of any data warehousing environment. Exactly what this means and how the information is delivered is dependent on the business requirement. Perhaps the requirement can be met with reporting functions, such as a simple listing of a customer's recent orders on a Web page; however, these days users typically require much more in terms of data analysis capability than simple reports of operational data.

In today's environment, there is an ever-increasing need for more information to be presented to users, such as charts and graphs, in an easy to understand manner. Database and reporting technologies continuously evolve to meet these needs. The development of relational databases was a great leap in providing easier data access around which new and improved reporting tools grew. In addition, reporting tools added increasingly more sophisticated functions over the years. However, there is still a limit as to what can be accomplished with a typical relational database design. So this puts the responsibility to handle more sophisticated data analysis on all new reporting tools.

Another trend in data access and analysis is in the direction of enabling interaction with the data. By this, we mean that users want to be able to look at data from multiple perspectives, drill down for greater detail, drill up for less detail, change the time range of the data, and dynamically add and remove specific measurements or descriptive information. This style of analytics is a more refined description of what we previously referred to as OLAP, and it is

what enables data analysis from a multidimensional perspective. The business information and knowledge gained from this type of analytics is called *Multidimensional Business Intelligence*.

A business analyst might look at weekly sales for a particular store type for a particular state over the last three months, and then based on some clue in the data, the business analyst wants to see sales by product line for that state over the same three months or see the detail for each of the stores in that store type. The goal is to be able to allow analysts to quickly change the view of the data before losing their train of thought, which is also called *analysis at the speed of thought*. Another characteristic of the OLAP style of analytics is that analysis is against aggregated data, at various levels, looking at trends over time or caparisons of different time periods. But still, there is always the need to occasionally drill down into more detailed data.

So, what is required to be able to deliver this interactive information capability? To accomplish this, you must establish a data warehousing environment on which to store the historical data. And the data in the data warehouse must be organized in such a manner that it can be quickly and easily retrieved. Finally there must be some tooling that understands OLAP style analytics and can make the translation from relational data and deliver aggregated data to the user in an interactive manner with reasonable response times.

Having all of these new types of data organization and delivery is what is needed to enable the required multidimensional business intelligence in this fast-paced business environment on which we find ourselves today. *This is . . . the Destination!*

1.1.1 Dimensional model

The OLAP style of reporting is typically performed using a specific type of data organization called multidimensional modeling. Multidimensional modeling organizes the data into facts, also called measures and dimensions. Facts are the data values that you want to analyze and the dimensions contain the values of how you want to view the data. In this example, sales amount is a fact that we want to measure, while store, region, and time are the dimensions or the way in which we want to see the data presented, which is called viewing sales by, store by geography, and by time. When this type of organization is drawn in a diagram, it resembles a star, as you can see in Figure 1-1 on page 4, which is where the term star-schema is derived. There is likely one star schema for each business subject area.

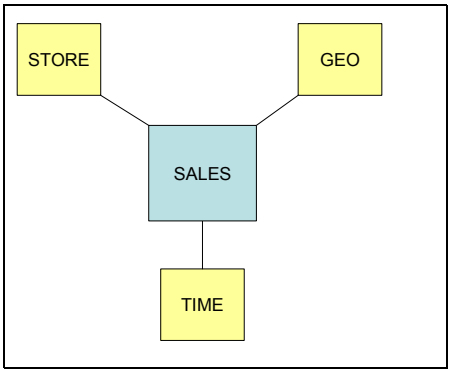


Figure 1-1 A simple star-schema

In Figure 1-1, all of the values from where we want to derive analytical information are contained in a relational table called SALES, which is called the fact table. All of the other tables, such as STORE, GEO, and TIME, contain descriptive information, which is how you can view the data. These tables are the dimensions of the star. The data in the fact table refers to the data in a dimension table using the primary key of each dimension table. Fact tables are many times larger than dimension tables. In a star-schema design, the data is denormalized, which results in fewer tables that must be joined to satisfy a data request.

One of the most important characteristics of a star-schema design is simplicity, as compared to a traditional database design that is used to support transactional systems. In a transactional system, the database is designed to provide optimal performance involving discrete amounts of data and also for fast updates, which typically results in many more tables with complicated relationships. This type of design is not optimal for query or analysis requirements. The star-schema design, on the other hand, is a simpler design. It is fairly easy to determine what tables are needed for a request. In the example, if a requirement is to see the sales for a store by month over the last thirteen months, it is a simple translation to see that the STORE and TIME dimensions are used to qualify the query and are joined with the SALES fact table.

In addition to the relationship between fact and dimensions, there are data relationships present within a dimension. Some of the attributes within a dimension represent the way in which data is summarized or aggregated, and they are organized in a hierarchy. Figure 1-2 on page 5 shows a typical time hierarchy.

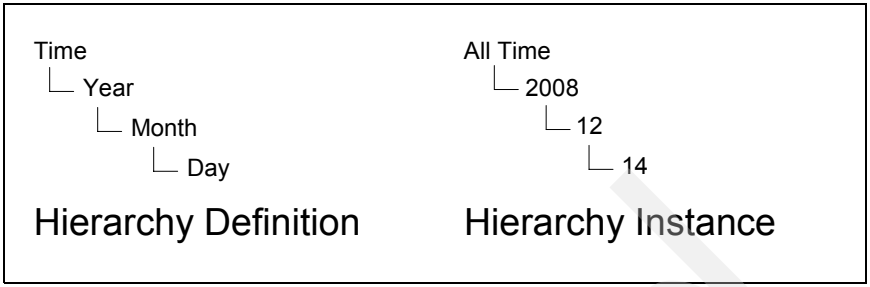


Figure 1-2 A simple time hierarchy

The levels of the hierarchy are where aggregations of some sort typically occur; the most common is a simple addition. In the time hierarchy:

- All of the day values for a month are added up to become the value for the month
- All of the month values for a year are added up to become the value for the year
- All of the year values are added to become the one value represented at the top of the hierarchy

The hierarchy seems simple until you consider that a dimension is not used in isolation but in conjunction with other dimensions. A report showing sales by month for year 2007 for all stores and all geographies might not be as useful as seeing the sales by month for the stores in San Jose, particularly if you are the city manager for the stores in San Jose; therefore, you might need to return a value for any combination of any value of any dimension. Although the San Jose city manager might be interested in the sales of San Jose stores for the last three months, the California region manger might want to compare the current month sales to same month sales last year by California cities, which can require a lot of work if you were just using a relational database.

It is important to model the relational database correctly to meet not only the business requirements but the performance requirements as well, which we discuss extensively in the first book in the *multidimensional trilogy* series, *Dimensional Modeling: In a Business Intelligence Environment*, SG24-7138.

1.1.2 Providing OLAP data

Given the requirement to do OLAP, what kind of database engine is needed? One characteristic of OLAP analysis is that the analysis usually begins by looking at aggregated data over some time period. Another important characteristic of

OLAP analysis is the exploratory nature of the analysis such that we cannot predict the exact path a user might take through the data.

Although relational databases are good at ad hoc types of queries, the ability to effectively deliver highly aggregated data has traditionally been an issue that led to the development of specialty database engines that delivered highly-aggregated data with good response times. These databases used special storage mechanisms and pre-aggregated the data along all dimensions using highly-indexed system to directly access specific cells of aggregated data across the various dimensional values. These types of databases were called *Multidimensional OLAP databases*, or MOLAP.

Although MOLAP systems delivered highly-aggregated data across dimensions with impressive response times, there was definitely a cost that was associated with it. These databases required that data be extracted from the data warehouse and loaded into the MOLAP database, thereby duplicating the data. MOLAP databases also required that the data be pre-calculated across all combinations of dimension members, which resulted in an exponential growth rate in the required storage. This calculation step was also expensive in terms of the CPU resource cost and elapsed times. It was not unheard of to have calculation times measured in days. However, MOLAP engines provide excellent response times and usually much better calculation engines for sophisticated financial analysis.

On the other end of the spectrum, there were products that used the relational database as the OLAP engine. These products are known as *Relational OLAP databases* or *ROLAP engines*. ROLAP products developed a way to provide aggregated data in an effective manner, usually using some type of summary tables. The ROLAP engines determined, either automatically or through user choice, whether to use summary tables or the detail tables to satisfy a particular query.

The advantage of ROLAP is that it was relational based, and the data was not copied into another storage mechanism that needed new and different operational procedures. ROLAP data was stored in the relational database and the normal operational procedures applied. However, there was a drawback, primarily in the area of inconsistent response time, because relational database solutions do not typically have the level of sophisticated calculations that are MOLAP engines have.

However, relational database technology did not stand still. There were dramatic improvements in their ability to handle aggregated data or summary tables. IBM DB2, for instance, has *Materialized Query Tables* (MQTs). MQTs are summary tables that you can use in multidimensional or OLAP analysis. The power of MQTs in DB2 is that it is transparent to the user or calling program. You can develop the query as though it is going against the detail, base, or tables. The

DB2 optimizer analyzes the query and decides if it is cost less to satisfy a particular query by using the base tables or by using the MQTs. Even updating the MQT as data is loaded into the data warehouse can be delegated to the relational database engine. Although using MQTs in a ROLAP scenario can improve the response time of OLAP queries, it still does not approach the level of response of a MOLAP engine.

InfoSphere Warehouse Cubing Services takes a hybrid approach to delivering OLAP. From a user or application viewpoint, it appears to be a MOLAP engine using *multdimensional expressions* (MDX) as its query language. However, the data is not copied or persisted in another storage mechanism, but rather the eventual back end is the DB2 relational engine. Cubing Services is in fact a multidimensional OLAP *hot cache*.

In memory, InfoSphere Warehouse Cubing Services builds a partial MOLAP structure that can deliver quick response times for the data that is in memory. If a user or query tool requests data that is not in the cache, the Cubing Services engine generates queries to the relational database to retrieve the data and build an in-memory cube for that data. Although the first request for that data sees a longer response time, subsequent requests for that data are satisfied from the in-memory cube and experience MOLAP style response times if that data is maintained in memory. Using Cubing Services as the OLAP engine results in a MOLAP type of response, in a large percentage of queries, without the need to copy the data into another storage mechanism. To improve the relational access, there is an optimization wizard that recommends the appropriate MQTs that can be implemented to improve response of aggregated data when it is needed from the relational database, which we discuss in more detail in Chapter 2, “A multidimensional infrastructure” on page 11.

1.1.3 Consuming OLAP data

Having data in an OLAP database is meaningless unless it can be retrieved and displayed to the user in a manner that is relevant and effective for the business requirement, which is an area that is constantly changing to add more capability to how data is turned into information that is timely and relevant for the user audience. One key aspect of this is in part of the acronym OLAP, which is the term *Online*. You can certainly perform multidimensional analysis using batch reports. However, it is not uncommon for a user to receive a report and scan through it looking for items of interest based on their experience and knowledge of the business. Many times the user is looking for items that are out of line with expectations, whether good or bad, and then one or more additional reports is run to explore those deviations. Sometimes these reports already exist and are returned in minutes or hours, but in some cases a new report must be defined,

which can take days or even weeks, by which time the information might be irrelevant or just plain too late to act upon.

Business decisions these days are often made quickly, and as such the user cannot afford to wait hours or days while the next report is developed. In fact, the user needs to see new views of the data while the thought is fresh. Remember, that the goal of OLAP is to deliver information at the *speed of thought*, which does not imply sub-second response time to every data request but rather that the new view of the data must be delivered to the user quick enough to keep the train of thought moving forward. This quick movement might mean 10 seconds to some users while minutes might be fine for another users. Another aspect to OLAP style analysis is that the user is not submitting report requests but directly interacting with data, which means that the user clicks a data value or graphical element that represents a data that can then return more detail behind that particular data value. Alternately, a user can drag-and-drop additional data attributes onto the OLAP display to add to or take away from values or to pivot the view of the data. The important point here is that the user is interactively engaged directly with the data.

How the OLAP data is displayed to the user and how much interaction is allowed depends entirely on the requirements of the user. Data for a business executive might be displayed entirely using maps and other highly graphical elements and using color coding to represent business conditions, for example, the VP of sales for the US can see actual sales versus target by using a map of the US with each state color coded to represent how that state is doing against targets. Using the color red to indicate that a state is missing their target is a common example. The executive might then click a particular state of concern to see a view of the sales territories in that state and so on. A business analyst or knowledge worker might see line or pie graphs accompanied by a grid view of the data with more freedom to explore by allowing pivoting, drill down, and drill up capabilities. There is a trend to push analytics out to the line-of-business workers giving line management a guided analysis capability of data scoped to be relevant to their part of the business.

In all of these cases, the OLAP display tool must be able to speak to the underlying OLAP engine, and maybe even directly to the relational database, to integrate OLAP style analytics with more traditional reporting and perhaps even integrated in a Web portal.

1.1.4 Pulling it together

So what does all of this mean to the business? Actually, it is simple. Using multidimensional analysis through OLAP databases and tools you can gain much better insight to your business than traditional reporting. It is all about delivering

the right information to the right user at the right time in a manner that is relevant to the user to support the user decision making process.

It takes a combination of the right multidimensional data model, the right OLAP database engine, and the right OLAP data delivery tooling to make OLAP effective for an organization. It is not always easy, but when done right, the payoff is huge.

1.2 Conclusion

In the remainder of this book, we focus on the IBM software stack that enables you to implement OLAP using the Cubing Services component of InfoSphere Warehouse, along with Cognos BI, which you can use in conjunction with the other two books previously mentioned that comprise the multidimensional IBM Redbooks publication trilogy. These publications can help you understand how to develop the relational multidimensional model, understand the Cubing Services technology, see various scenarios modeled in Cubing Services, and how to deploy OLAP to users using Cognos BI. Good reading.

2

A multidimensional infrastructure

In this chapter, we describe the basics of a multidimensional infrastructure and how the IBM InfoSphere Warehouse family of products can help you achieve a powerful multidimensional analysis environment. First, we provide a brief overview of how source data is captured from the operational environment and transformed into an informational, or data warehousing, environment. From there you can use Cubing Services, one of the components of the IBM InfoSphere Warehouse, to create a *multidimensional cube*, which enables users to perform multidimensional analysis of their data. InfoSphere Warehouse Cubing Services is the focus in this book.

For additional reading on this subject, there are two Redbooks publications that complement this book and provide details about multidimensional modeling and InfoSphere Warehouse Cubing Services:

- *Dimensional Modeling: In a Business Intelligence Environment*, SG24-7138
- *InfoSphere Warehouse: Cubing Services and Client Access Interfaces*, SG24-7582

These three books comprise a *trilogy* that can help you to understand, model, and build a multidimensional environment for powerful data analysis.

Now is the time to start building your multidimensional infrastructure with InfoSphere Warehouse and Cubing Services, and then continue building a multidimensional environment for problem solving and a business advantage.

2.1 The need for multidimensional analysis

From a strategic point-of-view, a good way to analyze data is from top to bottom. Start with results from the most aggregated values, and then obtain more details about the values that comprise the results. As an example, for a global business you want to know the profit by country. So, start with the result that represents profit by all countries. In the country dimension, you want a query that shows the profit numbers for each country in that dimension.

With those numbers, you can then compare the countries and rank them from highest profit to lowest. Next, you can drill-down on any of the countries and access even more detailed data, for example, you see the profit numbers for each of the states in that country. This type of analysis lends itself well to quickly identifying the areas that might not make their performance targets, which, in turn, gives management an opportunity to take fast action to correct the issues involved, and possibly get performance for those states back on track.

So far, we only analyzed data from one perspective (or dimension), and that was countries. But, what if you now want to know details about product performance within each country? You then want to compare actual profit numbers with those from the previous month or months. Now you are talking about performing multidimensional analysis!

Regarding multidimensional analysis, a dimension is used to organize a set of related attributes that describes one aspect of a measure. As examples, typical dimensions are measures, time, location, product, scenarios, customers, and departments.

It is no surprise to anyone that the volume of data that is generated by running any business keeps growing steadily, and there are no signs of reversing that trend, which implies that you must work with more and more data, as time passes. The larger the data volume, the longer it takes to select filter and aggregate the data that is needed to generate a query response. To obtain the profit numbers that we previously mentioned over a single dimension (or perspective), such as countries, you are required to scan many large tables that contain millions or even billions of rows each, and then doing some math with the profit numbers for all countries. As more dimensions are added, the volume of data grows quickly.

InfoSphere Warehouse Cubing Services provides for the multidimensional analysis capability that we just described and has the resources needed to quickly answer those queries.

2.1.1 Identifying uses for a cube

If you ask a user what type of report or analysis is needed, you typically get a response, such as “I must analyze something BY something BY something,” and so forth. For such an analysis, using a cube is likely required because there are multiple dimensions involved. A cube is not needed to answer questions, such as how many stores are there or what is the sum of all salaries of a department. Any relational database can provide answers to those types of questions. But, a cube is appropriate to enable analysis of data according to multiple perspectives and to change any or all perspectives easily and quickly.

So what are the benefits of using a multidimensional analysis? The answer is to analyze a huge volume of data, from multiple perspectives. Suppose you must look at the sales data for products in several countries, for all periods of time. To help visualize data from these multiple perspectives, visualize a cube with each of the axes representing a dimension, for example, consider country, product, and time. Also consider the implicit 4th dimension, measures, which is the sales amount. Look at the cube, comprised of a number of mini-cubes that is represented in Figure 2-1. Each mini-cube contains a value (or measure) for that specific intersection of the country, product, and time. As an example, the darker mini-cube, blue if you can see it in color, represents the sales for the first quarter of 2008 (time dimension) for the grocery item milk (product dimension) that was sold in Germany (country dimension).

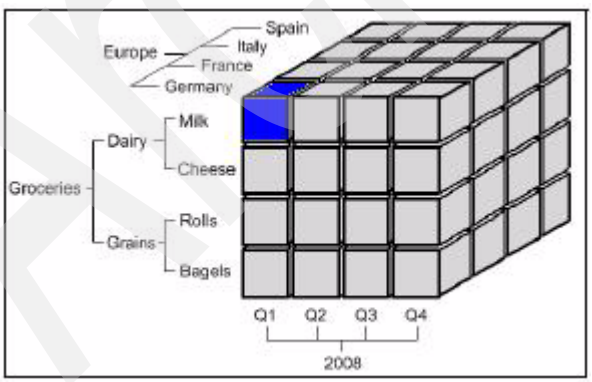


Figure 2-1 Multidimensional data represented as a Cube

It can help your understanding to see the data in a spreadsheet, as shown in Figure 2-2 on page 15. The darker cells (gray if you can see it in color) contain

the input data from the operational systems. The multidimensional (or OLAP: On Line Analytical Processing) engine can then calculate the aggregate cells (all non-grey cells if you can see them in color) after the data is loaded. Cell C2, the dark blue cell, represents the same intersection of data points as shown in the cube example in Figure 2-1 on page 13 (sales of Milk in Germany, in the first quarter of 2008).

Be aware that in the cube example, the mini-cubes, or cells, that are used for storing the aggregate data (values for Europe or 2008, for example) are not represented.

	A	B	C	D	E	F	G
1			Germany	France	Italy	Spain	Europe
2	Q1	Milk					
3		Cheese					
4		Rolls					
5		Bagels					
6		Dairy					
7		Grains					
8		Groceries					
9	Q2	Milk					
10		Cheese					
11		Rolls					
12		Bagels					
13		Dairy					
14		Grains					
15		Groceries					
16	Q3	Milk					
17		Cheese					
18		Rolls					
19		Bagels					
20		Dairy					
21		Grains					
22		Groceries					
23	Q4	Milk					
24		Cheese					
25		Rolls					
26		Bagels					
27		Dairy					
28		Grains					
29		Groceries					
30	2008	Milk					
31		Cheese					
32		Rolls					
33		Bagels					
34		Dairy					
35		Grains					
36		Groceries					

Figure 2-2 Multidimensional data represented as a spreadsheet

2.1.2 Getting answers with no queries

One of the greatest benefits of using OLAP analysis is the fact that users do not need to learn a new language to query the data. With tools, such as Cognos, Alphablox, and Microsoft Excel, users can analyze the data using a *drag-and-drop* method. These tools have an interface to the data in the InfoSphere Warehouse Cubing Services server. When using one of these tools, all the user must do is to drag-and-drop dimensions, or one or more of the members (children), to the spreadsheet-like interface, and the data related to that intersection is fetched from the cubing services server and simply appears in the tool.

You can then move the dimensions or members with these tools to easily change the perspective, which is demonstrated in Figure 2-3 by showing two perspectives of the same data. The left side of Figure 2-3 shows the sales of *all the products* (in column C) for each quarter. We changed that perspective by dragging the Q1 from column A and dropping it into column B, which resulted in the items on the right side of the figure. You now see the sales of *each product* by quarter.

So, the user can manipulate the data to see the desired perspective without submitting a query. With this capability, users can now stop learning how to generate queries and concentrate on forming business questions.

	A	B	C
1			Sales
2	Q1	Milk	1
3		Cheese	2
4		Rolls	3
5		Bagels	4
6		Dairy	3
7		Grains	7
8		Groceries	10
9	Q2	Milk	10
10		Cheese	20
11		Rolls	30
12		Bagels	40
13		Dairy	30
14		Grains	70

	A	B	C
1			Sales
2	Milk	Q1	1
3		Q2	10
4	Cheese	Q1	2
5		Q2	20
6	Rolls	Q1	3
7		Q2	30
8	Bagels	Q1	4
9		Q2	40
10	Dairy	Q1	3
11		Q2	30
12	Grains	Q1	7
13		Q2	70
14	Groceries	Q1	10

Figure 2-3 Different perspectives of the same data

Be aware that SQL is not used for these queries. There is another query language for OLAP, which is called MDX. However, it is not intended for users,

but only for programmers. For more details about MDX, refer to another cubing services book, *InfoSphere Warehouse: Cubing Services and Client Access Interfaces*, SG24-7582.

2.1.3 Components of a cube

So, what is in a cube? A cube contains dimensions, and measures for each of the dimensions intersections. Even if a dimension is not referred to in a query, there is still an implicit referral. If we again use the spreadsheet example in Figure 2-2 on page 15, those measures would be for the dimensions Milk and Q1. Because Country was not referred to, all of those measures are related to the highest level of aggregation of the Country dimension, which in this case is Europe. Remember that the measures, or metrics, belong to a dimension which is usually called Measures, so our cube actually has four dimensions.

2.1.4 Selecting dimensions

One question that might arise is how to select the dimensions to use? Two dimensions typically found in most cubes are two special dimensions:

- Measures, the dimension that holds all loaded and derived measures
- Time, the dimension that enables the ability to track how the same measure compares over time periods. If this dimension (which can have any particular name) is tagged as TIME, then the OLAP engine can perform some automatic calculations, such as Year-to-Date or Season-to-Date.

Other dimensions depend on the purpose of the cube. For retail analysis, for example, besides Measures and Time you might want to add such dimensions as Geographic location, Store, Cashier, Product, and Loyalty Card. Insurance companies might need more than just one time dimension. As examples, there can be one for underwriting dates, another for accident date, or yet another one for payment date, and possibly others, such as customer and insurance plan.

As a rule of thumb, do not create a cube with too many dimensions in an effort to try to answer the many different business questions. It is preferable to create two or three smaller cubes, with fewer dimensions, and treat many of those business questions separately. Even though there is no limit on how many dimensions you can add, a cube with say 10 or 12 dimensions becomes difficult to manage and even more difficult to use! You have too many variables to deal with when analyzing data. A good range is from four-to-eight dimensions. Create a cube with as few dimensions as possible to respond to a specific group of related business questions.

2.1.5 Why create a star-schema

After you identify the dimensions to be used, consider where this data for these dimensions comes from. Although it is possible to use a standard, normalized database as a source to the OLAP engine (InfoSphere Warehouse Cubing Services), you do not want to do it because there are many joins between large tables, which result in performance that is not optimized, which, in turn, has a negative impact on response time for queries that are submitted.

The fastest response you can possibly get from a database occurs when all you need is data from a single table, that is, no joins are required. This is the concept behind the use of a star schema, and the fact table it uses. In the creation of a fact table, the data from many tables (from the ER model) is extracted and prepared, and then inserted into the fact table. That fact table also contains key columns for each dimension that is needed to solve the business problems. It also contains, for each row, the corresponding measures, which comprises the basic multidimensional structure. Even though a fact table might have a huge number of rows, proper use of indexes can make it much faster than the conventional normalized model.

Also present are the dimension tables, which, ideally, are joined to the corresponding column of the fact table using an artificial numeric key. They have the description of, and more details about, that specific dimension, such as names and the hierarchy from the lowest level up to the top members, which is the dimension name.

The star schema is then used to load the multidimensional cube. The cubes that we used in our examples could have been loaded using this simple fact table, such as the one in Figure 2-4 on page 19.

	A	B	C	D	E	F
1	Q1	Milk	Germany	53	2	51
2			France	31	7	24
3			Italy	12	4	8
4			Spain	78	8	70
5		Cheese	Germany	2	0	2
6			France	11	2	9
7			Italy	59	4	55
8			Spain	66	2	64
9		Rolls	Germany	82	0	82
10			France	16	4	12
11			Italy	87	6	81
12			Spain	19	9	10
13		Bagels	Germany	37	5	32
14			France	57	3	54
15			Italy	96	4	92
16			Spain	88	7	81
17	Q2	Milk	Germany	94	8	86
18			France	6	6	0
19			Italy	52	8	44
20			Spain	40	8	32
21		Cheese	Germany	43	7	36
22			France	43	1	42
23			Italy	14	7	7
24			Spain	89	4	85
25		Rolls	Germany	75	5	70
26			France	37	3	34
27			Italy	38	3	35
28			Spain	29	1	28
29		Bagels	Germany	97	3	94
30			France	0	0	0
31			Italy	89	4	85
32			Spain	53	4	49

Figure 2-4 Simple Fact table sample

In Figure 2-4, columns A, B, and C are used for dimensions, and columns D through F contain the measures, selling price, discount, and final price. Note that typically only the lowest level of detail data (or grain) exists.

2.1.6 More help from InfoSphere Warehouse Cubing Services

In any OLAP system, the more dimensions that are added, the bigger the occurrences (or different combinations) between all of the dimension members. The number of possible combinations for the intersections of all dimensions can grow fast. You can calculate it by multiplying the number of existing members of all dimensions, for example, all Dim1 members x all Dim2 members x all Dim3 members, and so on.

The IBM InfoSphere Cubing Services not only provides the multidimensional analysis capability, but also helps with creating summary tables. These tables hold the aggregated values for the hierarchies above the grain level, for example, Dairy, which is calculated summing the values for Milk and Cheese, for all intersections of the remaining dimensions (and their descendants). These aggregations can be calculated online (however, certain restrictions apply) or scheduled to be calculated at a suitable non-peak hour. A cube has one or a more summary tables, or MQTs (Materialized Query Tables), which tend to be much smaller than the fact table (often by one or two orders of magnitude). The IW Cubing Services analyze the data, and it generates the scripts that are needed to create and maintain those MQTs. The original data is not duplicated in the process of creating the MQTs because they contain only the summarized, aggregated data, which can increase the performance of a query by 95% or more!

2.2 An architecture

Figure 2-5 on page 21 shows a generic data flow for a business intelligence (BI) environment. The data needed in that environment can come from a variety of sources. Typically, data from various operational sources are extracted, transformed, modified, and loaded into an operational data store (ODS), a data warehouse, and one or more data marts.

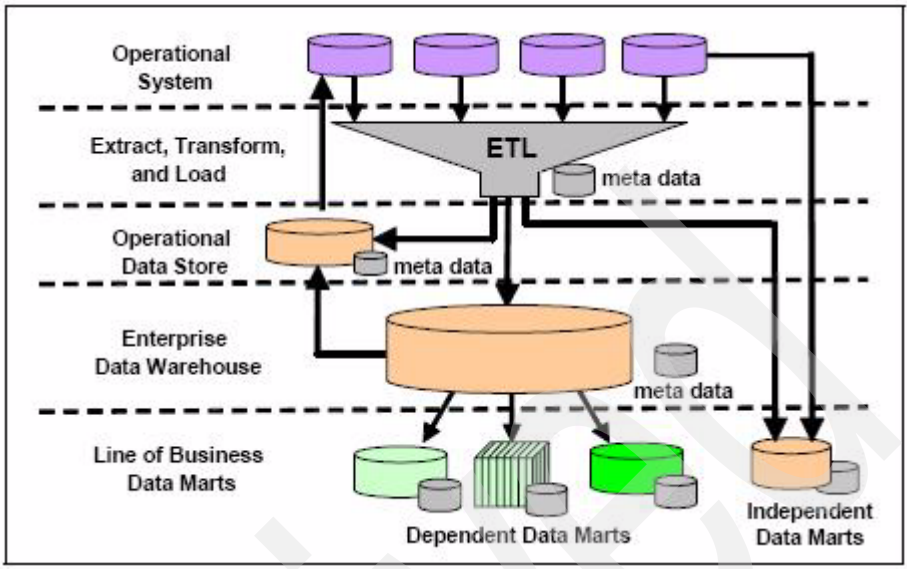


Figure 2-5 Example data warehouse architecture

The data in an ODS might or might not be fully normalized, but the data warehouse is almost always normalized. The data marts, dependent or independent, are essentially dimensional models.

To explain how to build an *enterprise data warehouse* (EDW) is beyond the scope of this book; however, throughout this section, we present and discuss some of the IBM software products that are typically used in building every phase of such a project.

2.2.1 From data sources to data marts

In this section, we highlight the steps that might be involved to store and analyze data that is generated, during normal business operations, from transactional systems and stored in data warehouses and data marts.

There are a number of ways to implement a data warehousing environment because of the large number of variables that are involved in the process. Building a data warehousing environment is a major and critical task because it becomes the source for data analyses across the entire business. It typically takes quite a long time to implement it, so it is wise to give it much consideration and planning before you implement it, which is primarily because any design problems that you find after implementation are time consuming and difficult to fix after implementation. As you read further, you find many more details about

dimensional modeling discussed in two other publications:

- *InfoSphere Warehouse: Cubing Services and Client Access Interfaces*, SG24-7582
- *Dimensional Modeling: In a Business Intelligence Environment*, SG24-7138.

Data sources for a data warehousing environment include operational data from the internal transactional systems and data from external sources. Processes to extract, transform, and load (ETL) data from those sources provides all of the necessary data transformations and then might store the data on a staging area to await further processing. After all processing finishes, the data is placed in a data warehouse. In many cases, some of the data is distributed to one or more data marts, which typically occurs when many users are remote, need particularly fast response times, and need the data to be highly available, as shown in Figure 2-6.

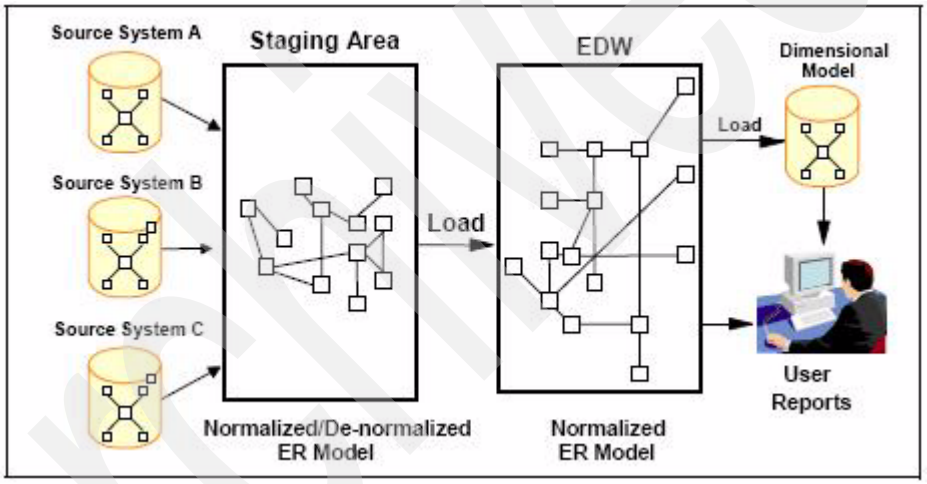


Figure 2-6 Querying from a dimensional model

Building those data repositories requires careful planning. A data modeling life cycle for a data warehouse begins with a critical business requirements gathering. They then drive the development of a logical data model, which then leads to development of the physical data model.

IBM has an extensive line of powerful software products for all phases of the data warehouse life cycle. Some of those are represented in the example process flow shown in Figure 2-7 on page 23.

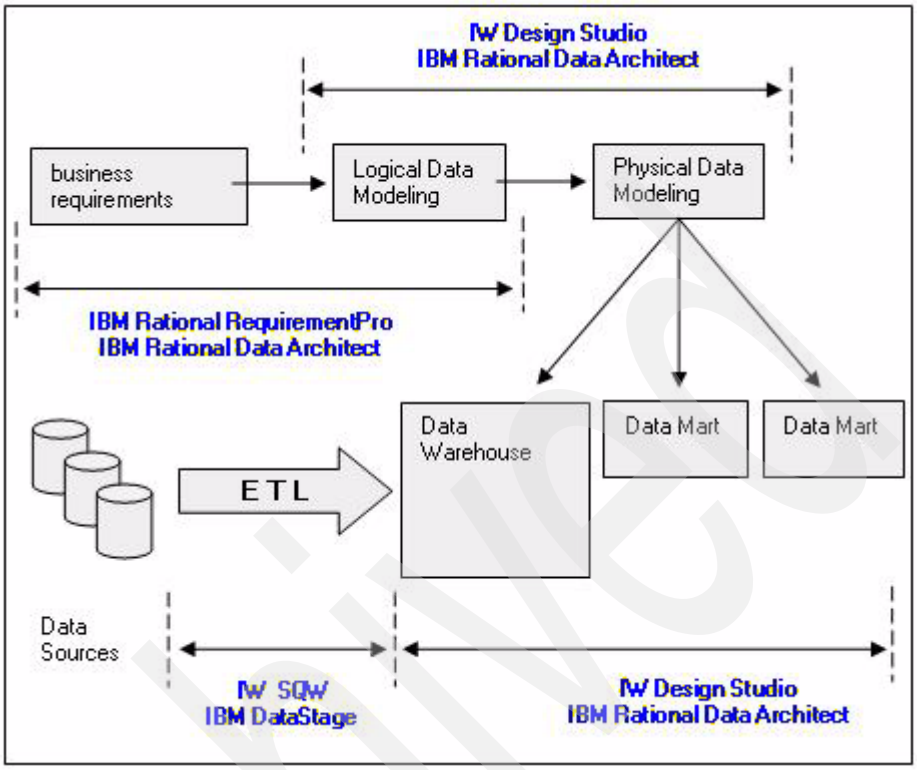


Figure 2-7 IBM software products for data warehousing

Table 2-1 shows each life cycle phase and the suggested product to use.

Table 2-1 Phases and recommended IBM software products

Phase	Software
Business Requirements	Rational® RequisitePro® Rational Data Architect (RDA)
Logical Data Modeling	Design Studio Rational Data Architect (RDA)
Physical Data Modeling	Design Studio Rational Data Architect (RDA)
ETL Processes	InfoSphere SQW DataStage® / QualityStage
Data Servers	InfoSphere Warehouse InfoSphere Balanced Warehouse™

IBM Rational RequisitePro

IBM Rational RequisitePro is a requirements management tool that facilitates requirements organization, integration, traceability, and analysis. The business requirements can be gathered from users and used in designing a database logical model.

Integrating RequisitePro with the Rational Data Architect can make it even easier to develop the logical model. In Figure 2-8, we show an example of that integration, where you can see the list of data models (car is highlighted) and the list of requirements (with Feat 6 highlighted). On the right side of Figure 2-8 is a view of the logical model that is being built. The car model and Feature 6 were dragged and dropped as a step in creating the logical model.

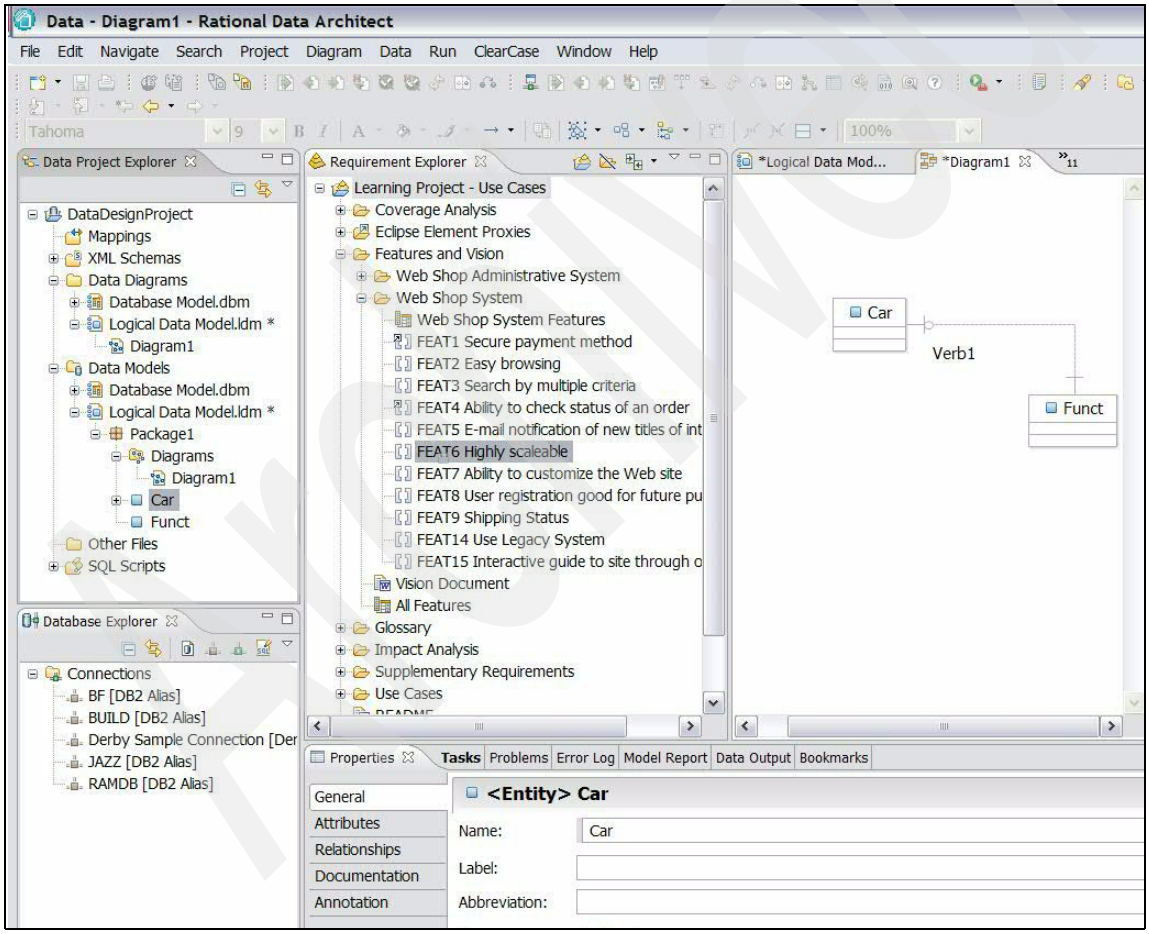


Figure 2-8 Integration of RequisitePro and Rational Data Architect

Another key value of RequisitePro is the Impact Analysis. When a business requirement changes, RequisitePro can identify all affected tables and measures, as shown in Figure 2-9.

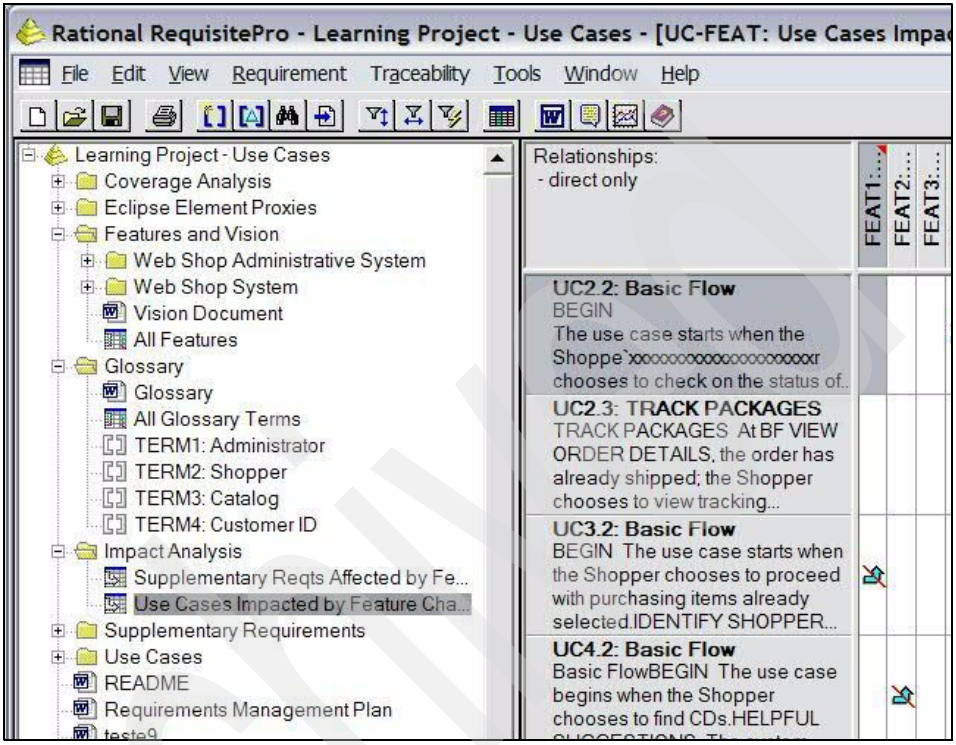


Figure 2-9 IBM Rational RequisitePro

You must purchase this product separately. For more information, visit the following Web site:

http://www-01.ibm.com/software/awdtools/reqpro/

IBM InfoSphere Warehouse Design Studio

The Design Studio is an Eclipse-based graphical tool that you can use to create and maintain Cube Models and Cubes. It can also generate the scripts for creating and refreshing the MQTs that you can use to increase the OLAP performance. The OLAP objects that are created are stored in a common metadata repository database, as shown in Figure 2-10 on page 26.

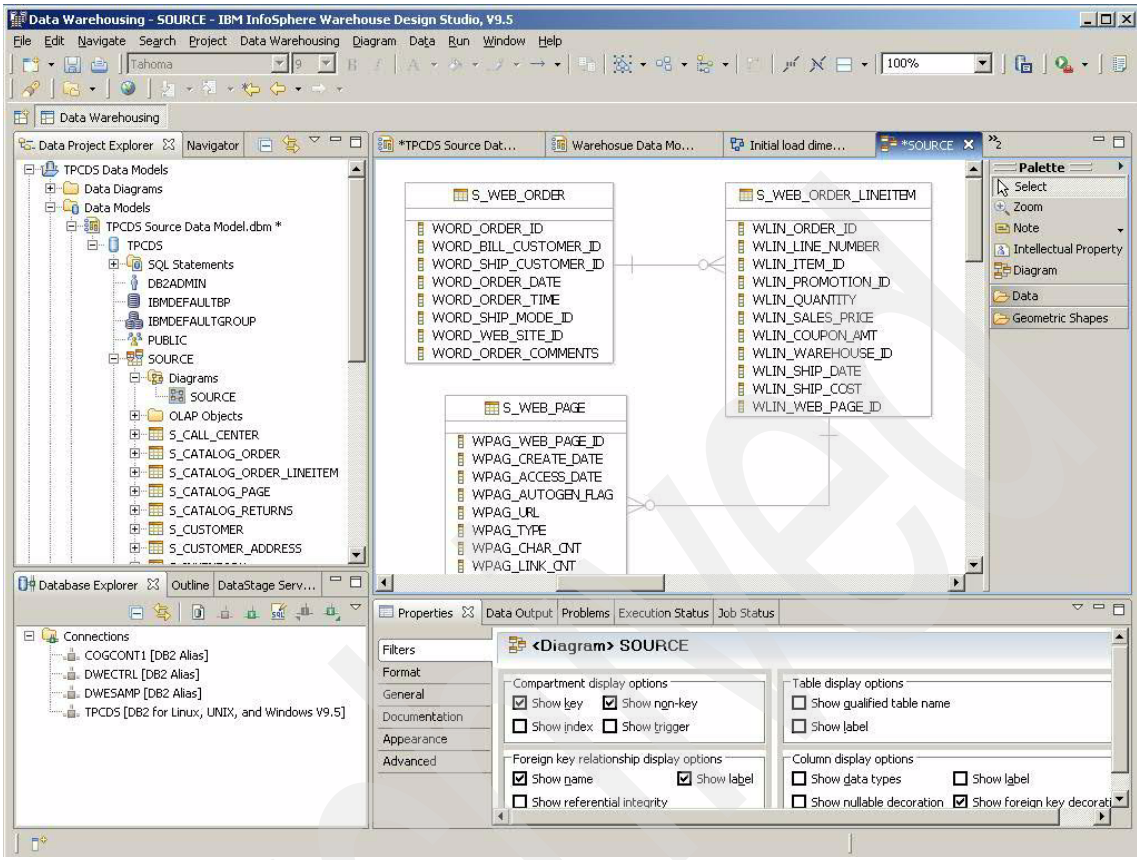


Figure 2-10 IBM InfoSphere Warehouse Design Studio

The Design Studio is included in all InfoSphere Warehouse Editions. For more information, visit the following Web site:

`http://www-01.ibm.com/software/rational/eclipse/`

IBM Rational Data Architect

IBM Rational Data Architect simplifies data modeling and integration design, which enables architects to discover, model, visualize, and relate diverse and distributed data assets. It is the product that you can use to create logical and physical data models. It also has capabilities for reverse engineering and impact analysis. You can also use it to compare and synchronize the structure of data sources and targets. The life cycle integration capability helps you integrate with other facts of the life cycle. Direct integration with IBM Rational Software Architect allows architects to work together with transformations between the software architecture model and the data model. You can access, associate to

corresponding modeling elements, and synchronize with user-selectable rules the requirements that are stored and managed in IBM Rational RequisitePro. IBM Rational ClearCase® or CVS can manage modeling files, providing seamless versioning, branching, and synchronizing of changes, which enables full support for the team environment. It uses the same Eclipse interface that the Design Studio uses, as shown in Figure 2-11.

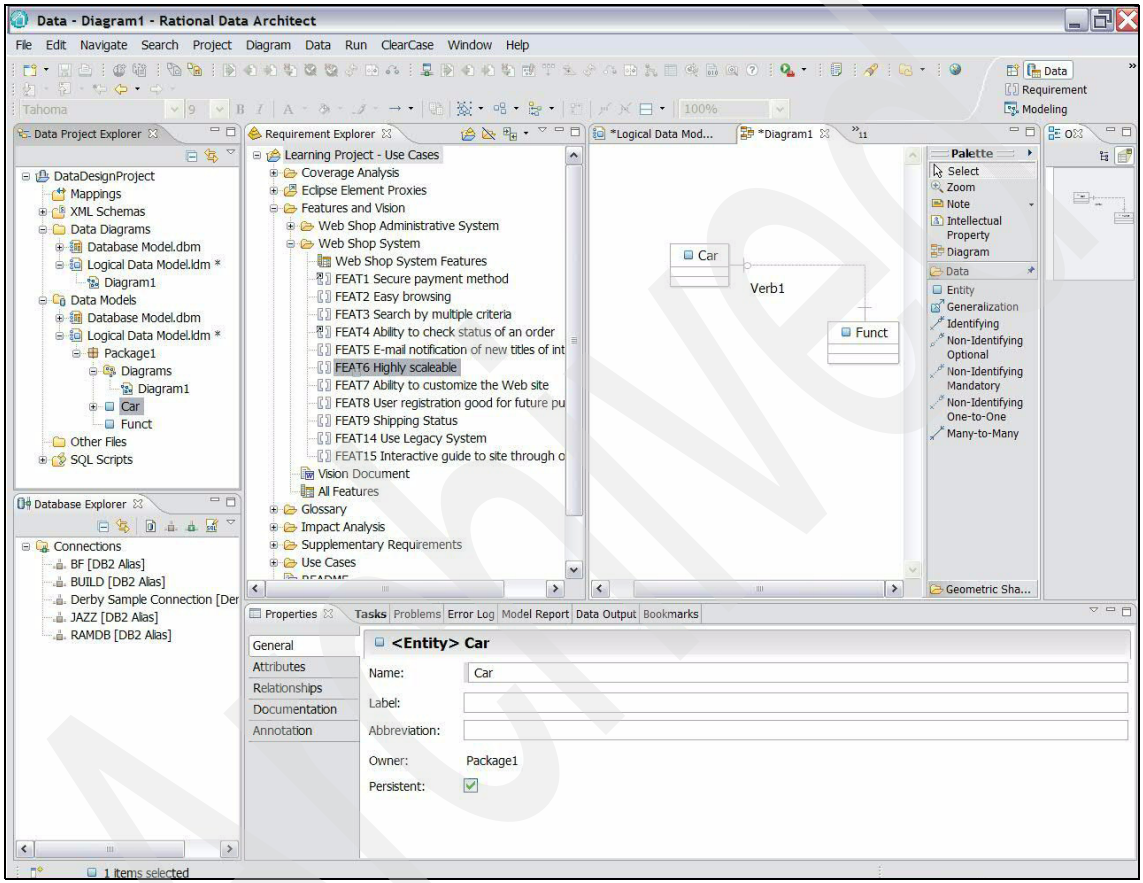


Figure 2-11 Rational Data Architect

You must purchase this product separately. For more information, visit the following Web site:

http://www-01.ibm.com/software/data/integration/rda/

IBM InfoSphere Warehouse SQL Warehousing

SQL Warehousing (SQW) is a tool that is designed to support data movement and integration requirements in an InfoSphere Warehouse environment. It

provides you with a drag-and-drop GUI to create logical flows of data movements. Each of those can include one or more SQL statements. Even with no SQL knowledge, you can benefit from the SQW graphical query construction. It has an extensive palette of standard SQL transforms and specialized warehousing operators, such as SCD, key lookup, and pivot, and a broad range of source and target operators, support for parallel processing, and it easily integrates with WebSphere® DataStage. Figure 2-12 shows the InfoSphere Warehouse SQL Warehousing (SQW).

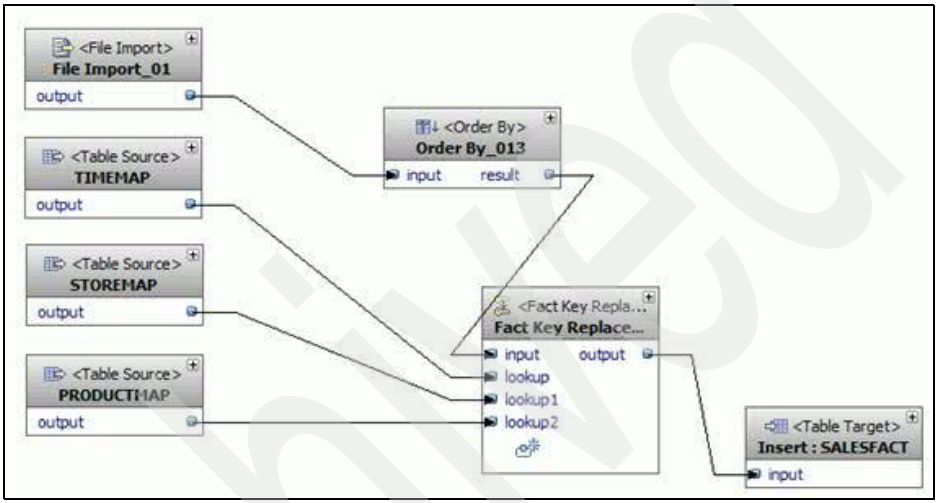


Figure 2-12 IBM InfoSphere SQL Warehousing Tool (SQW)

SQW is included in all InfoSphere Warehouse Editions. For more information, visit the following Web site:

`http://www-01.ibm.com/software/data/infosphere/warehouse/sql.html`

IBM InfoSphere DataStage

The IBM InfoSphere DataStage integrates data on demand with a high performance parallel framework, extended metadata management, and enterprise connectivity. It is a scalable and powerful ETL solution for large volumes of data and manages data that arrives in real-time and data received on a periodic or scheduled basis. It can connect to a virtually unlimited number of heterogeneous data sources and targets, which includes almost all databases (even partitioned databases), ERP systems, such as SAP® and PeopleSoft®, and even business intelligence tools, such as SAS. Figure 2-13 on page 29 shows an example.

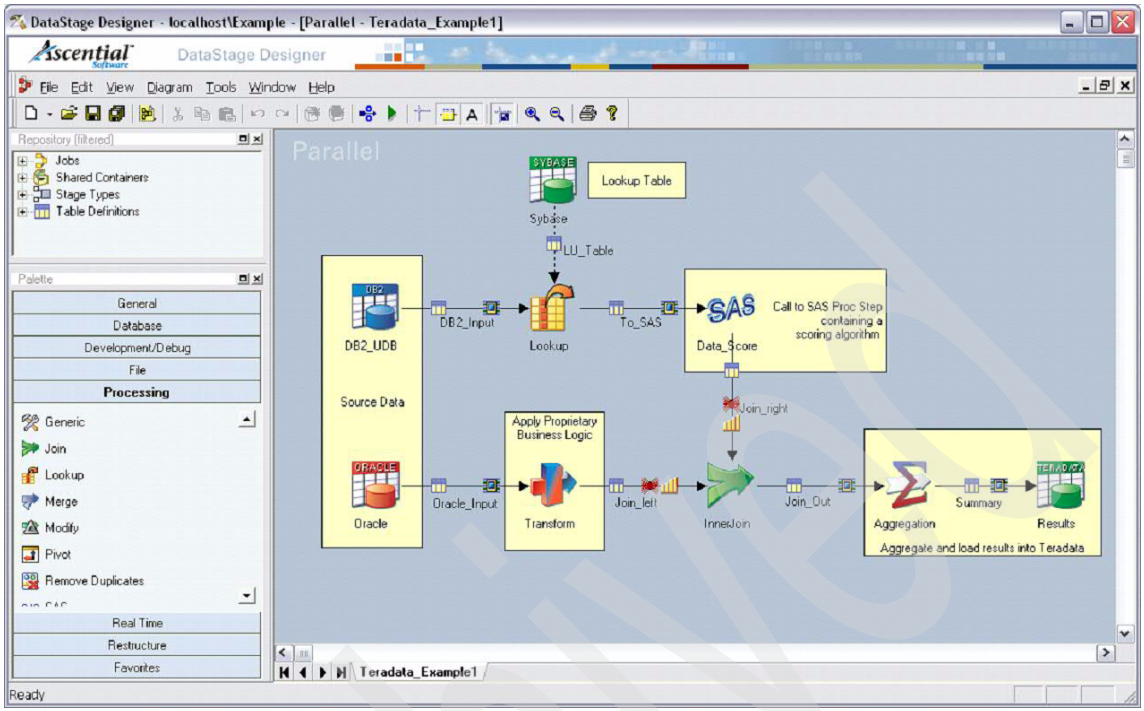


Figure 2-13 IBM InfoSphere DataStage / QualityStage

You must purchase this product separately. For more information, visit the following Web site:

http://www-01.ibm.com/software/data/infosphere/datastage/

IBM InfoSphere Warehouse Suite

IBM InfoSphere Warehouse provides everything that is needed to cost effectively implement a flexible, scalable data warehouse for dynamic warehousing. It enables an ideal solution for companies that must consolidate data marts, information silos, and business analytics to deliver a single version of the truth to all users, within context and in real-time. Providing a powerful range of capabilities that go beyond traditional data warehouses, InfoSphere Warehouse is a comprehensive platform that includes tooling and infrastructure to help data warehouse architects and administrators efficiently design, deploy, and maintain an enterprise data warehousing environment. Figure 2-14 on page 30 illustrates the suite.

The capabilities of the InfoSphere Warehouse Suite include:

- Powerful DB2 data server foundation
- Enhanced OLAP design and optimization (Cubing Services)

- Data Mining and visualization
- Modeling and design tools (Design Studio)
- Embedded data movement and transformation (SQW)
- Unstructured data analysis
- Alphablox Bloxbuilder
- Extreme Workload Management

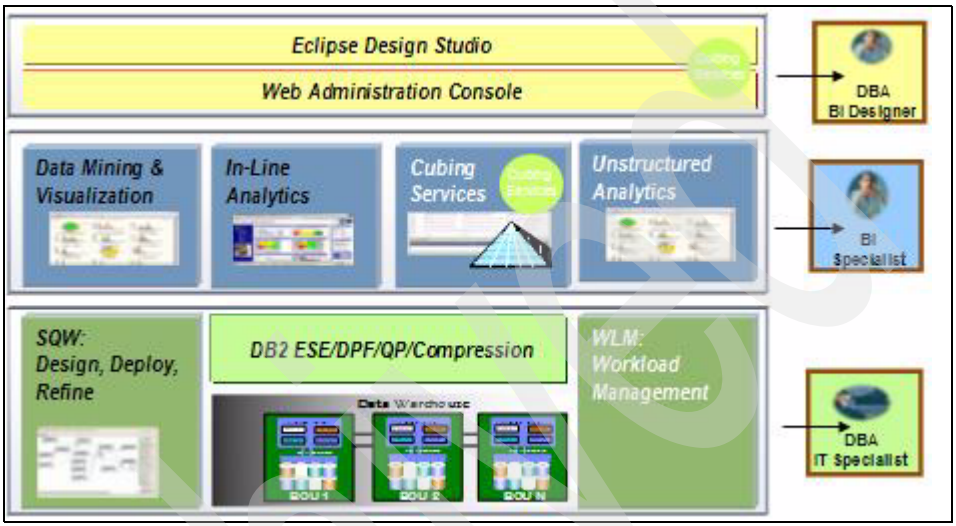


Figure 2-14 IBM InfoSphere Warehouse

For more information, see the following Web site:

http://www-01.ibm.com/software/data/infosphere/warehouse/enterprise.html

2.2.2 From data marts to accessing cubes

With the data needed for the cubes already in a data mart environment, we can take a look at what steps are required to implement a multidimensional cube. From a high-level perspective, you must do the following tasks, depending on which of the base platforms you choose:

1. Design Studio:
   a. Import the star-schema metadata from the data mart to Design Studio.
   b. Design the cube model and cubes and deploy them back to data mart.
2. WebSphere:
   a. Import the OLAP metadata from the data mart into cubing services.
   b. Create a Cube Server to which you can add the cubes.

Figure 2-15 shows the steps needed and some additional details.

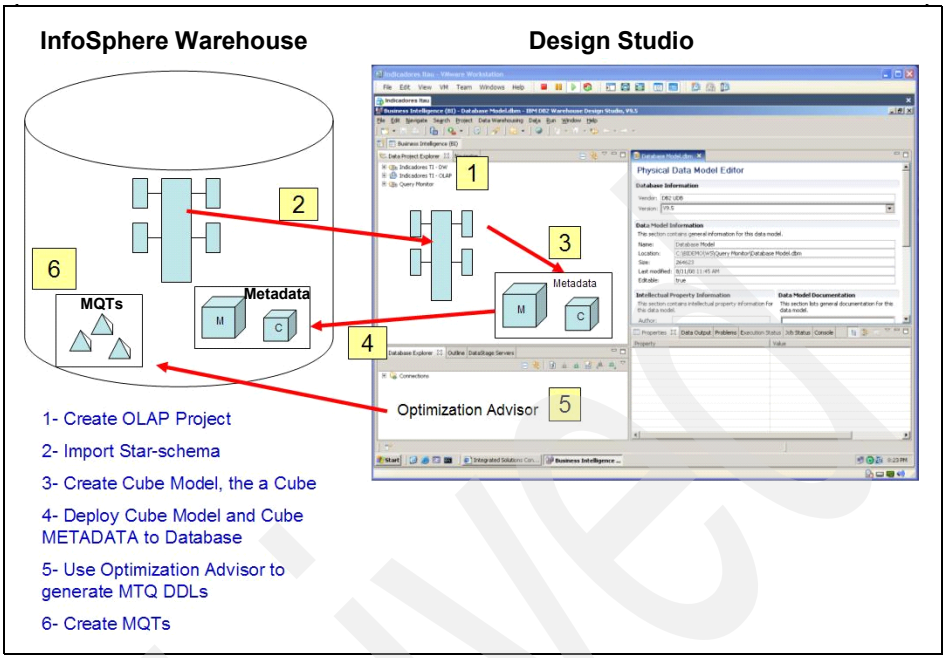


Figure 2-15 Creating a Cube: Part1

Important: We are using two different products, Design Studio and WebSphere Application Server Console; therefore, we must use two different Interfaces. In the following instructions, steps 1 through 6 use the Design Studio, and steps 7 through 10 employ the WebSphere console.

In the following instructions, steps 1 through 6 use the Design Studio, and steps 7 through 10 employ the WebSphere console. Refer to Figure 2-15 as you review the following steps:

1. Create a new OLAP Project in Design Studio, and name it accordingly.
2. Do a reverse engineer import to obtain the data mart star schema metadata. Note that we are not importing the data, but the metadata. Therefore only the information about the structure of the multidimensional model (star-schema or snow-flake model) is imported. The Design Studio now knows the multidimensional model of the data mart.
3. Create one or more CUBE MODELS out of the star schema model. During the cube model creation, you select which of the available star schema dimensions to use. Configure the existing levels for each hierarchy (time dimension, for instance: months, days, years, and quarters) and also the hierarchy of them (years, quarters, months, and days).

After all CUBE MODEL dimensions are set, it is easy to create CUBES. You simply select the dimensions and measures that are needed. At this point, the Design Studio knows the cube models and cubes to create. You can create one or more CUBE MODELS out of a star schema (or snowflake model) and one or more CUBES out of a cube model.

4. Validate the CUBE MODEL(S) and CUBE(S). If they are OK, deploy them to DB2. To do this, export the OLAP metadata to DB2. Now DB2 knows the metadata of the CUBE MODEL(S) and the CUBE(S) just designed, which enables DB2 to correctly map the cube's dimensions and measures to the data mart multidimensional model.
5. You can now use the Optimization Advisor to generate the script that is necessary to create the MQTs for the data mart fact table. The data definition language (DDL) for creating the MQTs is then generated.

Attention: The Optimization Advisor is on the Database Explorer, not on Data Project Explorer!

6. Note that the Optimization Advisor does not create the MQTs. It only generates scripts (DDL) for the tables, one to create the MQTs and another to refresh them. You must run the script to create or refresh the MQTs. It is also possible to create MQTs with auto-refreshing contents (refresh immediate) depending on the type of aggregation that is used (applicable to SUM and COUNT only).

   DB2 now has summary tables that vastly improve the performance of the queries and is now well prepared to answer OLAP queries.

We now switch to the WebSphere Administration console, as seen in Figure 2-16 on page 33.

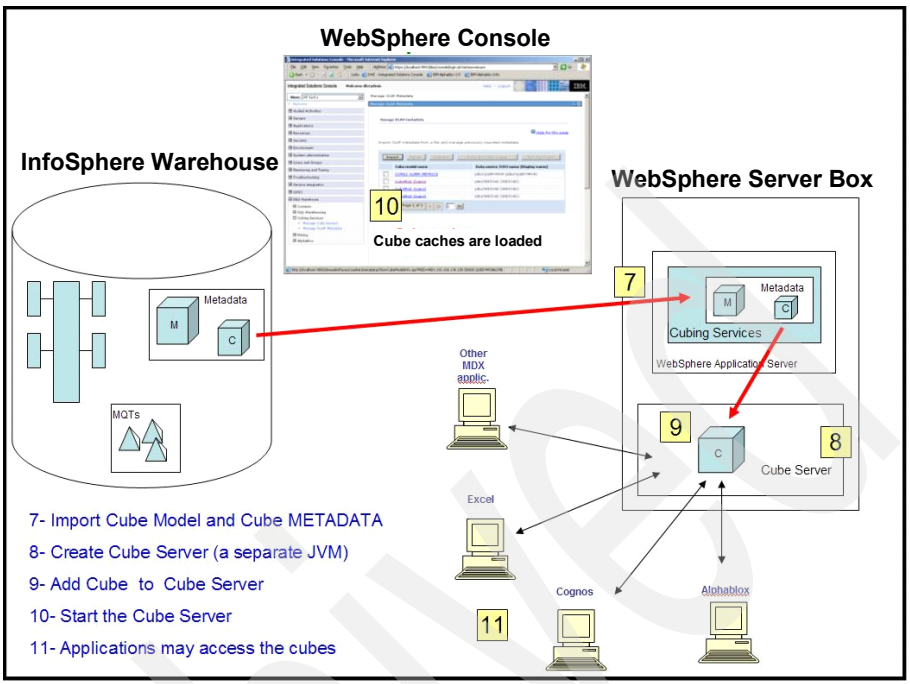


Figure 2-16 Creating a Cube - Part2

7. Now that the database is prepared for OLAP requests, you must configure the Cubing Services:

   a. Import the metadata from CUBE MODEL(S) and its CUBE(S) from DB2 to Cubing Services.

   b. On the WebSphere Console, click **DB2 Warehouse** → **Cubing Services** → **Manage OLAP Metadata**, and notice that Cubing Services now has all the information about the Models and Cubes that were imported.

8. Create one or more CUBE SERVERS. You must choose a port number for the Cube Server. When searching for a free port to use, keep in mind that Cube Servers use the port number selected, plus the following two consecutive ports. Each Cube Server is a separate JVM™, so it is not running under WebSphere. However, it must reside on the same server as WebSphere.

9. For each Cube Server, add the cubes you want to make available to external applications. On the WebSphere Console, click **DB2 Warehouse** → **Cubing Services** → **Manage Cube Servers**. You can assign one or more CUBES to a Cube Server, even if a cube is already assigned to another Cube Server.

10. Start the Cube Server. Because it is possible to have the Cube Server started and one or more of its cubes not running because of problem, make sure you click the Cube Server name (it is a link to more details) to verify that all cubes were successfully started. Data is loaded into cubing services caches so that they can provide users with fast query responses.
11. External applications can now connect to and use any existing Cubes on that Cube Server (provided they have authorization for that).

Stopping all Cube Servers: Before shutting down the WebSphere Application Server, stop all Cube Servers; otherwise, when the WebSphere is brought up again, and you try to start the Cube Servers, you get an error message because they appear to already be started. They appear this way because when a Cube Server is started, a file with the same name as the Cube Server, but preceded with a dot (.cubename) is created. As the Cube Servers are stopped, the corresponding files are erased.

If this happens, just delete those files, and start the Cube Servers. The default path is:

C:\ IBM \ dwe \ CubingServices

An in-depth description of the previous steps is included in the book *InfoSphere Warehouse and Cubing Services Client Access Interfaces*, SG24-7582. Included also in that book are the steps for accessing the Cubing Services data using Excel, Alphablox, and Cognos clients.

2.2.3 Accessing cubes

After the cube starts, it is available to external applications. Applications can connect to the cubes and start sending MDX queries. If the data needed is not in data cache, the Cube Server generates one or more SQL statements to the star schema MQTs to retrieve the information, as shown in Figure 2-17 on page 35.

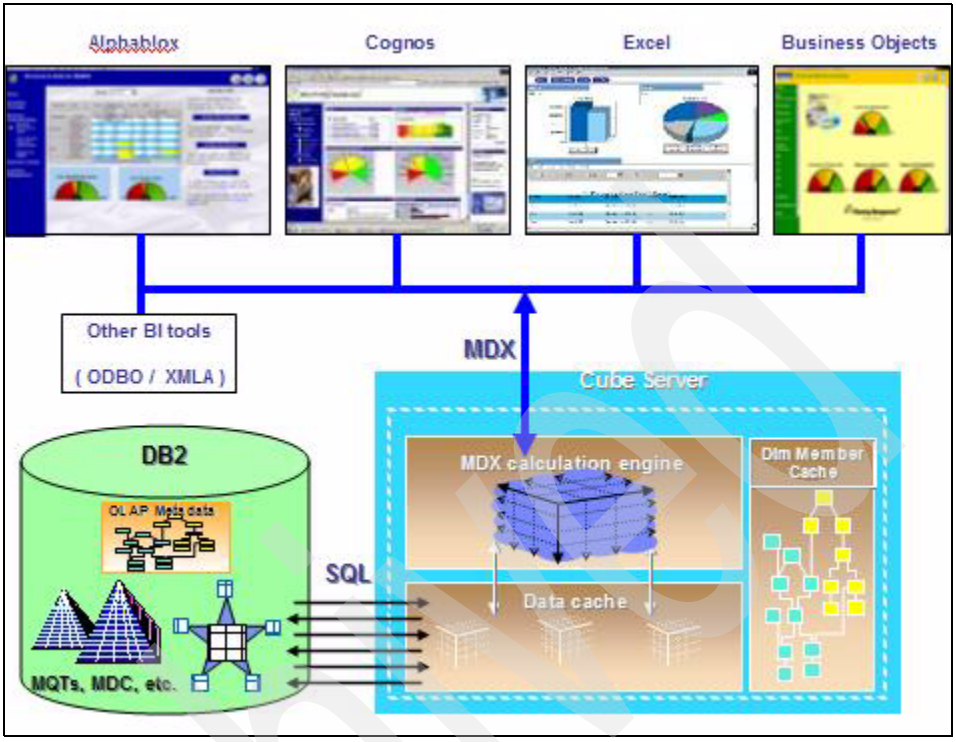


Figure 2-17 Accessing Cubes

The Cube Server uses the MDX query language, which stands for *Multi-Dimensional eXpressions* and is a query language for accessing multidimensional (OLAP) databases, just as SQL is a query language for accessing relational databases.

ODBO (OLE DB for OLAP) is an application programming interface (API) specification by Microsoft for the Windows® platform that became an industry standard. MDX is part of the ODBO specification and uses the Component Object Model (COM).

XMLA is a service standard for the Web. It is similar to ODBO, but uses an XML-based API between the application and the Web service. XMLA is an attempt to define a standard API for OLAP. The multidimensional query language used by XMLA is also the MDX.

When a Cube Server is created, you must define a port number, which is the User port, and MDX queries and navigation requests use it.

But aside from that base port, the next two consecutive port numbers are also used. The first one is the Admin port, used for administrative commands, and the second one is the HTTP port for XMLA connections.

Accessing cubes: A detailed description about how to access cubes using Excel, Cognos, and Alphablox is in the book *InfoSphere Warehouse: Cubing Services and Client Access Interfaces*, SG24-7582, in chapters 8, 9 and 10.

2.2.4 Cubing Services ports

When you create a Cube Server, it is necessary to specify the base port number. Do not choose an existing port number to avoid conflicts with other applications.

Ideally, you declare all ports that are in use in the Services configuration file, but it is not mandatory. So you can certainly find some ports in use that are not listed in the Services file.

You can verify the ports that are currently in use in a system by querying the operating system using the **`netstat -a`** command. The output shows the port number or the service name in use, so you can refer back to the Services file to find an empty range of three consecutive addresses for each defined Cube Server. The Services file are on:

```
C:\ Windows \ system32 \drivers \ etc \ services (Windows)
/ etc / services (Linux, UNIX)
```

Figure 2-18 shows the Cube Server ports that are required.

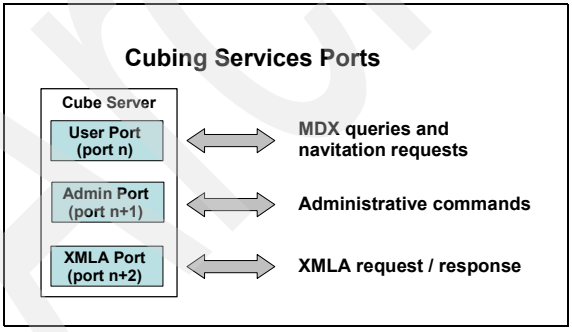


Figure 2-18 Cubing Services ports

2.3 Multidimensional life cycles

The life cycles for data marts, Cubing Services, and Analytics (Cognos) have different requirements, yet they must interact with each other.

2.3.1 Life cycle examples

In the book *Dimensional Modeling: In a Business Intelligence Environment*, SG24-7132, a life cycle for the dimensional model design was presented. It consisted of four phases that were identified:

1. Requirements gathering
2. Grain definition
3. Dimension definition
4. Facts identification

Each of those phases consisted of many steps to ensure that a good dimensional design resulted.

In the book *InfoSphere Warehouse: Cubing Services and Client Access Interfaces*, SG24-7582, similar steps were taken to assure that it was complete. The recommended steps were:

1. Identify business requirements
2. Design cube models and cubes
3. Validate cubes with business users
4. Deploy cube models

Similarly, you can configure the front-end tool to show Cubing Services data properly, for example, when using Cognos V8.4, all you do is configure a Cubing Services data source with the Metadata Wizard and use a special connection type: IBM InfoSphere Warehouse cubing services (XMLA), which creates a link from the Cubing Services cube metadata to Cognos (note that the metadata is not copied to Cognos) and a package is created at the end of the process. The package makes the metadata available to users after the Publish Package wizard publishes it.

In addition, you must take into account that any changes needed because of a new business requirement, or for changes to one of the components of the chain (database, OLAP engine and front-end tool), might require changes in other components as well, for example, changes needed in a given cube model hierarchy might then require a re-evaluation of the dimensional model and re-creation of Cognos Packages.

Also remain aware that you might make changes to correct unanticipated problems, to simply or modify the design for a new query tool, to increase the overall performance, or to avoid some software or hardware limitations. In addition, changes in a client tool might require changes in the structure of a cube or perhaps in the design of the dimensional model.

That type of process information is a primary subject of this IBM Redbooks publication. It is presented as another life cycle that is focused on data analysis.

Now we have what we might call a *Life Cycle Trilogy* for a Cubing Services environment, which Figure 2-19 shows. For more information about each of those Life Cycles, refer to the particular IBM Redbooks publication for each.

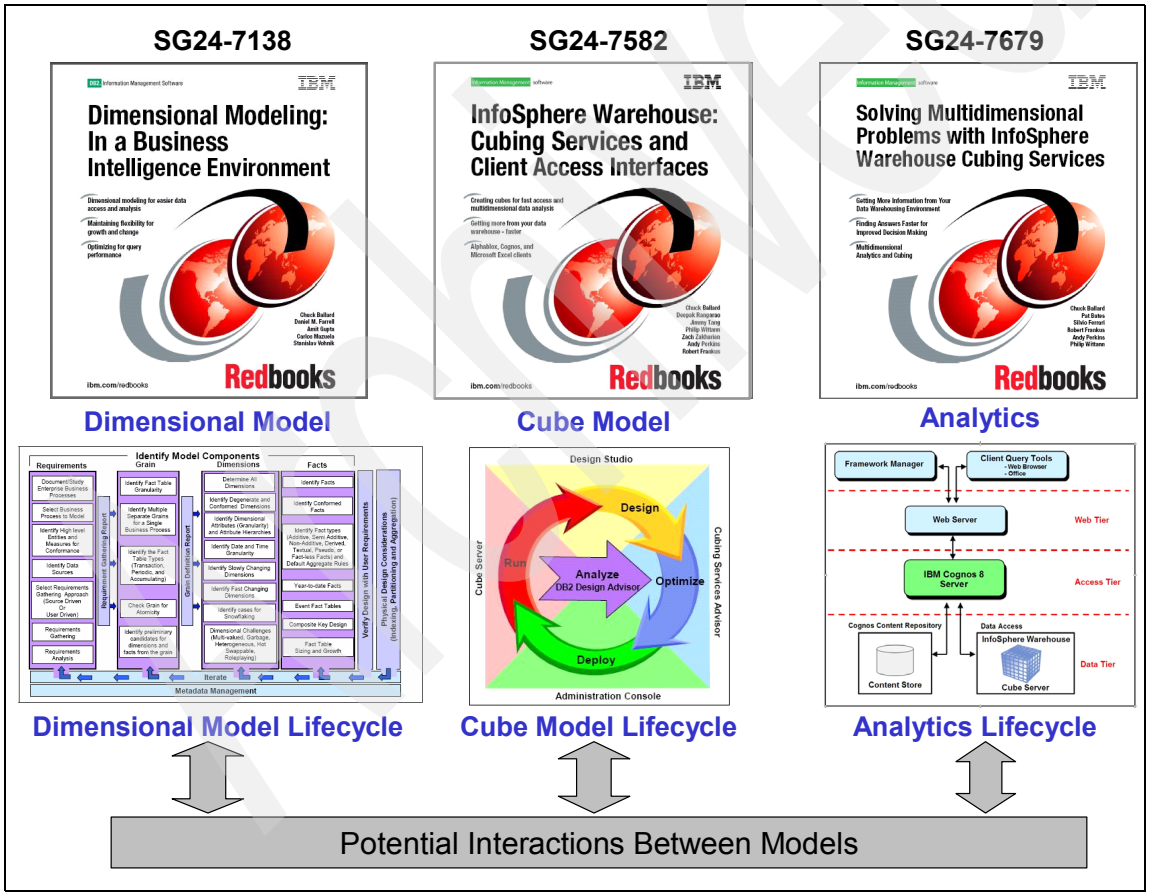


Figure 2-19 The Life Cycles

2.4 Designing for performance

Even though Cubing Services helps to create MQTs to improve the query response time, the physical design of the dimensional model can have a big impact on performance. In this section, we present some best practices in keeping a good overall performance.

2.4.1 Small fact tables

You can use a fact table for one or more Cube Models, which in turn, you can use to create one or more Cubes. This fact does not mean that we must have a single, big fact table, with all imaginable dimensions (keys) at hand, to allow many Cube Models to be created out of it.

Suppose we have a fact table with six dimensions (and some columns for measures) with millions of rows, and that we must create a new cube, with five dimensions already in that fact table, plus a new one with only 12 distinct members. The number of rows a fact table has depends on how many different primary key value combinations actually exists. Adding this new key column to that fact table can potentially increase the row count by 12, if every new possible combination exists for each of the original keys.

Because of that huge increase in the fact table's size, any query sent to it is likely much slower than before. Also the MQTs created for the original Cube Models take a lot longer to refresh because the RDBMS now must perform a GROUP BY clause for all six original keys to retrieve the same numbers from the new, expanded fact table.

It might be easier and faster to create a new, independent fact table and end up having two six dimension fact tables than to have a single fact table with seven dimensions. They most likely have a combined size that is much smaller than that single seven dimensions fact table. Try to use as few dimensions in fact tables as possible.

Adding more columns for measures does not have a big impact, though.

2.4.2 Logs

During the test phase, performance tuning, or problem debugging, you can turn on some logs. Use the WebSphere console to manage the Logs:

- DB2 Warehouse / Cubing Services / Manager Cube Servers
- Click the cube server name, and then select the Logging tab

See detailed instructions in *InfoSphere Warehouse: Cubing Services and Client Access Interfaces*, SG24-7582, chapter 6, section 6.3.7 Logging and Tracing. There is also related information in 2.4.3, “Production statistics in development environment” on page 40.

2.4.3 Production statistics in development environment

In a development environment, it is interesting to have the same table statistics as the production environment so that the DB2 optimizer can generate the same access plans in both environments. With similar table statistics, you can simulate the indexes and MQTs usage at production, in the development environment.

To collect the statistics, use the DB2s db2look utility in the production environment, with the **-m** option. The **-d** option lets you specify the database name:

```
db2look -d sample -m > stats.dml
```

Connect to the correspondent database in the development environment and run the script. Remember to edit the script to alter the database name or any other parameter needed before running it.

```
db2 -tf stats.dml
```

To restore the real statistics, just run the RUNSTATS utility for the desired tables on the development environment.

Also read related information in 2.4.2, “Logs” on page 39.

2.4.4 Limiting the optimization advisor

During the optimization advisor process, some user options are displayed. The first window (Target Queries) lets you select between generating DDLs for optimizing queries for Cubing Services (the default) or for ROLAP, and if you click the ellipsis (...)button in front of each cube, the Optimization Slices window is displayed, as shown in Figure 2-20 on page 41.

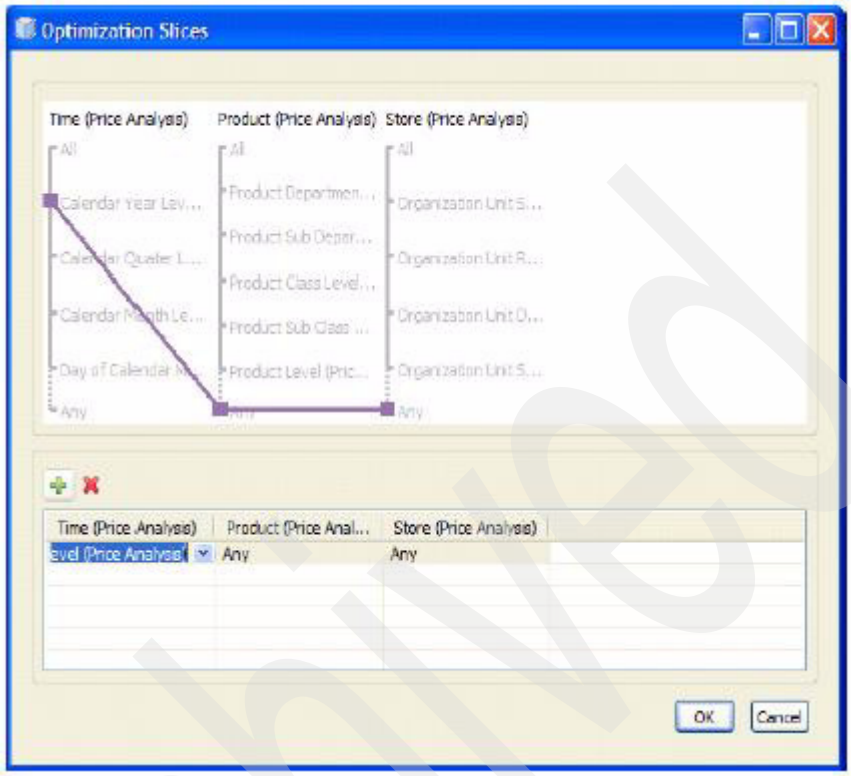


Figure 2-20 Optimization Slices

You can use these settings to define specific slices (the ones that users use most of the times). This process is shown in detail in the book *InfoSphere Warehouse: Cubing Services and Client Access Interfaces*, SG24-7582, chapter 5.4.2 Optimization Advisor in Design Studio.

However, setting those slices is only a concern if you impose any kind of constraints on the Optimization Advisor in the Limitations window, as shown in Figure 2-21 on page 42.

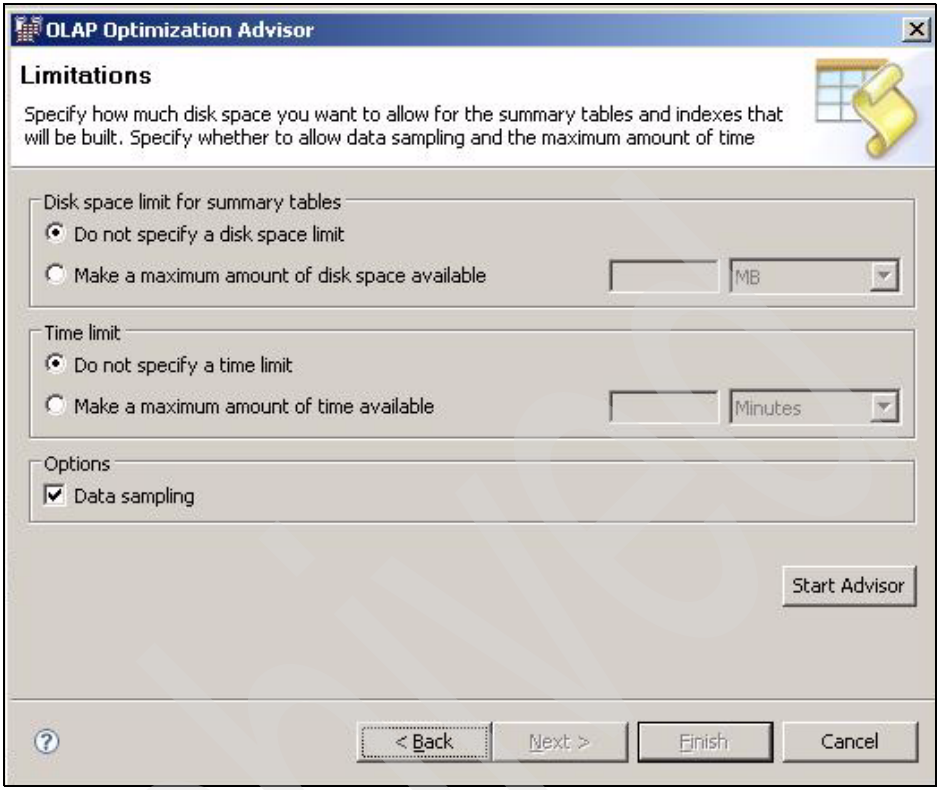


Figure 2-21 Optimization Advisor Limitations

If time or space limitations exist, the Optimization Advisor stops trying to create MQTs when one of those limitations is reached. If the important slices were selected, they are analyzed first and possibly have MQTs created for them.

If no restrictions are selected, the Cubing Services generate all of the MQTs it needs, which includes all existing slices.

2.5 Metadata

The Cubing Services metadata is stored in the InfoSphere Warehouse metadata database. The metadata database makes the Cubing Services metadata available for runtime access, which provides unlimited scalability, as shown in Figure 2-22 on page 43.

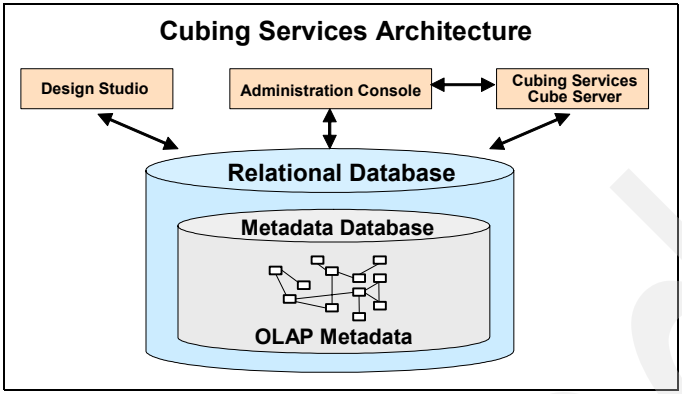


Figure 2-22 Cubing Services Metadata

Objects that are critical to the operation of Cubing Services are stored in the metadata database, which ensures the integrity of the data for a more robust and reliable system.

The metadata is accessible in the metadata database in read and write mode for all three defined clients: Cubing Services Cube Server, Design Studio, and Administration Console. The relational database allows multiple clients to read from and write to the metadata database, which ensures a consistent state for the metadata.

The metadata database for Cubing Services provides:

- A relational repository that multiple cube servers can access.
- A relational repository that provides you with all of the industrial-strength tools that are available in a database environment for things, such as transactional integrity, backup and restore operations, rollback to a consistent state, database replication, and so on.

2.6 Cubing services security

Cubing Services uses *role-based security* to control access to cubes and Cube Servers. A *role* is defined as a group of users that share the same security privileges. Access to a cube is granted or denied to a role rather than to a particular user. Role-based security simplifies security control because users do not need to be directly granted access to a cube. Instead, to grant privileges to a user, you need only to assign that user to the appropriate role.

Role-based security in Cubing Services involves two steps: authentication and authorization. The authentication process verifies the user identity and the role membership. The authorization process determines whether or not a user has the required privileges to access a specific cube.

By default, each cube is automatically associated with the role *Authenticated Users*, which by default holds the *Query Allowed* privilege, which allows users within this role to submit queries against a cube. You can assign one or more roles to cubes.

Cube security can be set to one of the following:

- Unrestricted access
- Access to authenticated users (default)
- Access to roles (subset of authenticated users)
- No access

To submit a query for cubes, the:

- Cube Server must validate the user in the DB2 instance (for example, LDAP) that the cubes access (Authentication)
- Cube security verifies if the user might or might not access it (Authorization)

To manage roles, configure cube security, and configure Cube Server security, you must have Administrator privileges.

2.6.1 Creating a role

To manage Cubing Services roles, you must have Administrator privileges. To create a role:

1. Open the Administration Console.
2. Select **DB2 Warehouse** → **Common** → **Users and Roles** → **Manage Cubing Services Roles**. The Console window is displayed, as shown in Figure 2-23 on page 45.

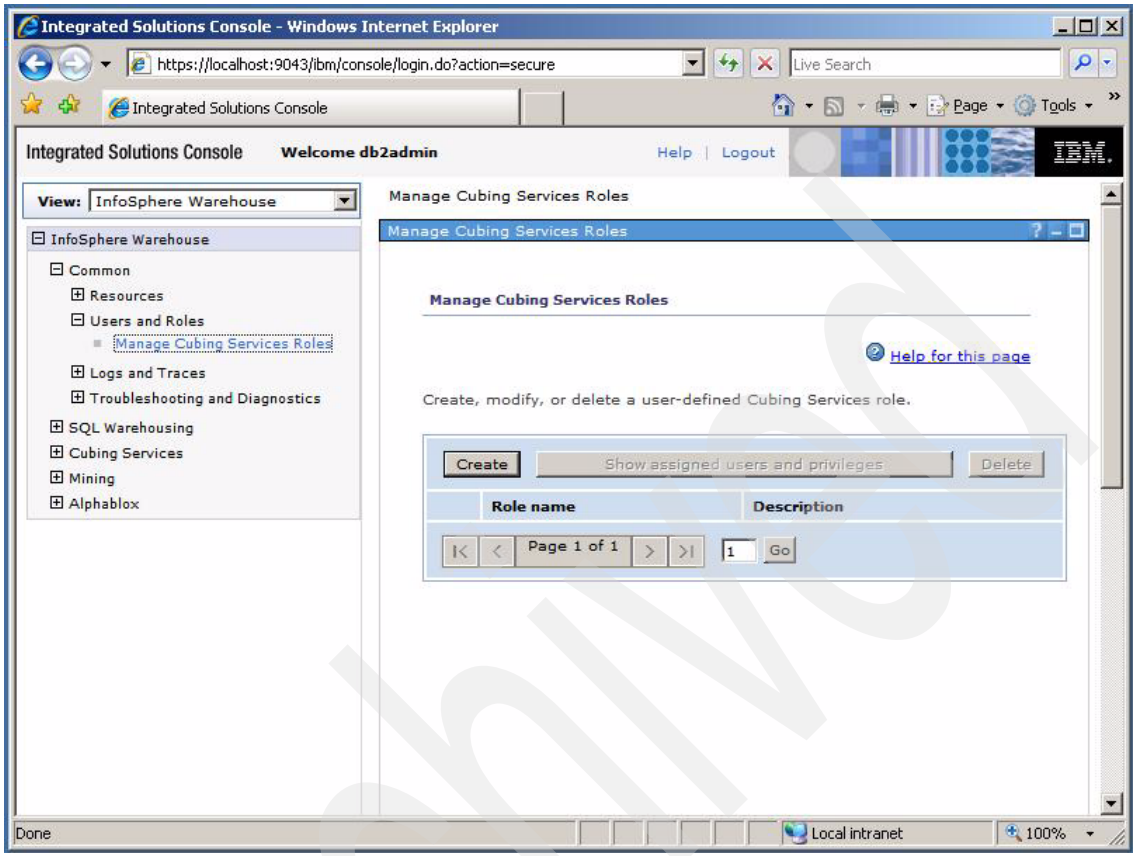


Figure 2-23 Manage Cubing Services Roles

3. Click **Create**, and enter the role name and description, and then click **Save**, as depicted in Figure 2-24 on page 46.

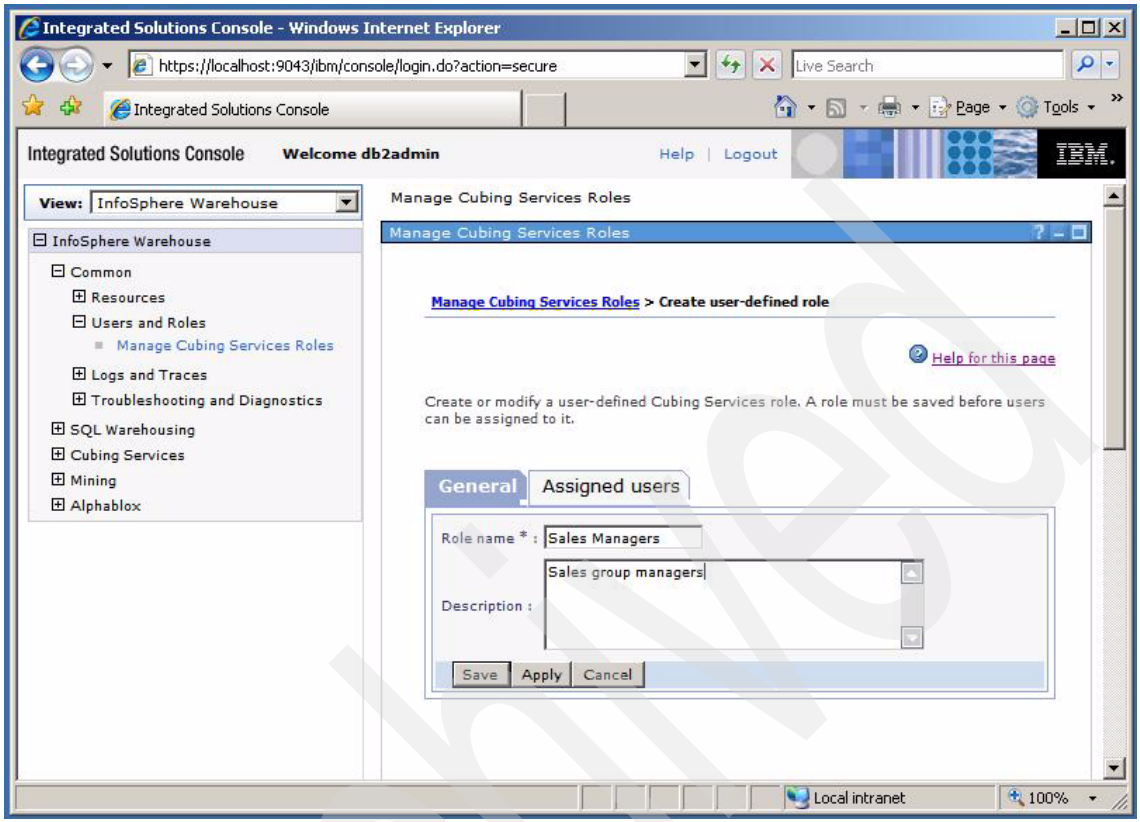


Figure 2-24 Create user Sales Manager

4. Click the link **Sales Manager**, to see a result, as shown in Figure 2-25 on page 47.

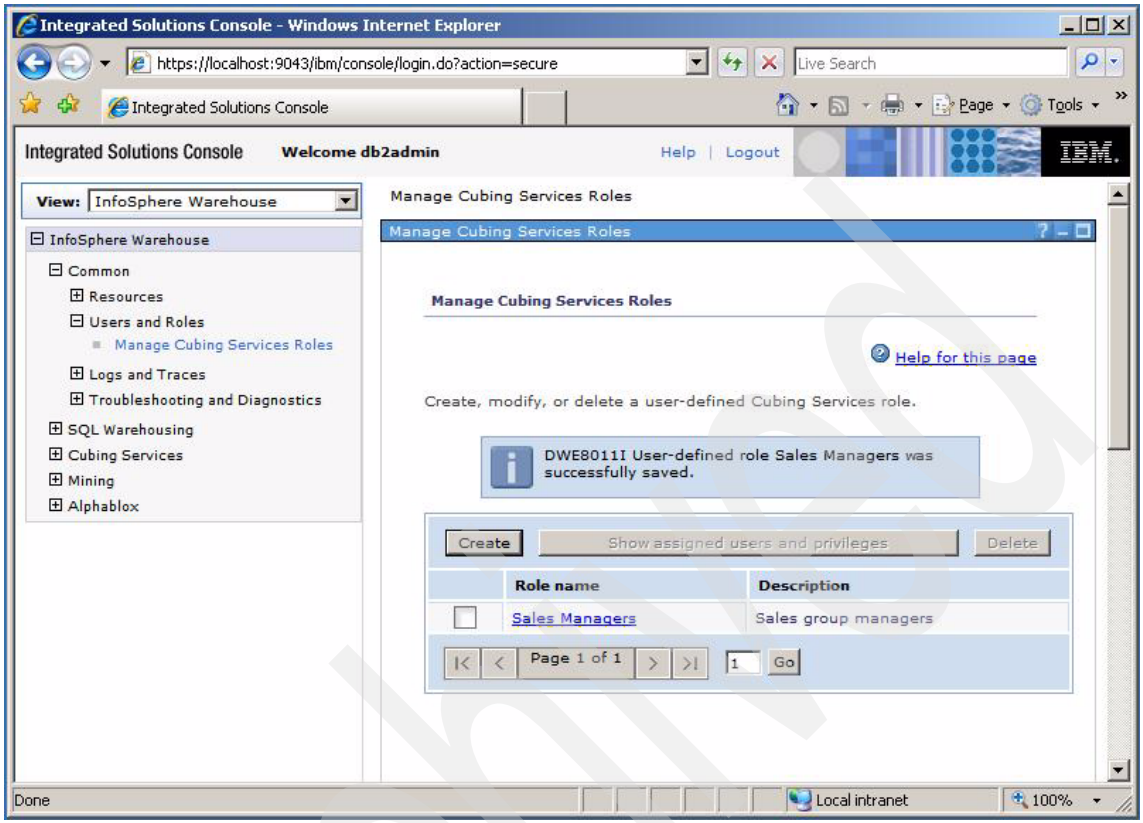


Figure 2-25 Saving definitions for user

5. Enter the user name, or select a list of comma separated user names, and click **Add** to see results similar to Figure 2-26 on page 48.

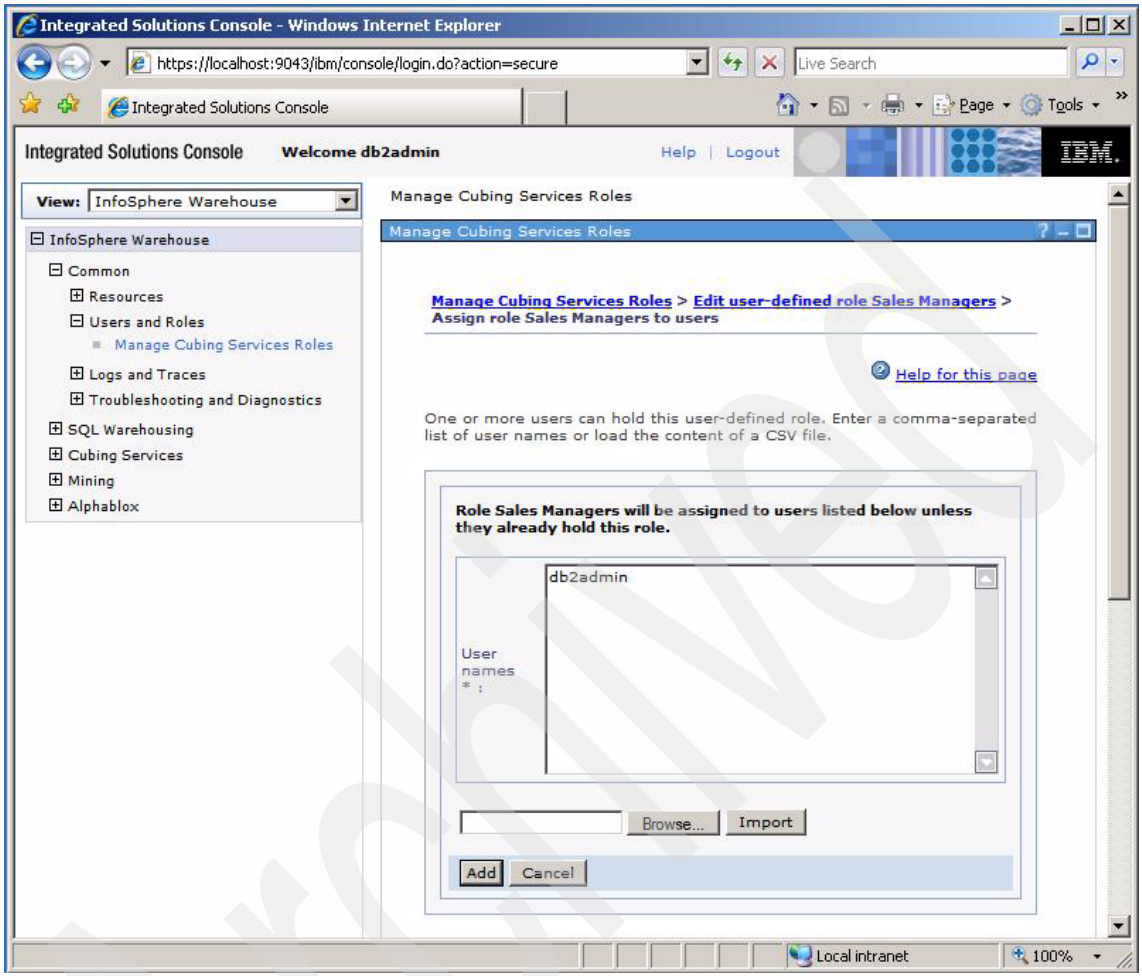


Figure 2-26 Add users to a role

2.6.2 Configuring cube security

Because you defined a role and a user, you can now configure cube security:

1. On the Integrated Solutions Console, click **InfoSphere Warehouse/Cubing Services/Manage OLAP Metadata** to show the Cube Models.

> **Enforced security:** The security is enforced at CUBE level, not at the Cube Model or Cube Server level.

2. Click the link of the cube model name to show the cubes that this cube model has, as shown in Figure 2-27.

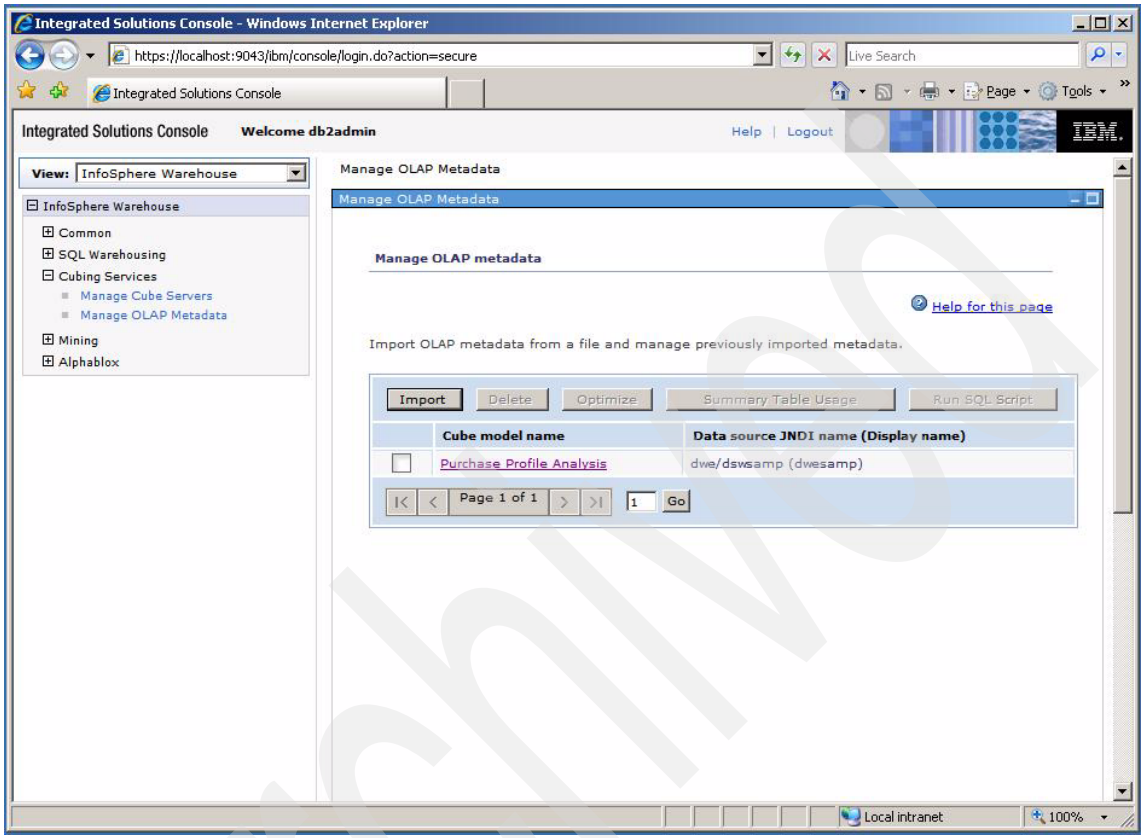


Figure 2-27 Manage OLAP Metadata

3. Click the link of the cube name to see the results, as shown in Figure 2-28 on page 50.

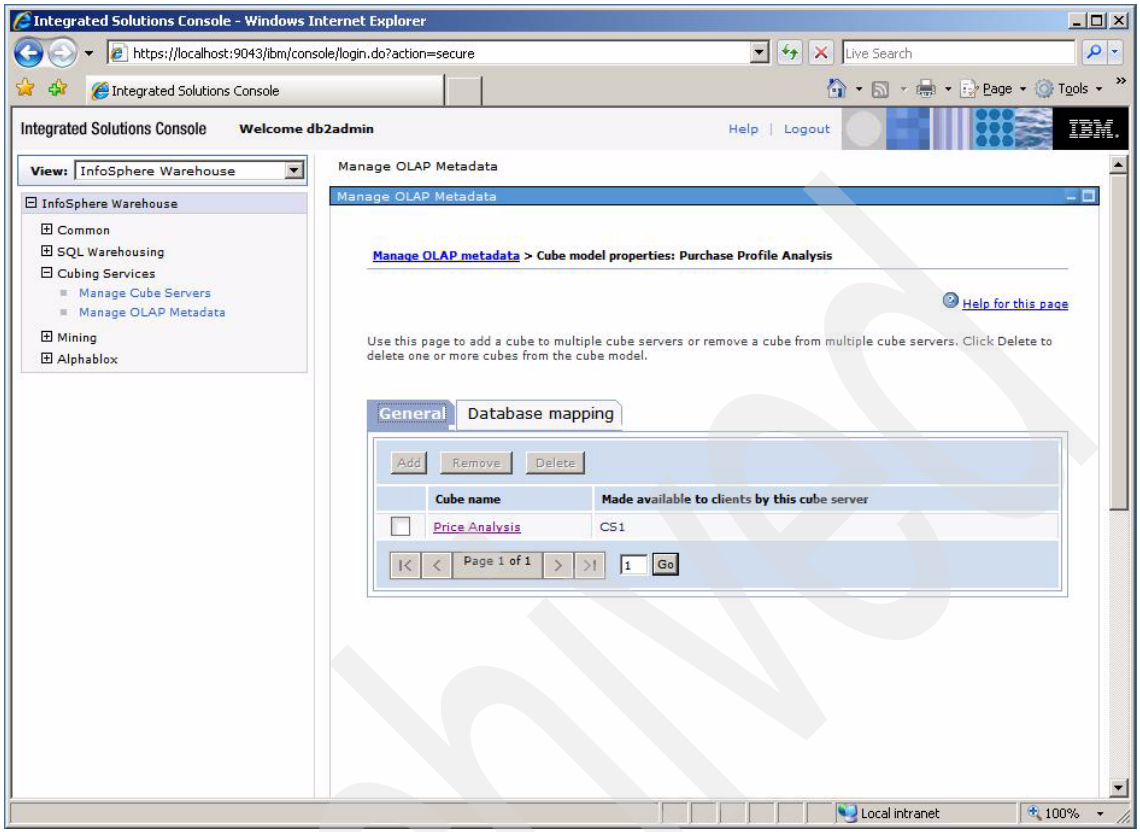


Figure 2-28 Cube mapping

Figure 2-28 shows the default setting for the cube, which indicates that all authenticated users are allowed to access it.

4. To add a role-based privilege, click **Add** to see the results in Figure 2-29 on page 51.

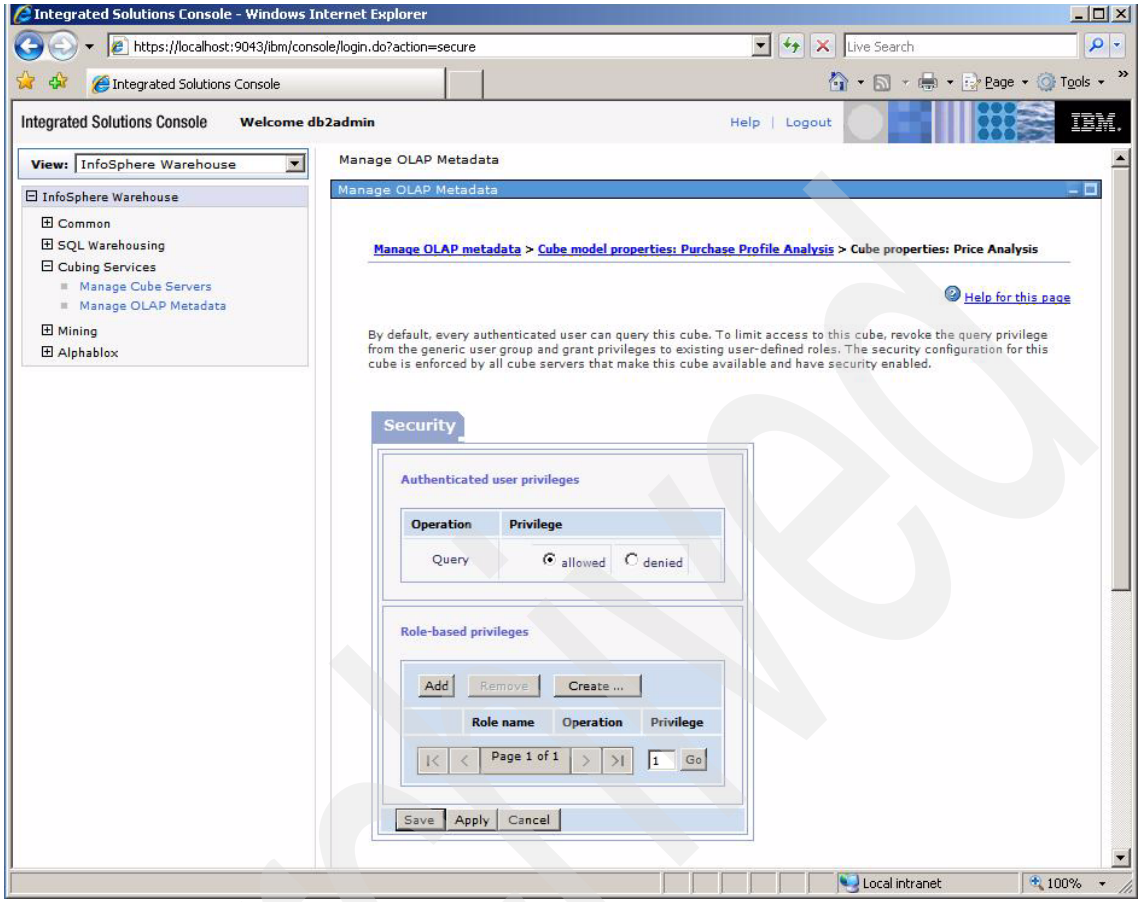


Figure 2-29 Cube security

5. Now you can add the user-defined roles to the cube, and click **Add** to see the results, as shown in Figure 2-30 on page 52.

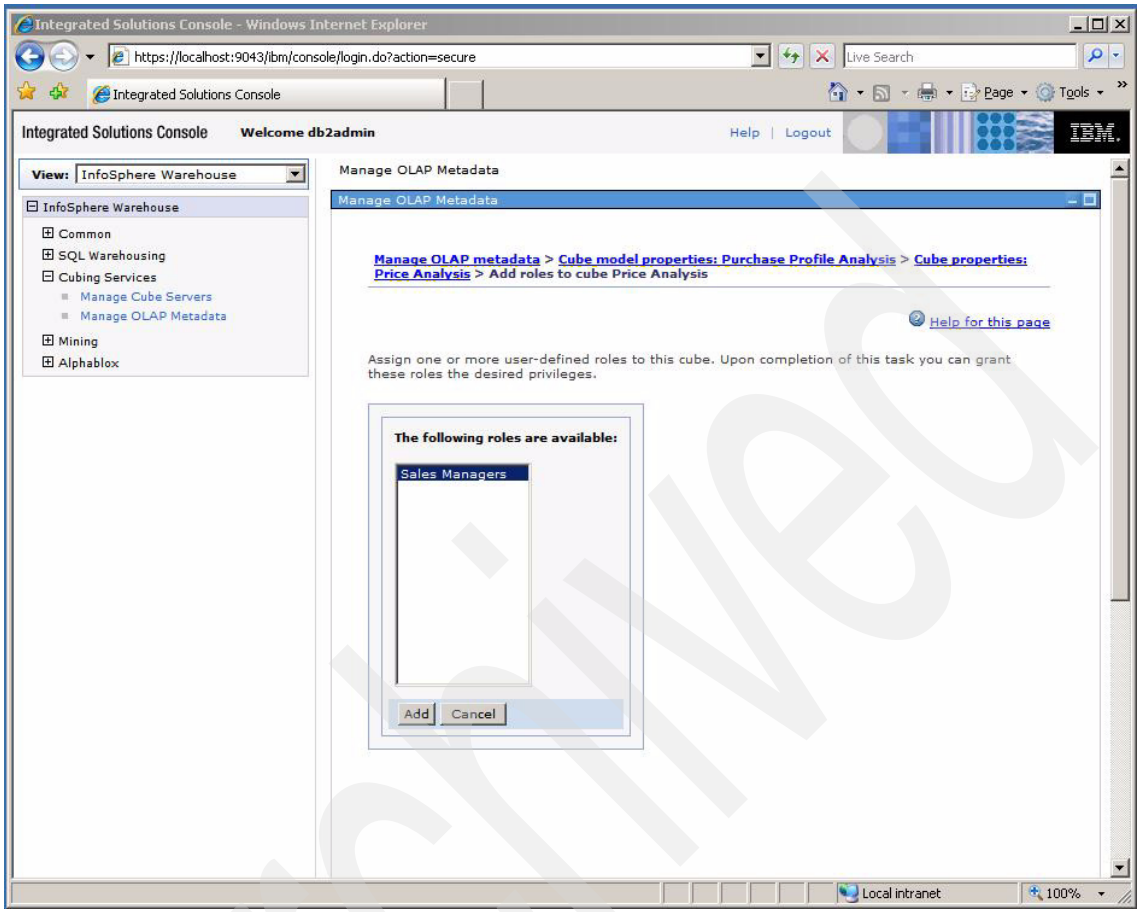


Figure 2-30 Add roles to cube

The selected roles are added to the cube, as shown in Figure 2-31 on page 53.

6. Switch the Authenticated User privileges to **denied**, and switch the desired role-based privileges to **allowed**, and then click **Save**.

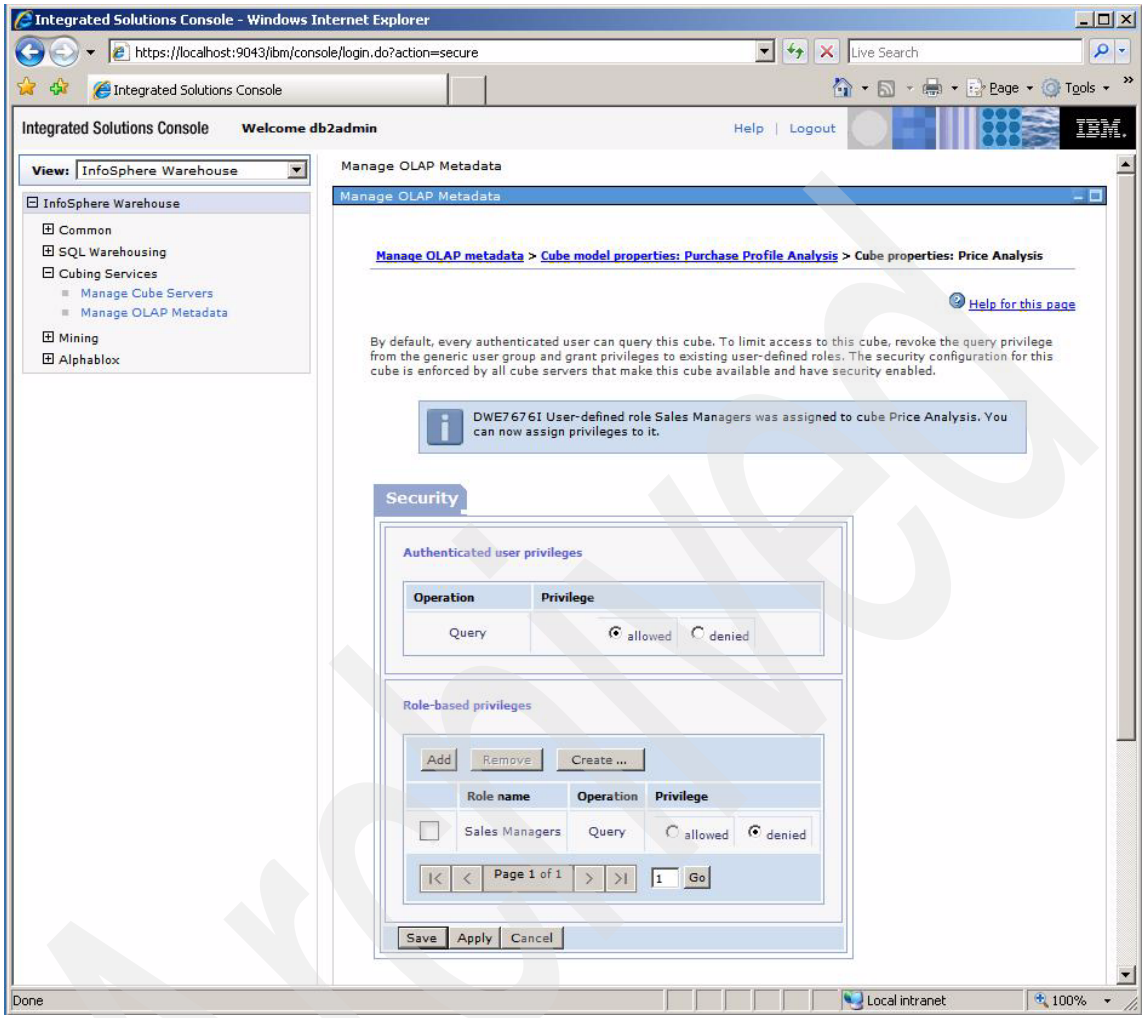


Figure 2-31 Roles added to the cube

The security configuration for cube Price Analysis was saved, as shown in Figure 2-32 on page 54.

Only authenticated users from selected roles are allowed to send queries to cube. So, make sure the other authenticated user privileges are set to denied.

Privileges: If a user has multiple roles and at least one of those roles is not granted a privilege, then the user does not hold the privilege because revoked privileges take precedence over granted privileges.

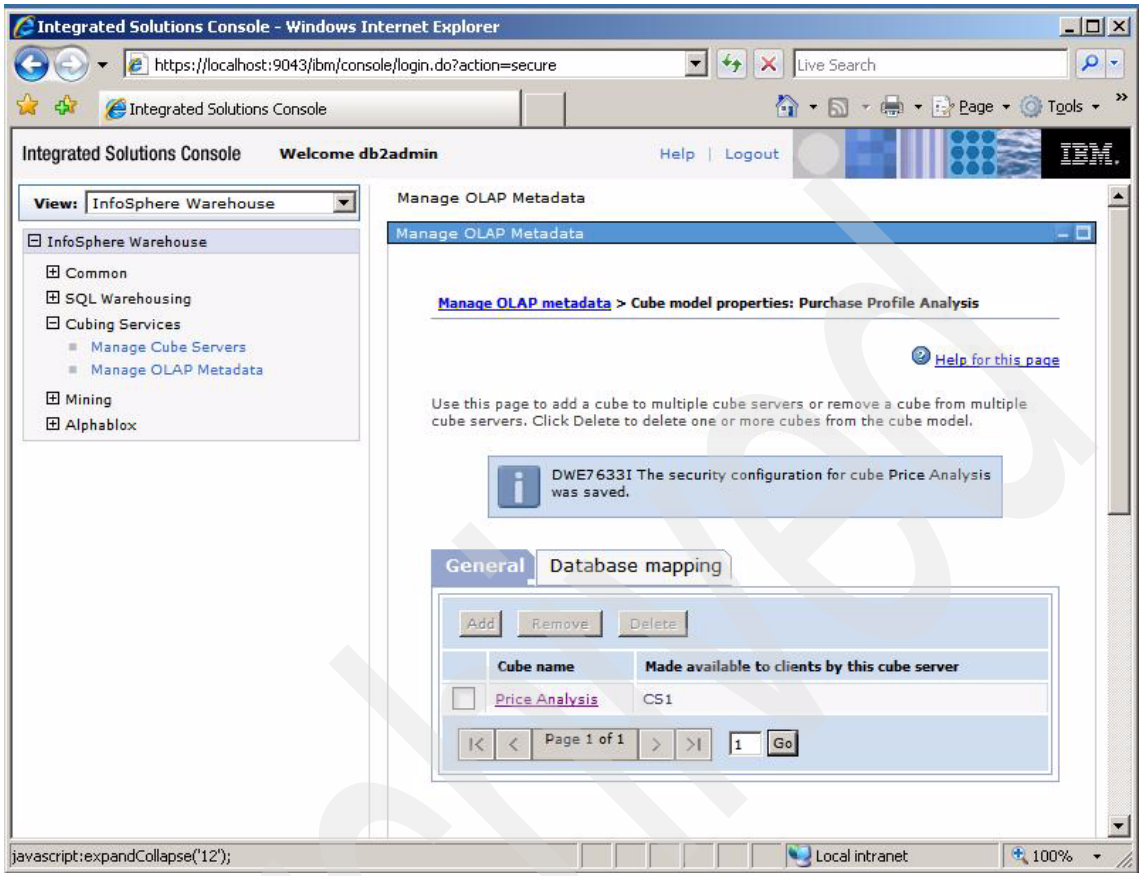


Figure 2-32 Security configuration saved

Security configuration:

- Role based: enabled by default
- Configurable for each cube: restricts query access
- Authentication performed by DB2
- Authorization determined by cube server

The cube security settings are:

- Enabled by default
- Only DB2 authenticated users are allowed to run MDX queries

3

The relational multidimensional model

In this chapter, we discuss and describe how the dimensional model is affected when optimized to operate with InfoSphere Warehouse Cubing Services.

3.1 Impact of cubing services on the relational model

Analyzing data is of vital importance for any business organization. The dimensional model is, with some possible exceptions, the place where data is extracted and turned into information. It is the preferred input for OLAP engines, such as Cubing Services. In today's fast moving business environment, answers to business questions are needed quickly for the agile organization. To achieve the desired level of performance, there are some best practices that you can apply to business areas, such as:

- Dimensional physical modeling
- OLAP related optimization
- Relational data optimization

3.1.1 Dimensional modeling best practices

The dimensional model defines the data that business users analyze. As an example, Cubing Services reads the data from the dimensional model during the cube start-up and every time the data needed by a query is not already in the data cache. The fact table, shown in Figure 3-1 on page 57, provides some guidelines for best practices that help in obtaining the best possible performance from a dimensional model.

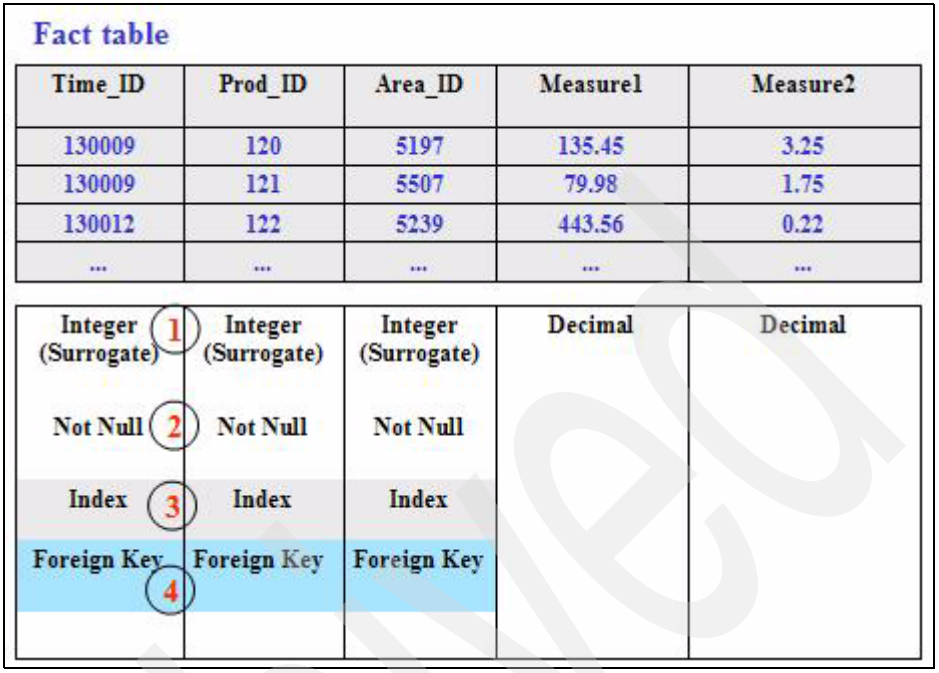


Figure 3-1 Fact Table impact on best practices

Fact tables can impact best practices. The highlighted points in Figure 3-1 are examples of best practices:

1. Columns that are used for joining the fact table to the dimensions must use an INTEGER data type for best performance. You can use an artificial (surrogate) key if the natural key is of a different data type.
2. The specified columns are created as NOT NULL to ensure that queries can be routed to the summary tables.
3. Define an index for each of the specified columns.
4. Always define a Foreign key from the specified columns to their related dimension table, either as enforced or as an informational constraint. Informational constraints provide a way to improve query performance without increasing maintenance costs. The DB2 SQL optimizer can use these constraints, but the database manager does not enforce them.

Foreign key definitions: The foreign key definitions are a pre-requisite for the Cubing Services Optimization Advisor, either enforced or not enforced. If they are not present, the advisor shows a warning message because without constraints, the chances of queries getting re-routed to MQTs are decreased.

Dimension tables can also impact best practices, as shown in Figure 3-2. The highlighted points are examples of best practices.

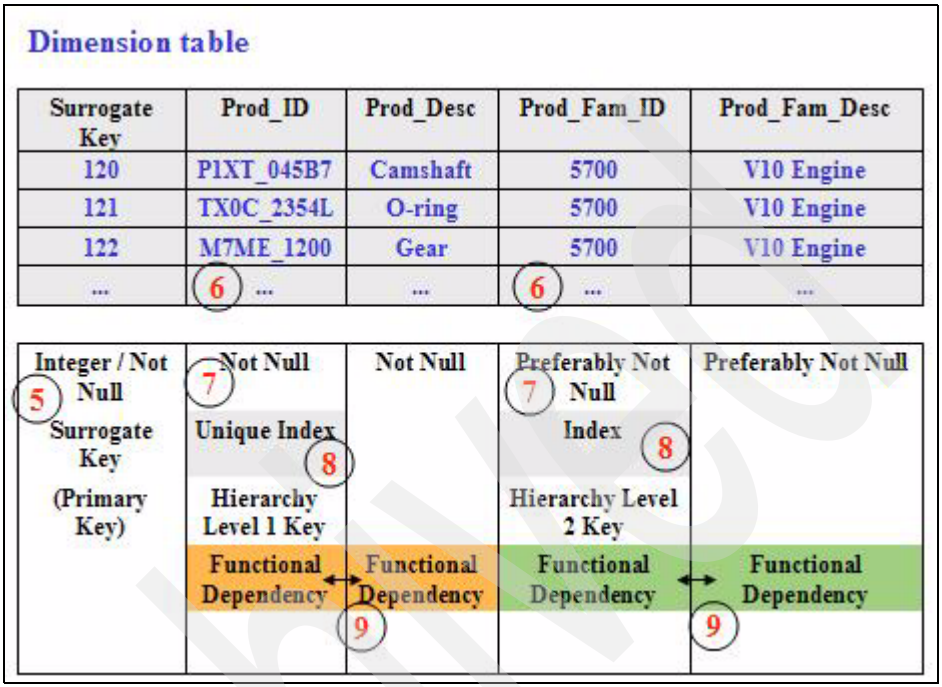


Figure 3-2 Dimension Tables impact on best practices

5. The key column must be identical to the corresponding column in the fact table. It is, preferably, an INTEGER (possibly a surrogate key), and defined as a primary key (so no null values are allowed).
6. Level Key columns must be integers because they are often used in grouping and are more efficient than strings for this purpose.

Surrogate key: Figure 3-2 shows the use of a surrogate key after the dimension natural key, Prod_ID, is determined not to be an integer.

7. Avoid using nulls on columns that are used as level keys.
8. Each Level key must have its own index defined.
9. Define a functional dependency between a level key and the corresponding level description or attribute.

Functional dependency

A functional dependency is a way to tell the Cubing Services Optimization Advisor that a level object's default attribute and related attributes are functionally determined by the levels key attributes. In other words, it indicates that the attribute(s) have a direct relationship with the level key. Using this functional dependency can help to minimize the MQT size (and speed) because it can now use the smaller Level ID column instead of the attribute's typically larger column definition. Cubing Services knows that they can be used interchangeably and uses the smaller column definition.

A functional dependency is not enforced by the database manager during normal operations, such as insert, update, delete, or set integrity. The functional dependency might be used during query rewrite to optimize queries. However, incorrect results are returned if the integrity of a functional dependency is not maintained.

You can define a functional dependency using the CREATE TABLE or ALTER TABLE statements, as shown here:

```
ALTER TABLE dim_product
ADD CONSTRAINT FD_prod_id_desc
CHECK ( prod_description DETERMINED BY prod_ID ) NOT ENFORCED
```

The Prod_ID column must be defined as NOT NULL, and the NOT ENFORCED option must be present.

For more information about functional dependency, search for *functional dependencies* in the DB2 Information Center, at the following location:

`http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp`

Additional material about best practices is in the Redbooks publication *InfoSphere Warehouse: Cubing Services and Client Access Interfaces*, SG24-7582. See section 12.1, "Best Practices for Star Schema.

Statistics

As always, an important activity is keeping the table of statistics up-to-date. To help the DB2 Optimizer select the best optimized access plan, collect statistics for the data and the indexes.

To collect statistics using the Control Center:

1. Select the tables desired, right-click one of them, and then click **Run Statistics**, as shown in Figure 3-3 on page 60.

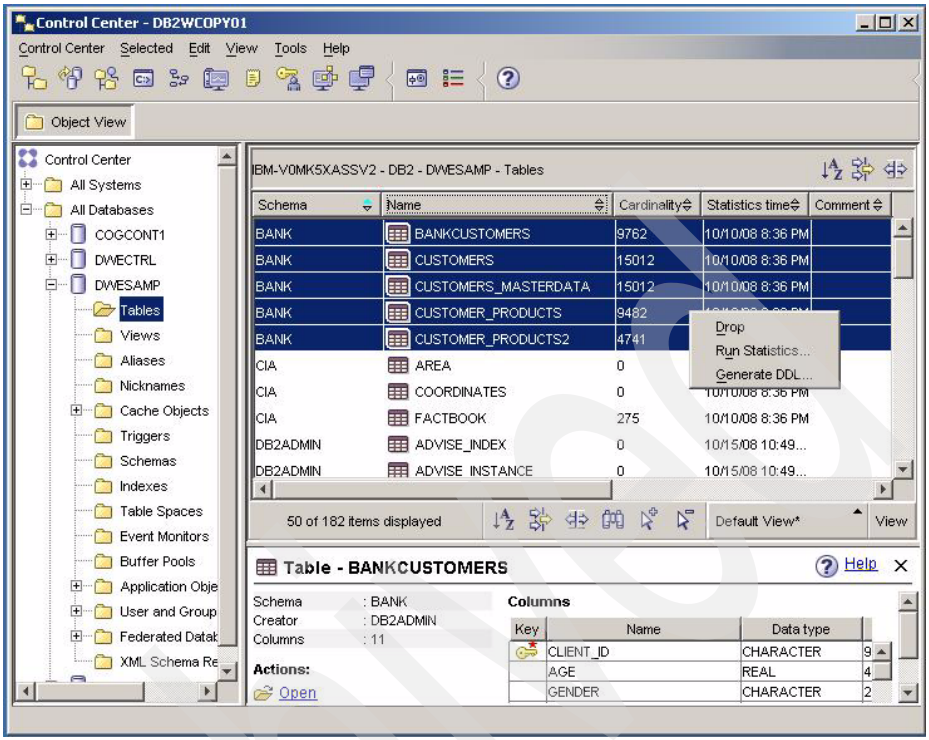


Figure 3-3 Selecting tables for statistics collection

2. Select the desired options, as shown in Figure 3-4 on page 61.

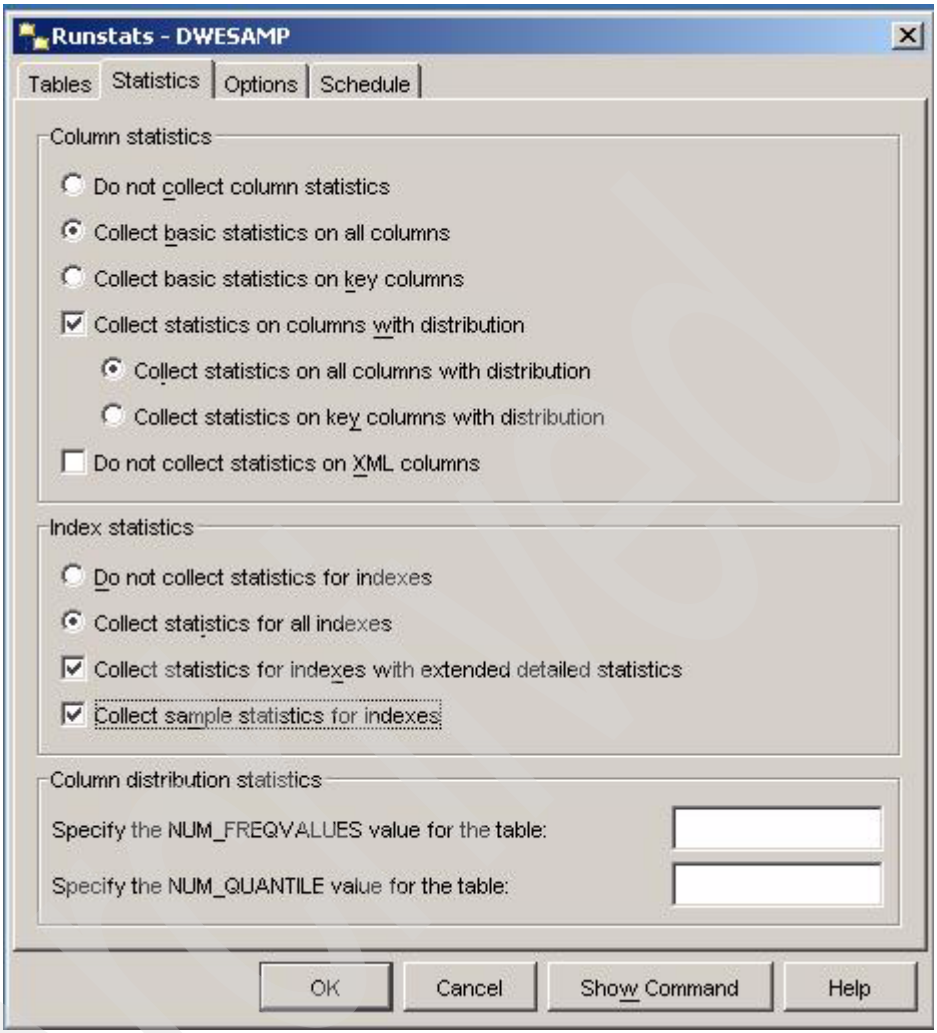


Figure 3-4 Statistics options

Select the options to collect detailed statistics for data and indexes, and perhaps also include the collection of sample statistics. Collecting the sample statistics can take quite a bit of additional time.

3. You can run the statistics by clicking **OK** to collect the statistics, or click **Show Command** to see the generated command. You can then copy and paste it in a file to be executed in batch mode, as shown in Figure 3-5 on page 62.

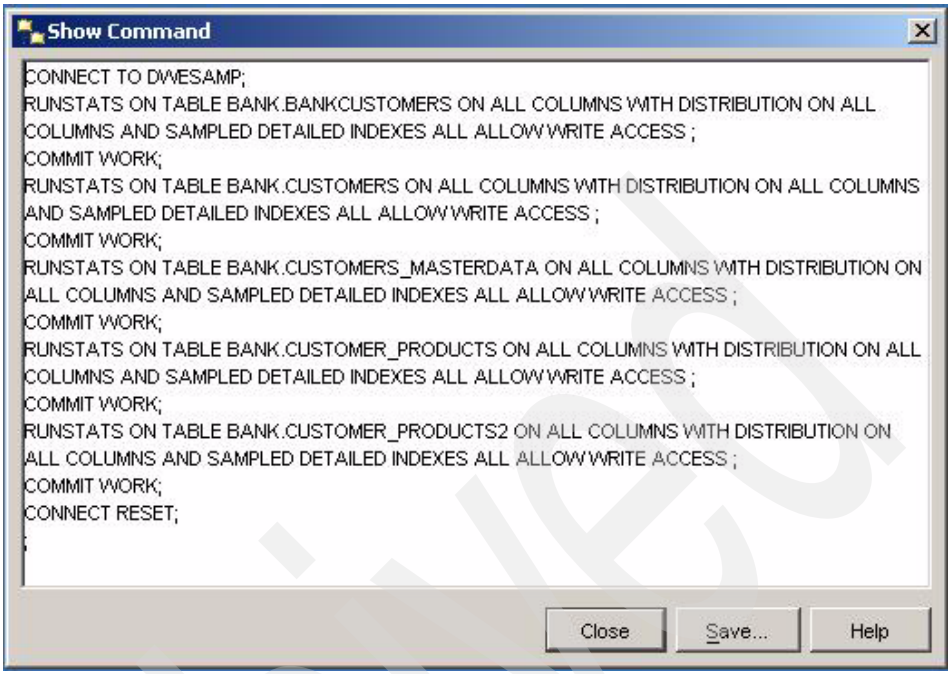


Figure 3-5 RUNSTATS command

Note: Databases that are created using DB2 V9.5 have the statistics automatically collected in the background, by default.

Column distribution statistics

Collect column distribution statistics when at least one column in the table has a highly non-uniform distribution of data and the column appears frequently in range or equality predicates. The distribution statistics are most useful for dynamic SQL, but also for static SQL that does not use host variables.

The *NUM_FREQVALUES* selects the *n* most common values for that column, and the NUM_QUANTILE specifies the number of sections into which the column data values are grouped, as shown in Figure 3-6 on page 63.

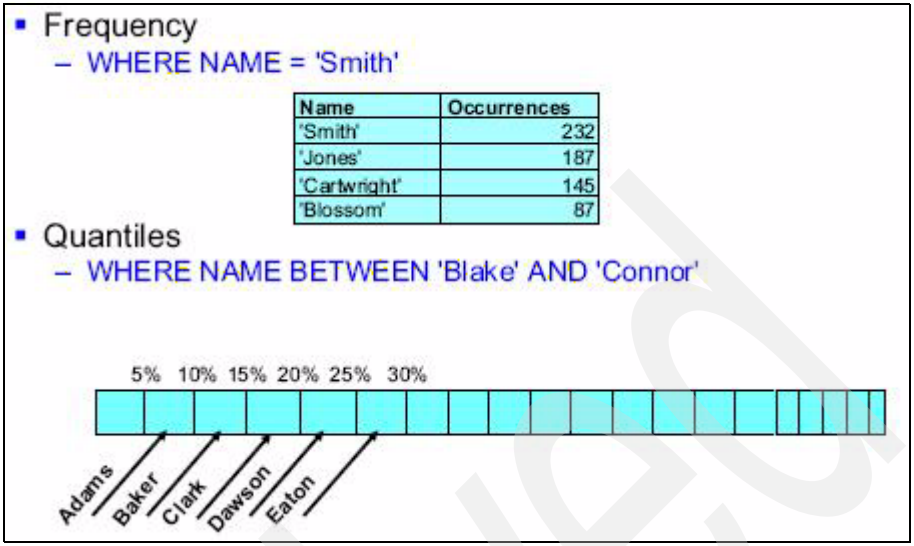


Figure 3-6 Distribution statistics

In Figure 3-6, the distribution statistics where collected for column NAME, and the four most common names are listed along with the corresponding number of occurrences. If an SQL query is to retrieve rows from that table where name = 'Smith', the DB2 Optimizer knows that exactly 232 rows will be returned and can decide that a table scan is better to use than an index on the NAME column due to the percentage of rows returned, particularly if the data is not well clustered according to the NAME column.

The default number for NUM_FREQVALUES is 10, and the available range is from 0 to 32767. The NUM_FREQVALUES is useful for equality predicates.

Also in Figure 3-6, the quantities can provide DB2 with better selectivity estimates when using range predicates, for example, suppose that the query issued was looking for all rows where NAME BETWEEN 'Blake' and 'Connor.' By looking at the statistics, you can see that 'Blake' falls between 'Baker' and 'Clarke' (second quantile), and 'Connor' falls between 'Clark' and 'Dawson' (third quantile), which allows DB2 to know that somewhere between 5% and 10% of the data in the table would be returned based on this predicate.

The default number for NUM_QUANTILES is 20, and the available range is from zero to 32767.

3.1.2 Cubing services optimization advisor

After an optimized dimensional model is built, the Cubing Services Optimization Advisor can improve the performance of most queries that are sent to that model to a great extent by helping to create MQTs for the fact table.

The Optimizations Advisor analyzes several factors, such as the:

- Multidimensional model
- Database statistics
- Purpose of the model (Cubing Services or other ROLAP tool)
- Cube models and cubes

The Cubing Services Optimization Advisor is a good starting point when trying to realize a performance improvement. Run it as a part of the initial cube models and cubes deployment.

The Optimization Advisor versus the DB2 Design Advisor: The Optimization Advisor suggests the creation of MQTs based on the cube models, the cubes, and the fact table data sparsity. The focus is the OLAP aspects of data, which is a key difference between the Cubing Services Optimization Advisor and DB2 Design Advisor. The DB2 Design Advisor analyzes the SQL workload that is sent to that model and might suggest new indexes and MQTs based on traditional relational aspects of data.

To get useful MQT suggestions from the Optimization Advisor, some configurations are required. As an example, create:

- An index for the dimension column of each of the fact tables.
- A constraint (foreign key) between each of them to the corresponding dimension table primary key. Those constraints are either enforced or non-enforced (informational).

An informational constraint is a rule that the database manager does not use, which means that data inserted in the fact table does not need to have a corresponding value in the dimension tables; however, the DB2 Optimizer uses that constraint to better define the data access plan strategy.

Materialized query tables

The materialized query tables (MQTs) are tables that contain aggregate data from the model's dimensional table. If data that is in the fact table is altered, for example by a delete, load, insert, or update, the related MQTs no longer have the correct numbers. They must be refreshed to become in sync with the fact table again. You can choose the refresh method at MQT creation time (or by an ALTER TABLE statement) using the refreshable table options of the

REFRESH IMMEDIATE or DEFERRED command. The REFRESH IMMEDIATE command instructs DB2 to automatically recalculate the values for the MQTs as soon as the fact table changes.

Not all MQTs can be set to REFRESH IMMEDIATE. Prior to DB2 V9.5.2, the only aggregations allowed in a refresh immediate MQT were SUM, COUNT, COUNT_BIG, and GROUPING (without DISTINCT), which means that if the aggregations used in an MQT definition are other than those mentioned, the synchronization does not happen automatically whenever the base table is altered. In those instances, you must manually refresh the base table or use a pre-scheduled script.

Having deferred refresh MQTs is not as practical as immediate refreshed ones, so the Optimization Advisor included in DB2 V9.5.2 was improved in two ways:

- Mixed MQT refresh method: Prior versions of the Optimization Advisor employed an all or nothing approach when choosing the refresh method for the MQTs. If any of the suggested MQTs do not have the REFRESH IMMEDIATE option, all of them have the REFRESH DEFFERED option. DB2 V9.5.2 now selects the proper refresh option for each MQT.
- Function rewrite: The Optimization Advisor now rewrites the AVG function (into SUM and COUNT) and COUNT DISTINCT so an MQT that has those functions can be set to REFRESH IMMEDIATE.

The functions that force the MQT into REFRESH DEFFERED include: the MAX, MIN, STDDEV, and CORRELATION aggregation functions.

With a good physical design and having detailed statistics collected and up-to-date, the Cubing Services Optimization Advisor can suggest the most efficient MQTs.

DDL scripts: The Optimization Advisor does not actually create the MQTs, but rather the DDL scripts create them. Therefore, you must run the scripts to physically create the MQTs.

3.1.3 DB2 Design Advisor

Although the Cubing Services Optimization Advisor tries to optimize the performance regarding the OLAP aspects (MQTs for data aggregations), the DB2 Design Advisor is a tool that can help with the relational aspects of the dimensional model, such as suggesting the creation of indexes, MQTs and MDCs, based on SQL statements that are sent to a particular database. These statements can be grouped together and submitted to the Design Advisor as a workload. The statements can be collected using a number of tools, such as DB2

Query Patroller, DB2 Performance Expert, or from the Cubing Services' own logs.

The Cubing Services Performance Logs captures cache hits and misses, the MDX query elapsed time, and the SQL query elapsed time. The SQL queries can be analyzed to verify if indexes are used or if they must be created.

The Cubing Services SQL Logs can be enabled so the queries submitted to DB2 are monitored. The MDX requests sent to Cubing Services whose data is not already in cache are translated to one or more SQL statements and sent to the fact tables and MQTs.

For more details about how to enable or disable the Cubing Services Logs, refer to the book *InfoSphere Warehouse: Cubing Services and Client Access Interfaces*, SG24-7582, section 6.3.7 "Logging and tracing."

The default location for the logs is <installation directory path> \ CubingServices \ <cube server name> \ logs. For a cube named CS1 that is installed on Windows, the path is:

```
C:\ Program Files \ IBM \ dwe \ CubingServices \ CS1 \ Logs \
CS1.SQLLog.txt
```

Open the SQL log, select the rows that contain the query statements, and then copy and paste them to a text editor. Remove everything but the statements. Save the statements to a file. You can keep the statement quotes, but be sure to substitute the period at the end of a line for a semicolon (;), so that the DB2 knows when the end of a command line is reached.

You can then use the file with the SQL queries as an input workload to the DB2 Design Advisor. The tool analyzes the supplied SQL statements and can recommend additional MQTs, new indexes, or MDCs to further optimize the database performance.

To invoke the Design Advisor:

1. From the DB2 Control Center, right-mouse click the name of the desired database, and from the list select **Design Advisor**.
2. On the Select performance features page, you can select the objects that you want it to create. The advisor suggests the creation of those objects only if it benefits the performance, as shown in Figure 3-7 on page 67.

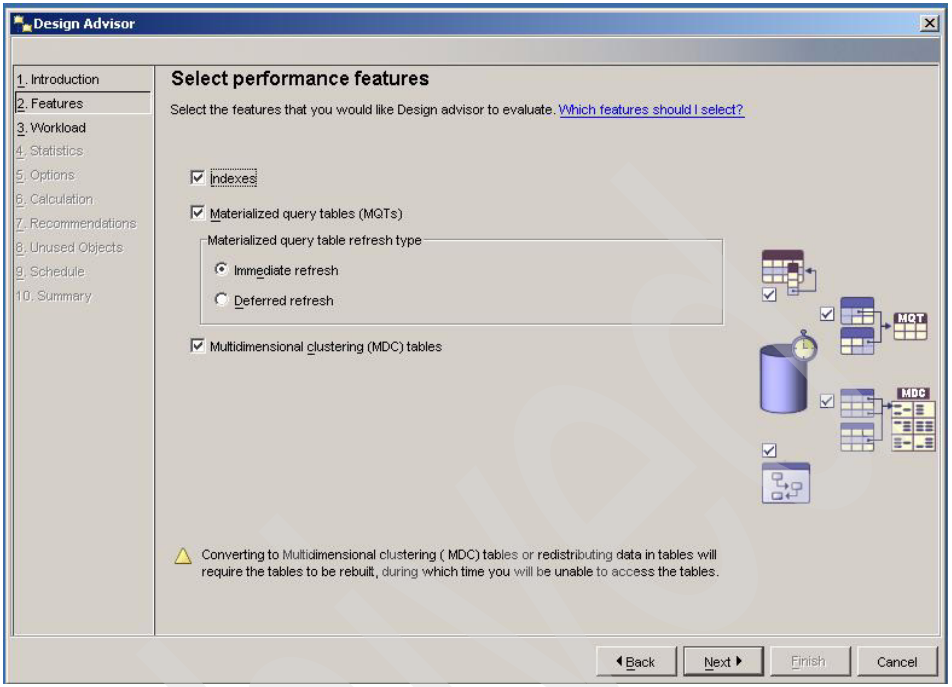


Figure 3-7 Design Advisor

For MQT and MDC basics, see chapter 6.3, “Impact of using MQTs and MDCs” on page 186.

3. Click **Next** to arrive at the Specify a workload page, as shown in Figure 3-8 on page 68, and then click **Create Workload**.

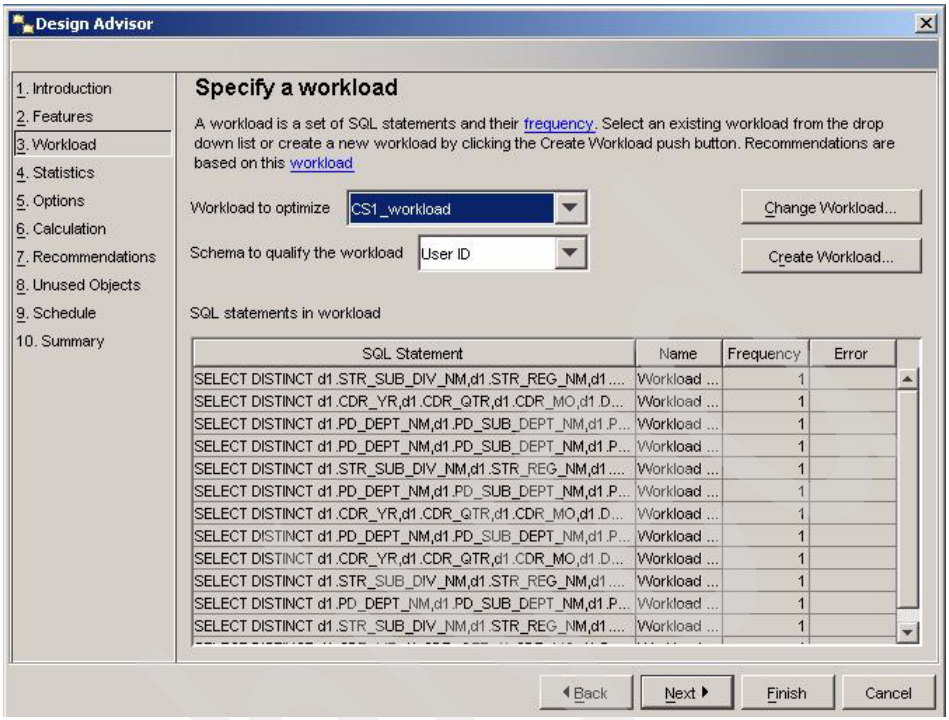


Figure 3-8 Specify a workload

4. On the Create Workload window, shown in Figure 3-9 on page 69:
   a. Enter a name for the workload.
   b. Select the desired schema for the workload.
   c. Click **Add** to manually input the queries, or more conveniently, click **Import**, and select a file with the SQL statements, as shown in Figure 3-9 on page 69.

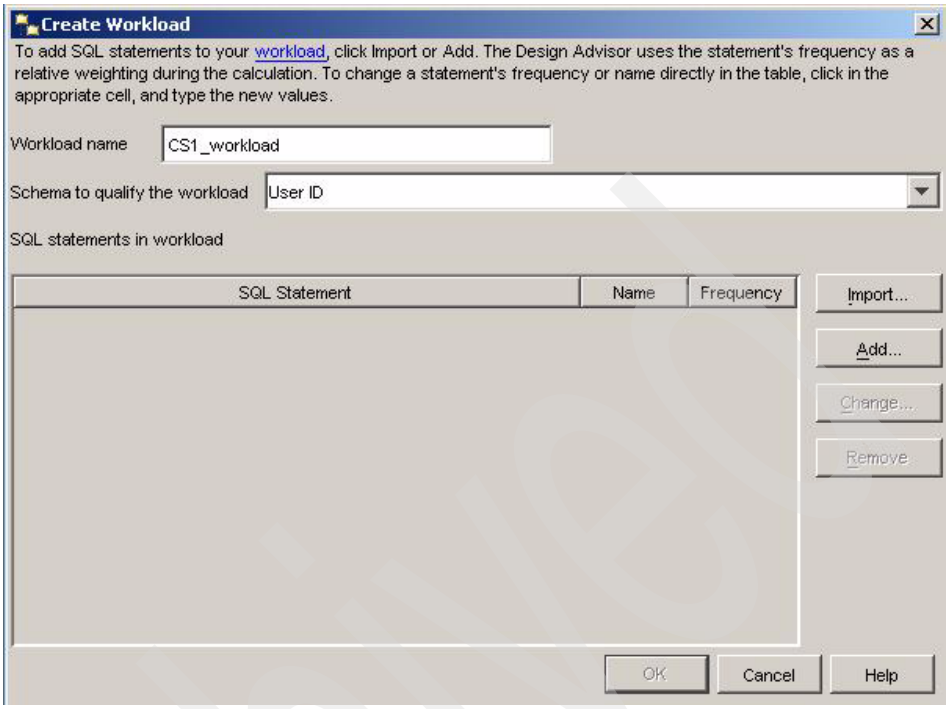


Figure 3-9 Create Workload

5. After you select the file name, click **Load File**. Select the statements in that file that you want to submit to the advisor, as shown in Figure 3-10 on page 70. Click **OK** twice.

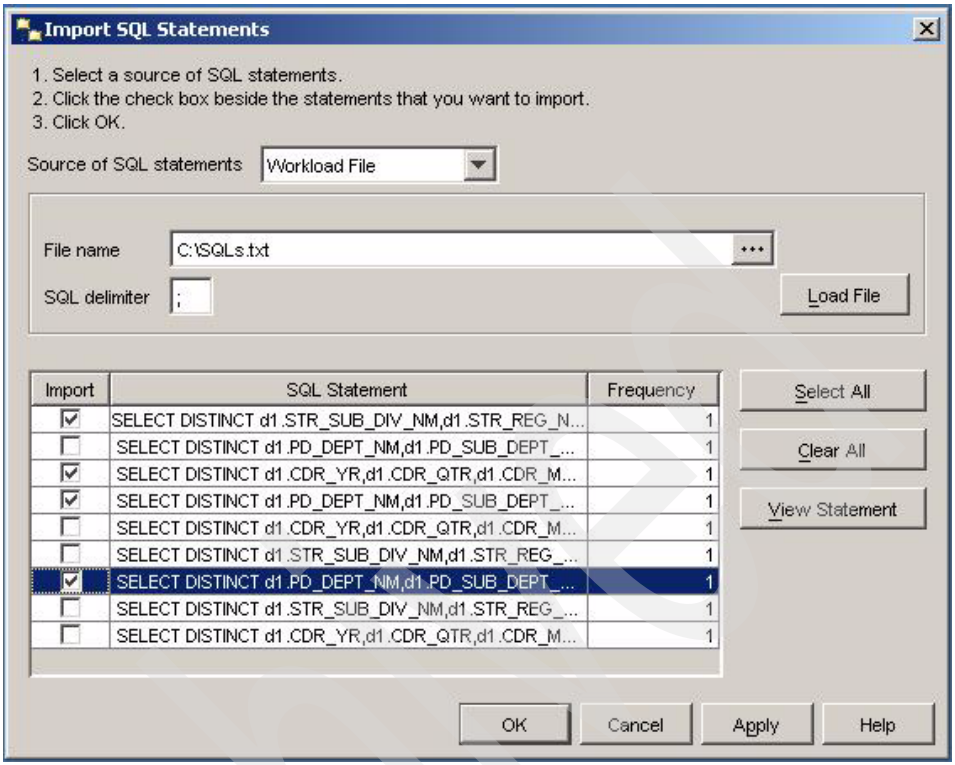


Figure 3-10 Import SQL Statements

6. If statistics are not up-to-date, select the tables for which you want to collect the statistics, and click **Next**, as shown in Figure 3-11 on page 71.

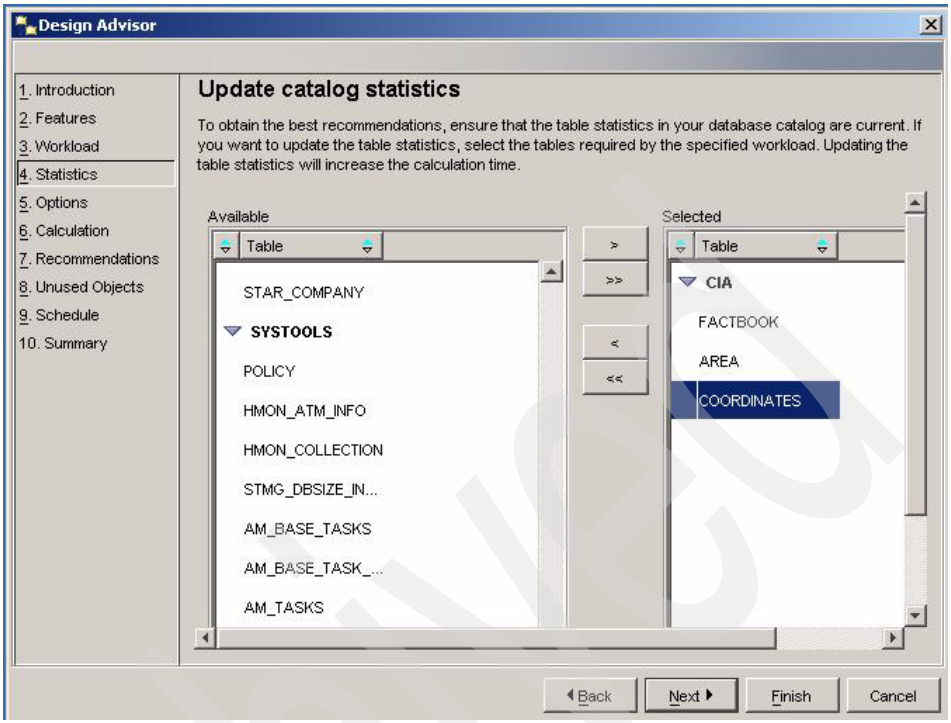


Figure 3-11 Update Catalog Statistics

The Design Advisor, like the Cubing Services Optimization Advisor, has options to limit the space that the new objects take on the disk. However, limiting the available space can limit the number of recommended objects.

Selecting the **Derive statistics by sampling** option is desirable for a more accurate estimate, but it might result in longer calculation times, as shown in Figure 3-12 on page 72.

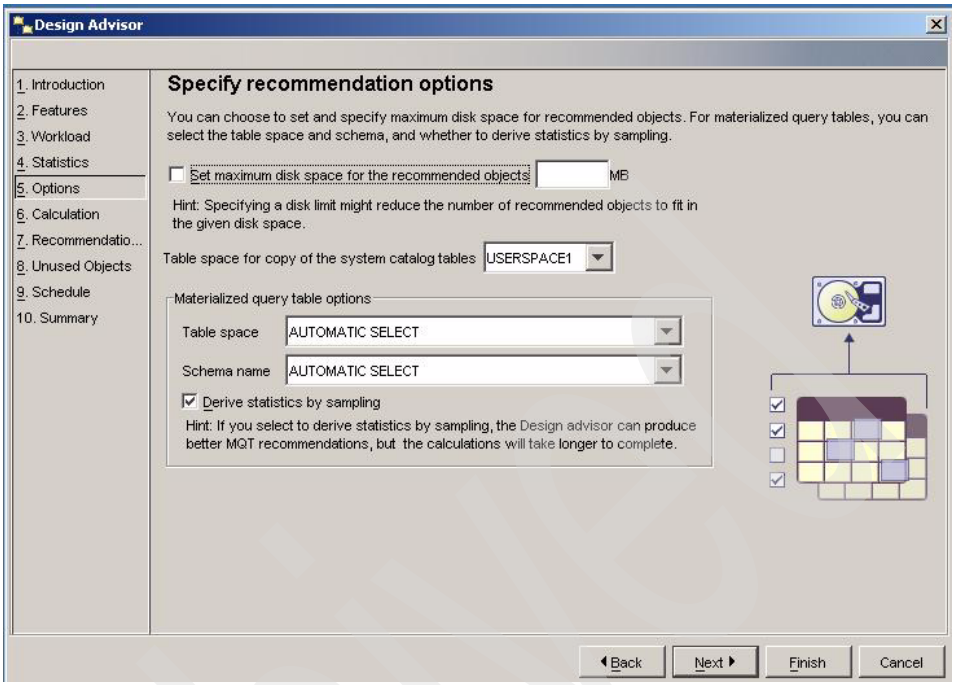


Figure 3-12 Specify recommendation options

7. You can specify when to run the optimization and also the maximum calculation time allowed. Select **Run now**, and the calculation starts when you click **Next**, as shown in Figure 3-13 on page 73.

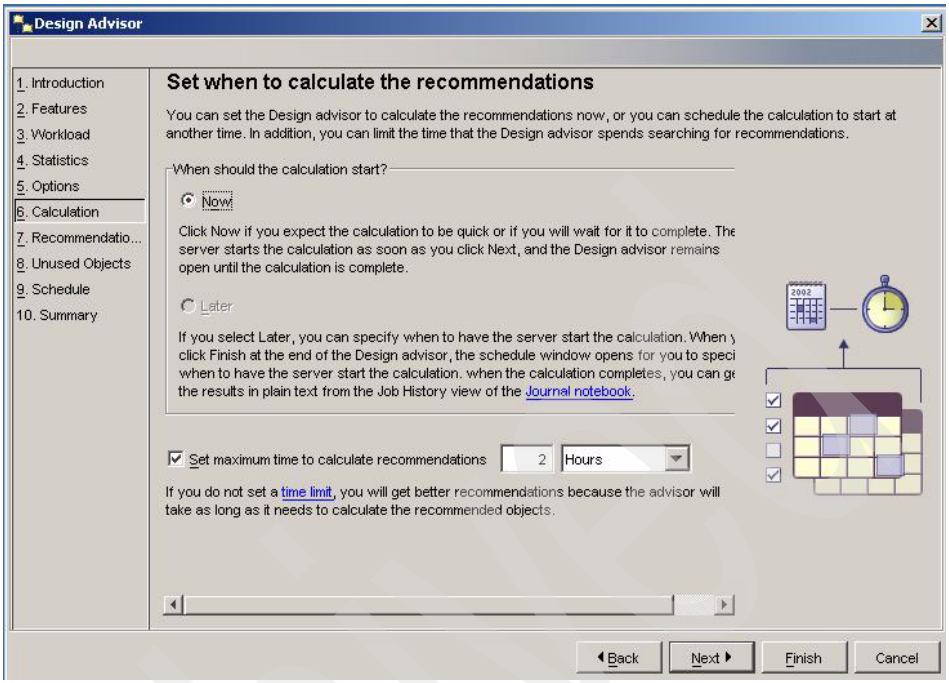


Figure 3-13 When to calculate the recommendations

After the calculation, the results are displayed. Notice that there might be no new objects to recommend. If one or more objects are recommended, the estimated performance improvement and disk space needed for each one is also shown, as illustrated in Figure 3-14 on page 74.

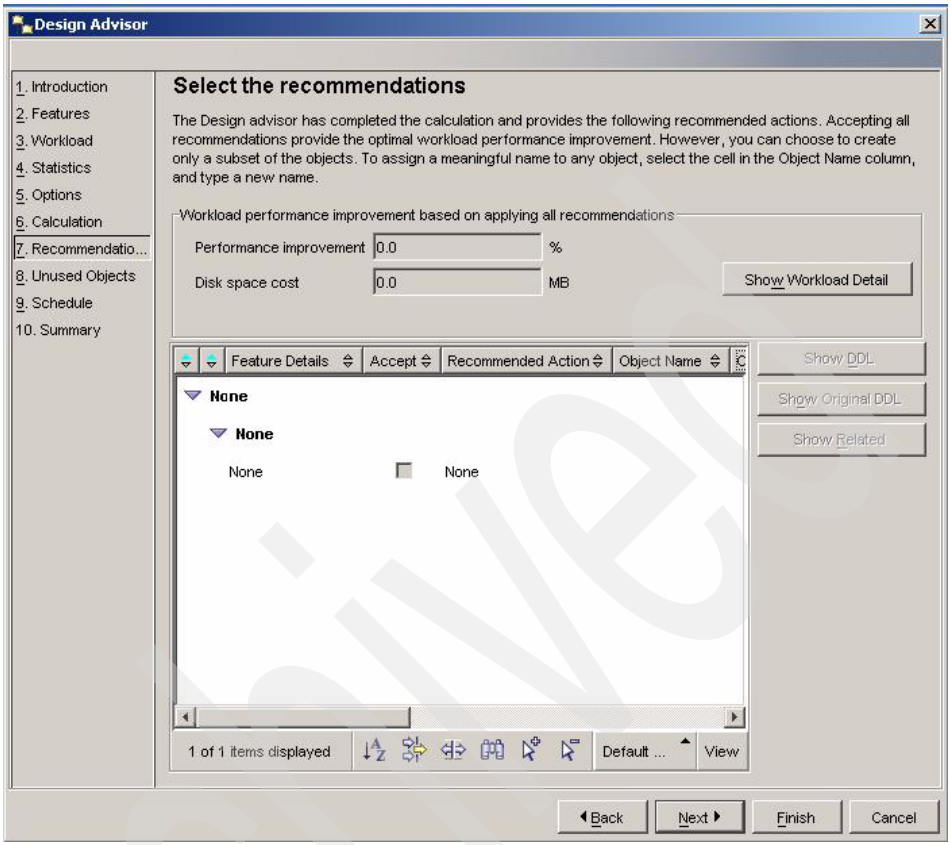


Figure 3-14 Select the recommendations

8. Another interesting feature is that the advisor not only recommends creating new objects, but also dropping unused database objects. Just select the objects that you want it to drop, as shown in Figure 3-15 on page 75.

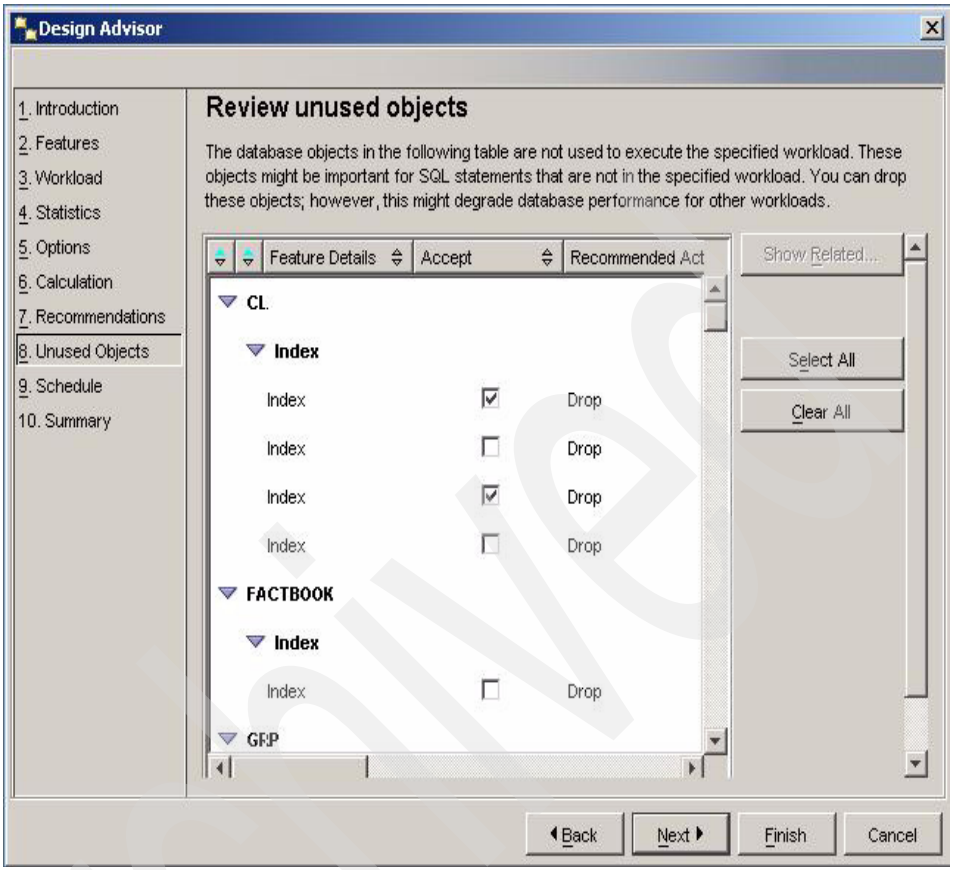


Figure 3-15 Review unused objects

9. You can also enable the scheduling functions or execute the recommendations immediately. Click **Generate** to create a report with all of the recommended actions, as shown in Figure 3-16 on page 76.

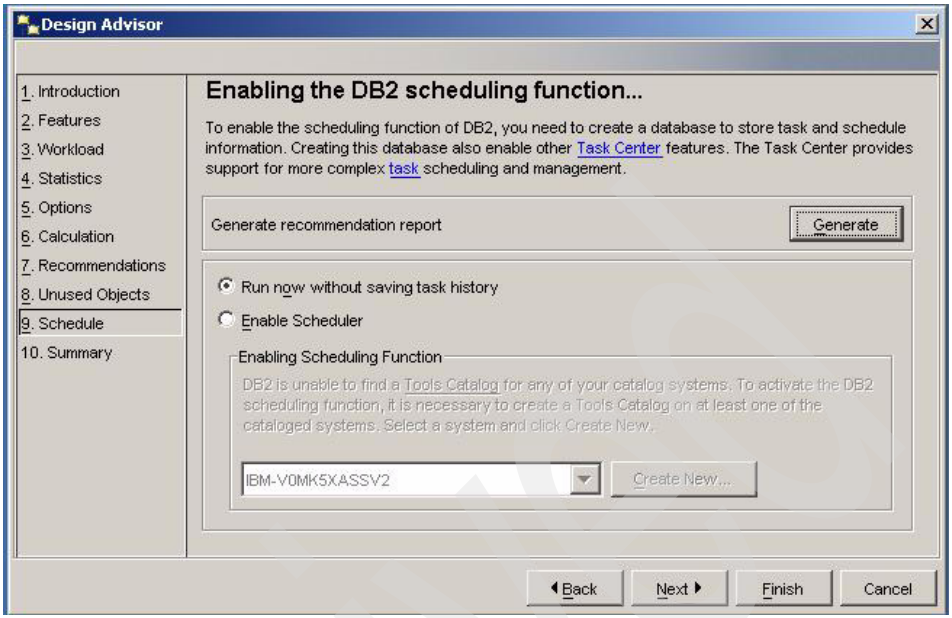


Figure 3-16 Enabling the DB2 scheduling function

10. You now have a generated report with a script. You either run this script from the advisor or run it manually to get the recommendations, as shown in Figure 3-17 on page 77.

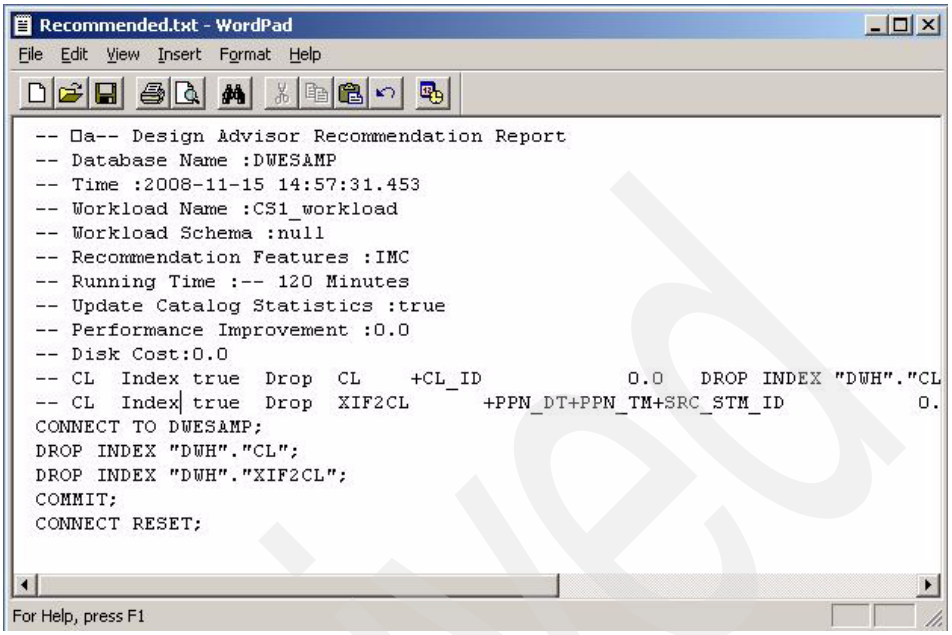


Figure 3-17 Recommended actions

11. The last step shows a summary of all recommendations, as shown in Figure 3-18 on page 78. Click **Finish** to run the script, provided that you selected the **Run Now** option in the Scheduling page.

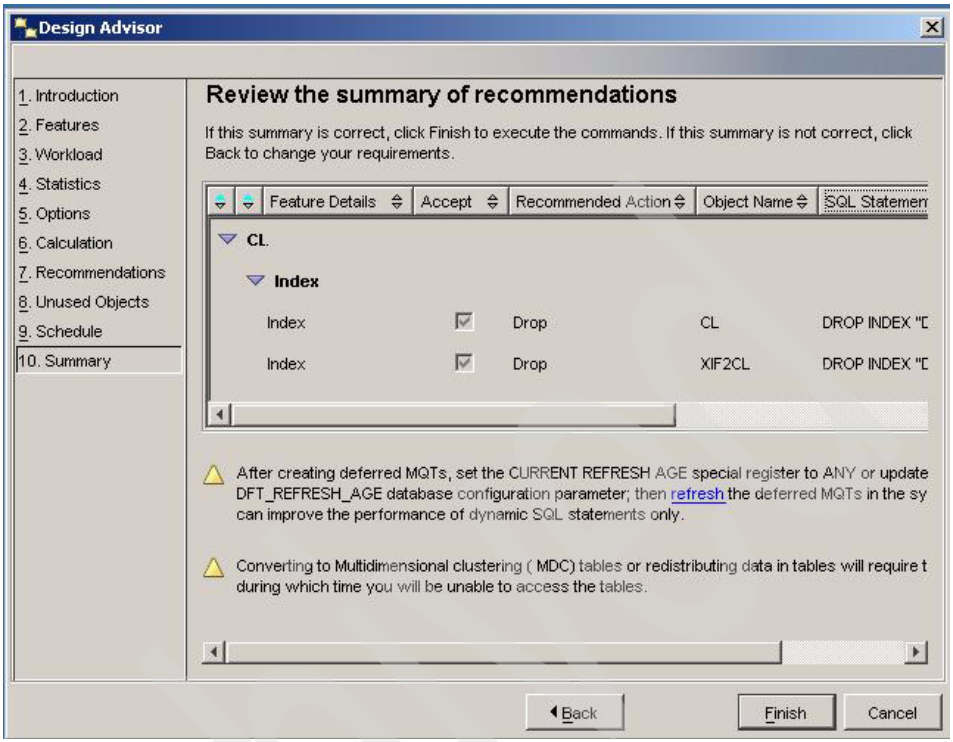


Figure 3-18 Summary of recommendations

3.2 Topologies: star, snowflake, and multi-star schemas

A multidimensional model is typically based on a large central fact table and a number of tables for the dimensions used to join with the fact table. Figure 3-19 shows examples of particular types of dimensional models.

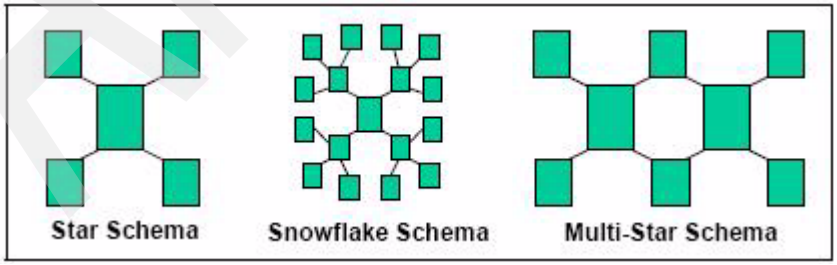


Figure 3-19 Types of dimensional models

When a query is sent to a cube and the data needed is not on cache, one or more SQL statements are sent to the dimensional model. The DB2 optimizer evaluates each of them to see if an existing MQT already has the data. If not, the SQL queries are sent to the dimensional model.

The star schema is commonly used because of its simplicity, which translates to speed. Just a few simple joins are needed to help the query performance when data volume is high. If proper indexes exist, the response time can be fast. This schema tends to be the fastest. The space savings that are offered by snowflaking is usually small when compared to the fact table size, and the SQL queries became more complex and time consuming.

A possible exception to that is when a large dimension table is used (for example, millions of rows) then snowflaking that dimension might have a good impact on performance.

More details about when to and when not to snowflake a dimension is in the publication *Dimensional Modeling: In a Business Intelligence Environment*, SG24-7138, in section 5.3, “Identifying the model dimensions.”

The Multi-Star schema is a group of star schemas that share one or more dimensions. Currently, Cubing Services does not support more than one fact table per model. If you must access more than one at a time, create a view that joins those fact tables together, and use the view as the fact table. However, in this case performance might become an issue.

Statistical views

In case a view joining fact tables are needed for the cube, use the DB2 statistical view feature to help the Optimizer with real statistics for the view’s result set. To do that, we must Enable the view for optimization.

You can enable a view for optimization using the new ENABLE OPTIMIZATION clause on the ALTER VIEW statement, for example, to enable the view myview for optimization, enter:

```
ALTER VIEW myview ENABLE QUERY OPTIMIZATION
```

A view that is enabled for optimization has a 'Y' in character position 13 of the PROPERTY column of its corresponding SYSTABLES entry. A view that is disabled for optimization has a blank in character position 13 of the PROPERTY column of its corresponding SYSTABLES entry.

Execute RUNSTATS, for example, to collect statistics on the view myview, enter:

```
RUNSTATS ON TABLE db2dba.myview
```

To use statistics sampling to collect view statistics, including distribution statistics, on 10 percent of the rows using row-level sampling, enter:

RUNSTATS ON TABLE db2dba.myview WITH DISTRIBUTION TABLESAMPLE BERNOULLI (10)

Executing RUNSTATS: Prior to DB2 Version 9.1, executing RUNSTATS on a statistical view only primed the catalog statistics tables for manual updates and did not collect any statistics. Starting from DB2 Version 9.1, executing RUNSTATS on a statistical view collects statistics, which means that RUNSTATS might take much longer to execute than previously, depending on how much data is returned by the view.

3.3 Model considerations

In this section, we present some model considerations about granularity, accessing more than one table as a fact table, and joining rules.

3.3.1 Granularity

The granularity of a fact table is the level of detail that it has. It represents the most detailed data that is available for the cubes. Determining the grain level has a huge impact in a fact table size. The grain level is determined during the business requirement phase and is certainly one of the most important project definitions.

You can choose to have a detailed data in the fact table, more than solicited by the business rules, because it is always possible to aggregate it into more summarized data, if needed in the future, but then its huge size might be a concern. Also, the backup size and time of execution are larger, and the same applies to restoring such a big table. Performance is impacted too. It might be a good idea to have a fact table with the level of data detail (or grain) equal to the one determined during the business requirements gathering phase.

If, in a retail store analysis, you just need the total amount of money spent for each customer, the fact table's number of rows might not increase. If you must track each item on every sales receipt, then the fact table's number of rows might increase 10 times or more.

An in depth study about how to choose the fact table's grain level is in the Redbooks publication *Dimensional Modeling: In a Business Intelligence Environment*, SG24-7138. See section 5.2, "Identifying the grain for the model."

3.3.2 Accessing multiple fact tables

During the process of implementing the physical models to meet the business requirement analysis, several fact tables might be created because of different data granularities or different subjects.

You might need to access more than one fact table, but Cubing Services does not allow a selection of more than one table for the fact table, which is for performance reasons.

One way to join fact tables and to use them in Cubing Services is to create a view with the desired join, and use it as the fact table. But because this involves big tables, the performance is certainly very low, and it might even be unacceptable. Because they probably do not have exactly the same set of columns, you must join them using FULL OUTER JOIN mode, which is not good in terms of performance.

If you just must use this approach, make sure the referential constraints are properly defined between all involved fact tables and their dimensions. Alter the view so that it becomes a statistical view, and collect detailed statistics for it. This procedure will help improve the performance for this non-recommended use of joining fact tables.

3.3.3 Join scenarios

The joins between the tables must follow some requirements:

- Constraints between a primary dimension table and the fact tables must be 1:Many (Star schema)
- Constrains between outrigger tables and a primary dimension table might be Many:1 or 1:1 (Snowflake schema).
- Constraints must be between non nullable columns
- Joins must be inner joins

See Figure 3-2 on page 58, which depicts a snowflake schema.

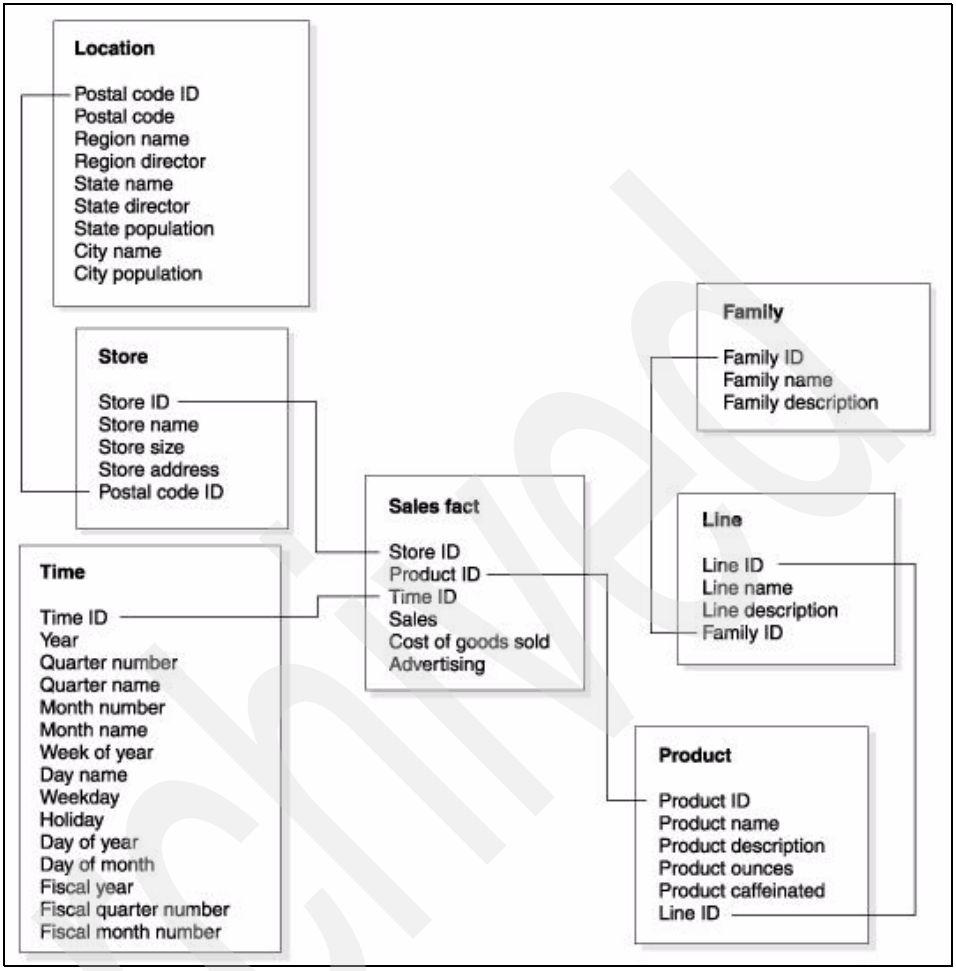


Figure 3-20 Snowflake schema

In a snowflake schema, each dimension has a primary dimension table to which one or more additional dimensions can join. The primary dimension table is the only table that can join to the fact table. Each of the outrigger tables that join directly to the primary table must have a join cardinality of Many:1 (where Many is on the side of the primary table) or 1:1. The primary dimension table usually has the most detailed level of information of all of the dimension tables because of these join cardinality rules. If a set of dimension tables only uses 1:1 join cardinalities, all of the tables have the same level of detail. Figure 3-21 on page 83 shows the valid dimension for optimization.

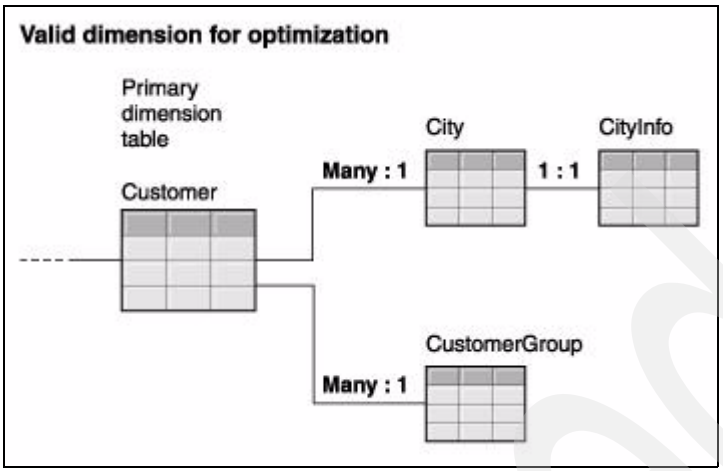


Figure 3-21 Valid dimension optimization

Figure 3-22 shows the invalid dimension for optimization.

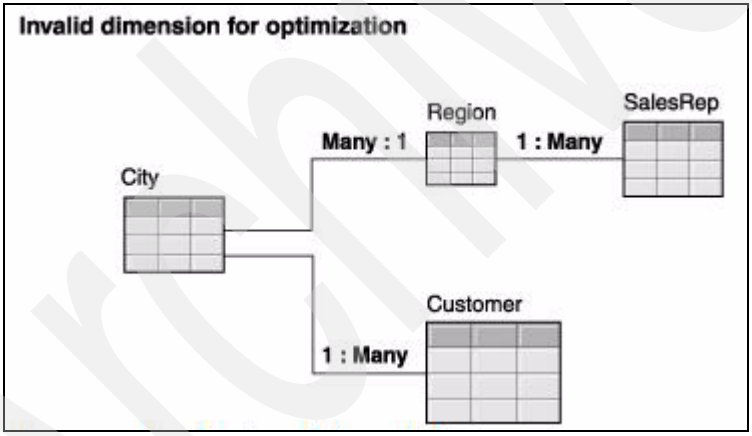


Figure 3-22 Invalid dimension for optimization

For more details about the Cubing Services Optimization Advisor, search for "Constraint definitions for optimization" in the DB2 Information Center. It is located at the following Web site:

`http://publib.boulder.ibm.com/infocenter/db2luw/v9r5/index.jsp`

A number of other best practices for a good dimensional model are described in the Redbooks publication, *InfoSphere Warehouse: Cubing Services and Client Access Interfaces*, SG24-7582. See section 12.1, "Best Practices for Star Schema."

3.4 Deciding to use Relational DB, OLAP, or tooling

In an OLAP environment, the most common way to investigate data is from top to bottom, that is, starting with the most aggregated numbers and then drilling to more detailed data, for example, going from the entire country to the state level and then drilling further down to city level. So, you must aggregate a large part of the fact table to get the numbers for the country, which can take a significant amount of time to calculate. The Cubing Services Optimization Advisor can help here, significantly, by generating the DDL scripts to create one or more summary tables and calculating the aggregated data in advance. A query for aggregated values is redirected to those MQTs that hold the values that are already calculated, which dramatically speeds up the process.

But what if you need something other than plain aggregation, such as calculating a metric derived from two or more other values? Should the calculation be realized at the database level (RDBMS), at the Cubing Services level, or even at the front-end tool?

The relational database is good at performing calculations when all of the values are from the same granularity. But it is not as good at calculating values from different granularities. Fortunately, this is an area were the OLAP engine does well. With this in mind, try to send the calculations to the appropriate place.

3.4.1 RDBMS

Leave calculations, such as a percentage between two column values (which are at the same grain level), to the RDBMS. To create a new measure using SQL statements:

1. Right-click the **Measures** members in Cube Model, as shown in Figure 3-23 on page 85, and write down the SQL expression.

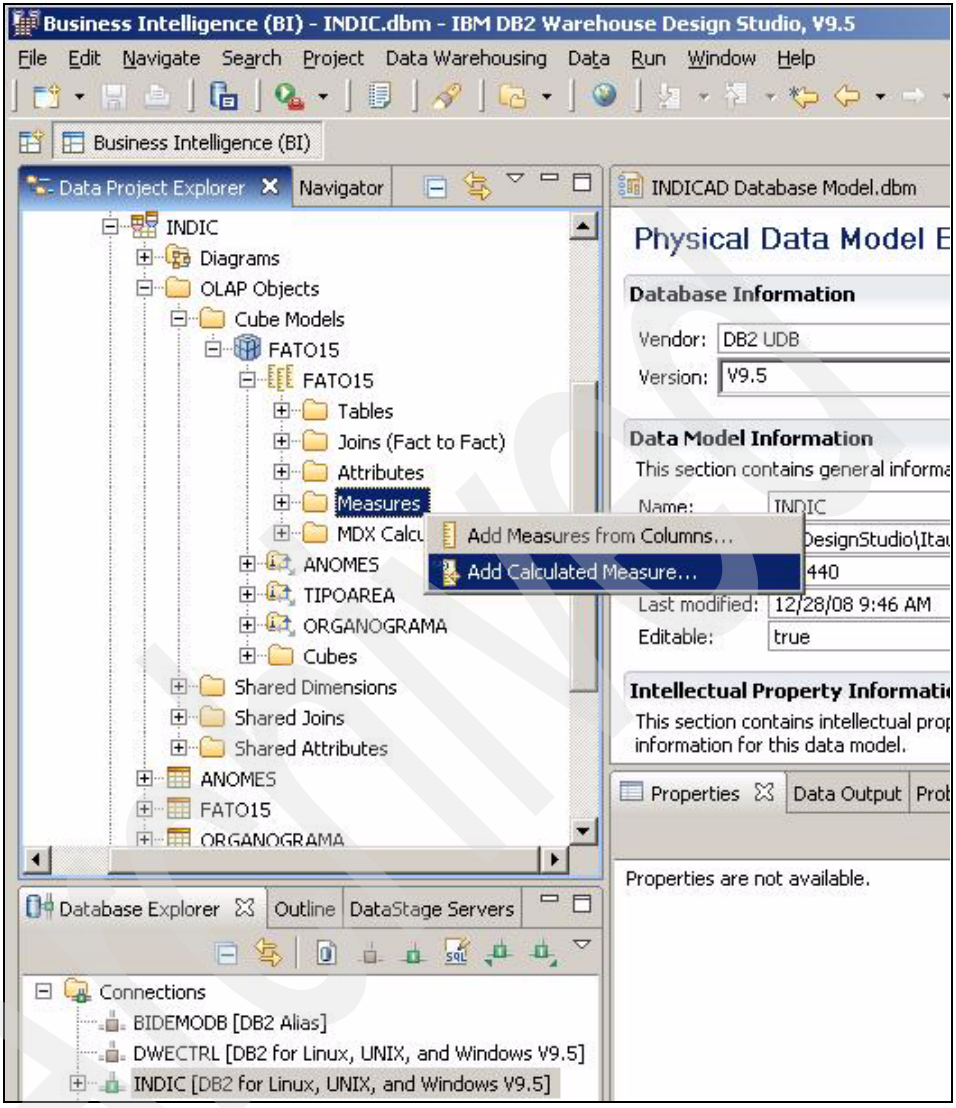


Figure 3-23 Creating a new measure using SQL statements

2. Use the OLAP SQL Expression® Builder to develop the SQL statement, as shown in Figure 3-24 on page 86.

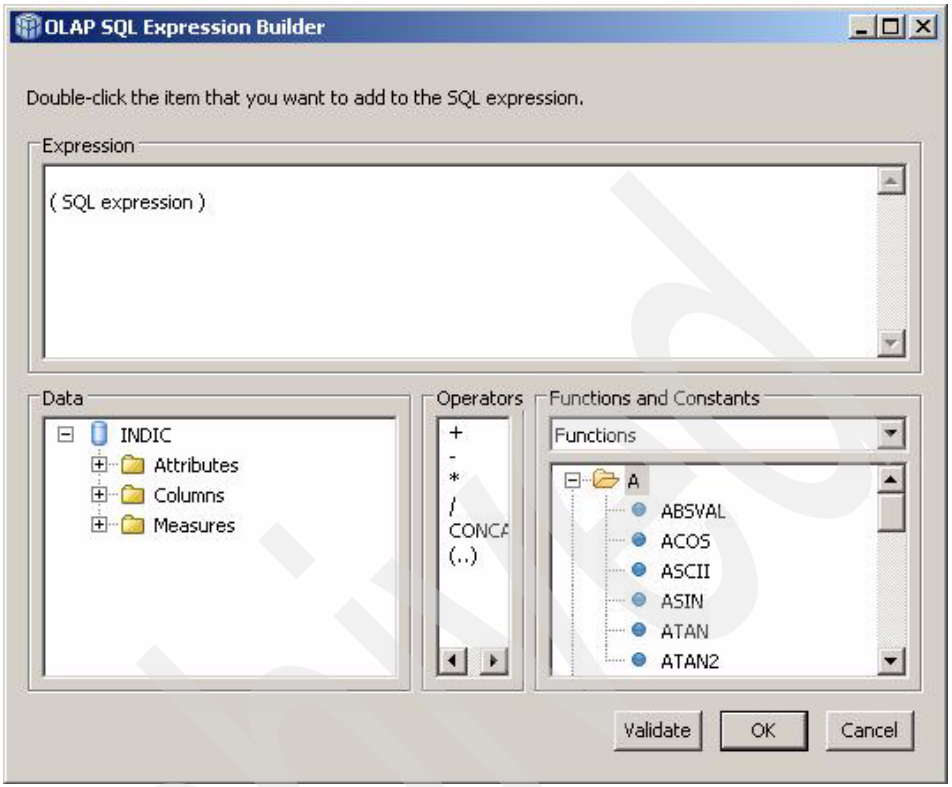


Figure 3-24 SQL Expression builder window

3.4.2 OLAP

On the other hand, when a calculation is needed using different grain levels, such as to show how each State compares to the Country, an OLAP engine (Cubing Services) is more appropriate for the task. Cubing Services makes the determination and sends the necessary SQL queries to the RDBMS, one or more for each involved level. Then, with the result set of those queries in cache, the OLAP engine performs the calculations to obtain the correct answer, as shown in Figure 3-25 on page 87.

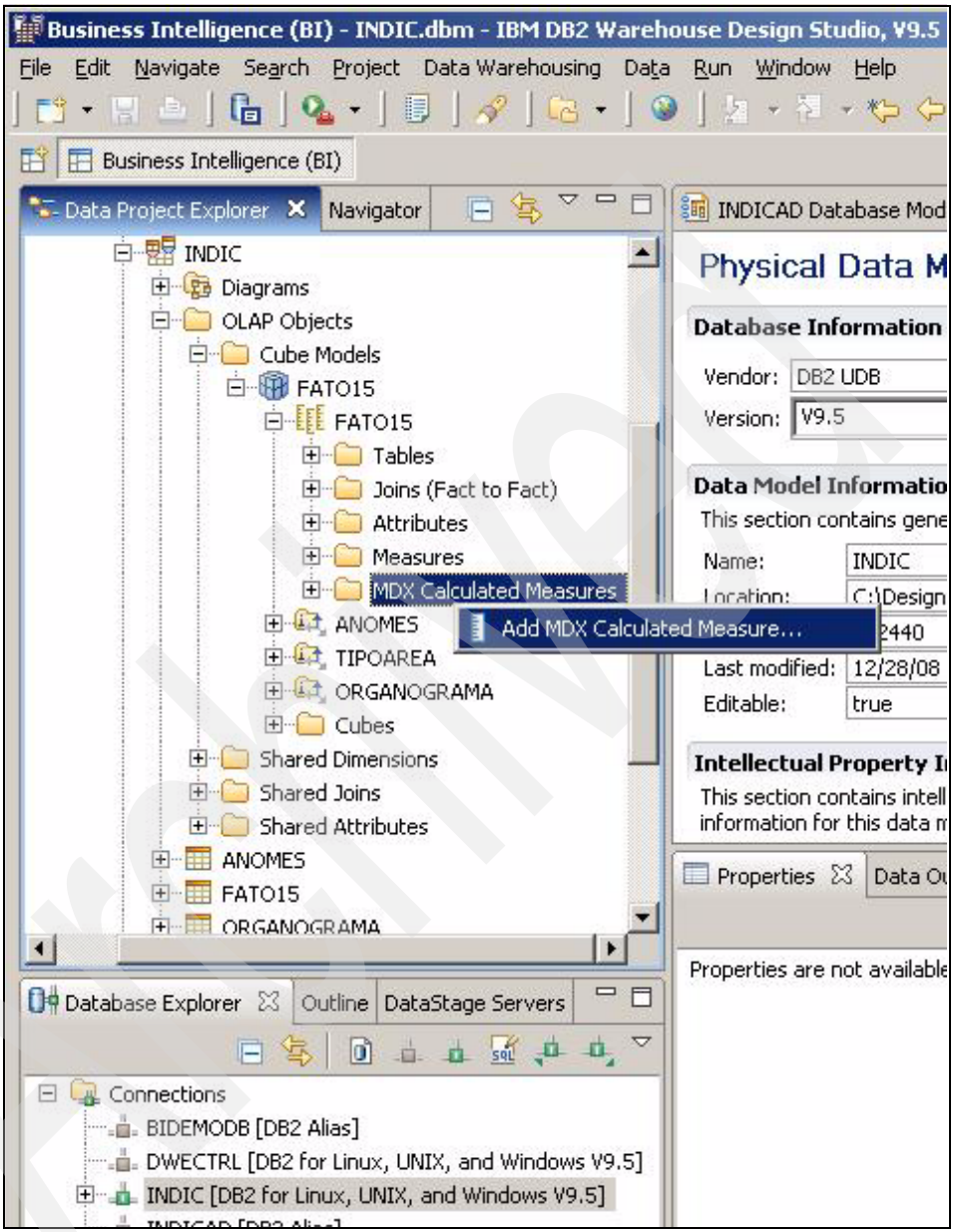


Figure 3-25 Creating a new measure using MDX statements

Use the MDX Expression Builder window to create the MDX expression, as shown in Figure 3-26 on page 88.

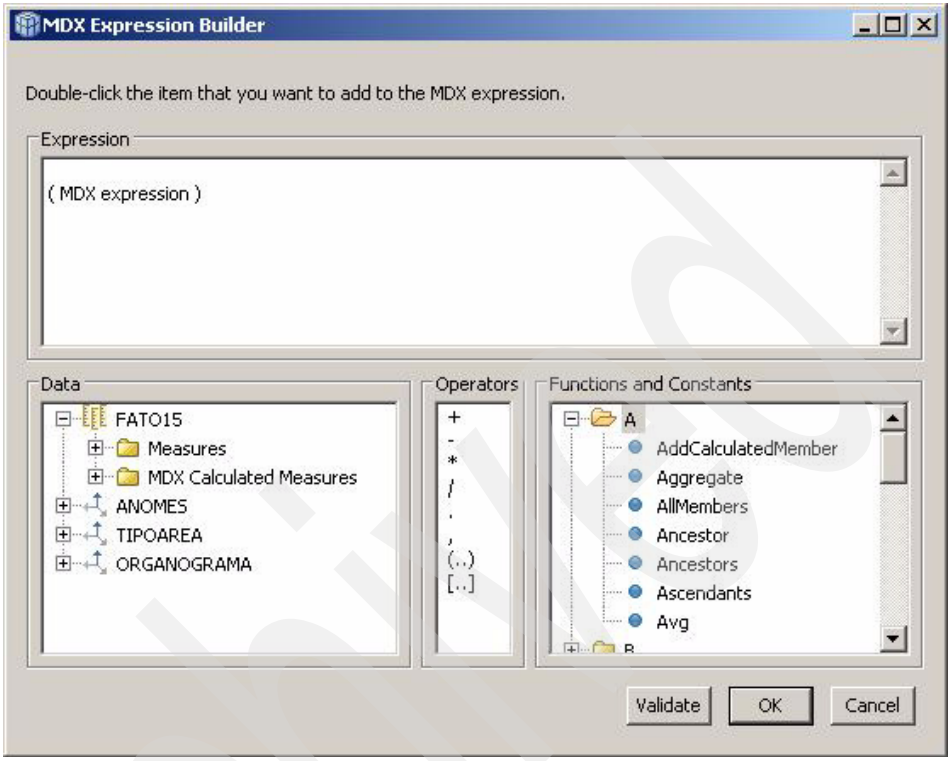


Figure 3-26 MDX Expression builder window

3.4.3 Tooling

The front-end tool might be used to create measures that a user might need, but it is not available in the OLAP cube, most likely by power users. Those users can create new calculations using any available measure, even other calculated measures. If this new calculation proves to be useful for other users, you can implement it in the cube model to be available to all users.

4

Cubing services model implementation

InfoSphere Warehouse Cubing Services cubes are developed using the InfoSphere Warehouse Design Studio. A Cubing Services cube is always part of a cube model, and a cube model consists of facts, which are also called *measures* and *dimensions*. Dimensions themselves consist of *hierarchies*, *levels*, and *attributes*. A single cube model can contain several cubes that get defined with a combination of the defined facts, dimensions, hierarchies, and levels.

In this chapter, we describe how to create dimension, hierarchy, and fact types in InfoSphere Warehouse Cubing Services for various dimensional design patterns.

4.1 Dimensions

A dimension is always part of a single cube model. To create a dimension the developer specifies the single dimension table, in the case of a star schema, or several dimension tables and the join definition between them, in the case of a snowflake schema. In addition, the developer must specify the join between the dimension and the fact table, attributes, levels, and hierarchies.

If the dimension gets created as part of a new cube model, the join definitions are created and the attributes are added automatically if foreign key relationships are defined, as shown in Figure 4-1; otherwise, the first step is to define the dimension tables, the joins, and add the attributes that are necessary for the levels.

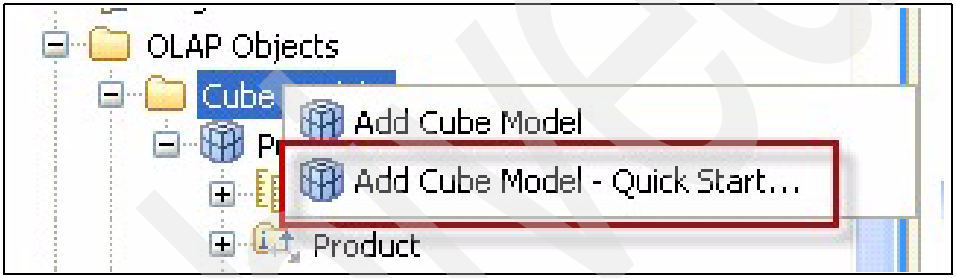


Figure 4-1 Creating a cube with Quick Start

4.1.1 Star and snowflake schema-based dimensions

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

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

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

The dimensional schema can be physical or logical. In a logical dimensional schema, the facts, measures, and dimensions can be represented as views over the underlying database tables. A physical relational schema is typically represented in the form of a star or snowflake schema, where the objects in the star or snowflake schema are actually database tables. Although not a common scenario, the dimensional schema can even take the form of a single table or view, where all of the facts and dimensions are simply in different columns of that table or view. Additionally, the dimensional schema can be composed of some combination of both logical and physical elements.

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

InfoSphere Cubing Services does not require a physical star or snowflake schema. It can be modeled logical, but a physical schema is recommended for performance reasons. Cubing Services can consume logical star schema and snowflake schema.

Cubing Services treats both dimension types the same. The difference is that for a snowflake dimension, you must specify the Joins (Dimension to Dimension).

4.1.2 Time dimension type

A time dimension is treated differently in InfoSphere Cubing Services because it allows time calculations, such as year-to-date (YTD), month-to-date (MTD), and so forth, using MDX calculations. For those MDX calculations to work, the type of the dimension must be Time versus Regular. All other dimensions must be of the type Regular.

To define the time dimension:

1. First create the dimension.
2. Click the created dimension, and click the Type tab from the property sheet.
3. Choose Time as the type for the dimension, as depicted in Figure 4-2 on page 92.

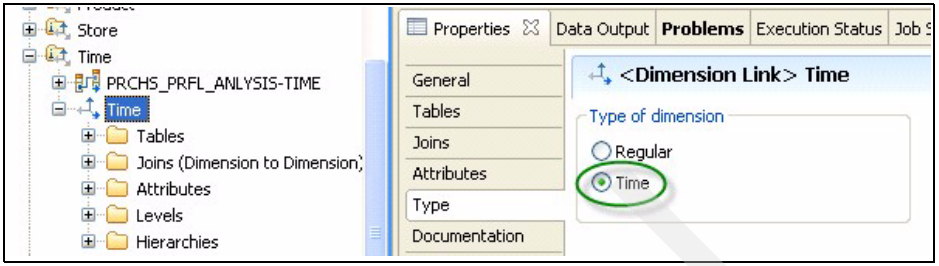


Figure 4-2 Time dimension property: set in Design Studio

Additionally each level in the time dimension must be of the type Time or Date and must be marked correctly so that the time series MDX functions work properly, for example, the quarter level must be of the type Time and Date and also must be set to Quarters, as shown in Figure 4-3.

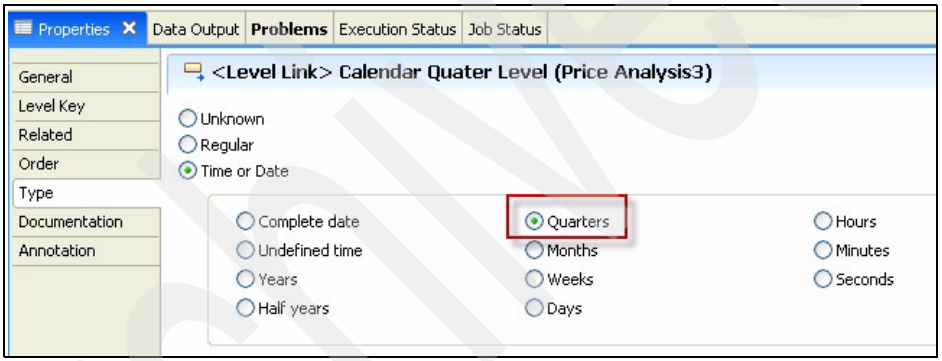


Figure 4-3 Mark level as quarters

Now, you can define an MDX Calculated Member for the QTD calculation, as shown in Figure 4-4, which is the QTD MDX Statement, for example:

[Time].[QTD] = SUM(QTD([Time].[All Time (Calendar)].[2004].[Q 3].[July]))

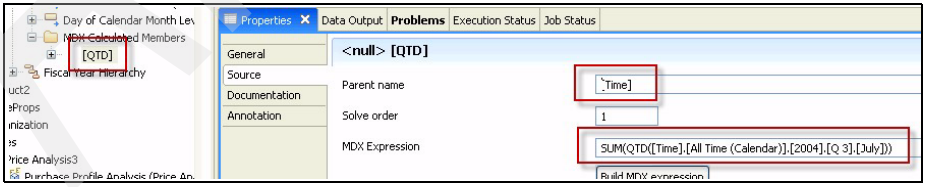


Figure 4-4 QTD MDS statement

Hierarchies

Next, you define levels by defining a level key, related attributes, default attribute, the order attributes, and in the case of a time dimension the level type. After that you must create a hierarchy to which the levels are added and put into the correct roll-up order. Additionally, you must also specify the hierarchy type.

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

In the Cubing Services Cube Server, the supported hierarchy types are:

- Standard:
  - Balanced
  - Ragged
  - Unbalanced
- Recursive:
  - Unbalanced

Standard hierarchies

Standard hierarchies use the relationship of the level definitions of the hierarchy, where each level in the hierarchy is used as one item in the deployment, for example, a balanced hierarchy for a Time dimension is organized by each defined level, which includes Year, Quarter, and Month. Standard deployment can be used with all four hierarchy types. Table 4-1 is an example of a standard hierarchy and shows how some of the balanced hierarchy attributes for a Time dimension are organized using a standard deployment.

Table 4-1 Standard hierarchy

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

4.1.3 Balanced standard hierarchies

A *balanced hierarchy* is a hierarchy with meaningful levels and branches that have a consistent depth. Each level's logical parent is in the level directly above it. A balanced hierarchy can represent time where the meaning and depth of each level, such as Year, Quarter, and Month, is consistent. They are consistent because each level represents the same type of information, and each level is logically equivalent. Figure 4-5 illustrates an example of a balanced time hierarchy.

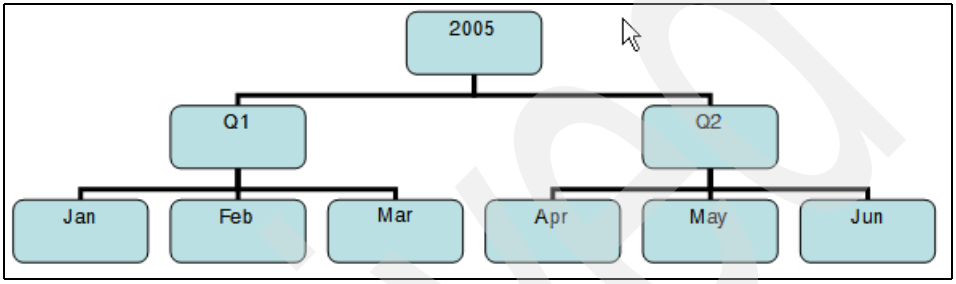


Figure 4-5 Balanced hierarchy

By default, a new hierarchy is of the type Balanced / Standard. Standard means that the hierarchy is not recursive. To set the property:

1. Select the hierarchy.
2. In the property sheet, select Type/Deployment, and select **Balanced / Standard**, as shown in Figure 4-6.

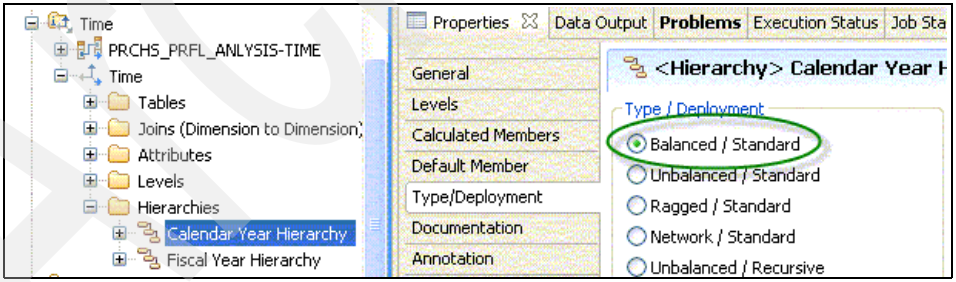


Figure 4-6 Balanced hierarchy: property set in Design Studio

4.1.4 Unbalanced standard hierarchies

An *unbalanced standard hierarchy* occurs when each level has a consistent meaning, but the branches have inconsistent depths because at least one member attribute in the leaf levels is unpopulated, for example, Figure 4-7 on

page 95 is an example of an unbalanced hierarchy that shows the hierarchy of entertainment products. It demonstrates that TV shows have an additional level for episodes compared to Movies, which causes the hierarchy to be deeper for TV shows. Unbalanced standard hierarchy structures are typical for product hierarchies with a diverse set of products.

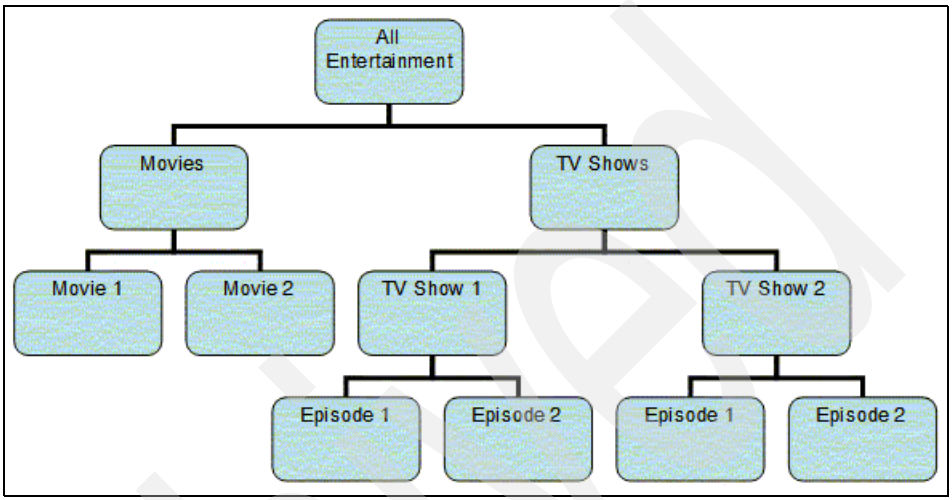


Figure 4-7 Unbalanced standard hierarchy

To create an unbalanced standard hierarchy from a star schema relational structure in InfoSphere Cubing Services:

1. Set the missing levels to NULL for the default attribute of the missing levels.

   Missing levels must have NULL values for the default attribute of the missing levels. In an unbalanced hierarchy, sample relational table, the ID column is used to join back to the fact table, and there are three levels: Category, Title, and Episode. Because movies do not have an episode level, all values for the episode column must be NULL to produce a hierarchy, as shown in Table 4-2.

Table 4-2 Unbalanced hierarchy: sample relational table

ID	Category	Title	Episode
1	Movies	Movie 1	NULL
2	Movies	Movie 2	NULL
3	TV Shows	TV Show 1	Episode 1
4	TV Shows	TV Show 1	Episode 2
5	TV Shows	TV Show 2	Episode 1

ID	Category	Title	Episode
6	TV Shows	TV Show 2	Episode 2

2. Set the property in InfoSphere Design Studio to mark the hierarchy as **Unbalanced**.
3. Select the hierarchy. In the property sheet, under Type/Deployment, select **Unbalanced / Standard**, as shown in Figure 4-8. Cubing Services creates an unbalanced hierarchy only if it finds cell values to be NULL in the underlying dimension table that corresponds to the default related attribute of the leaf levels.

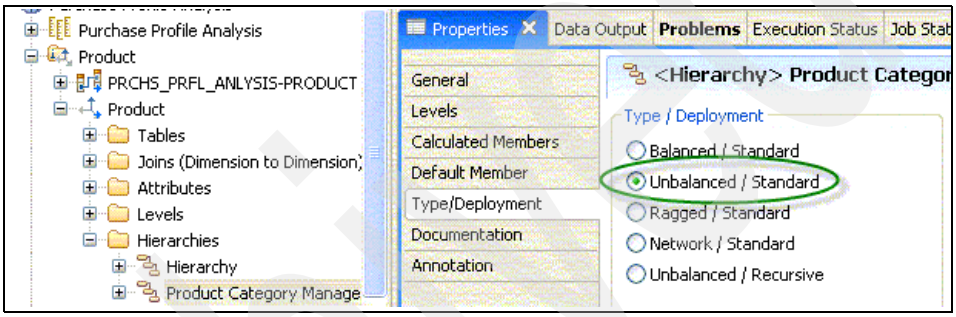


Figure 4-8 Unbalanced hierarchy: set property in Design Studio

4.1.5 Ragged standard hierarchies

A hierarchy in which each level has a consistent meaning but the branches have inconsistent depths because at least one member attribute in a branch level is unpopulated is called a *ragged standard hierarchy*. A ragged hierarchy can represent a geographic hierarchy where the meaning of each level, such as city or country, is used consistently, but the depth of the hierarchy varies, as shown in Figure 4-9 on page 97. The example of a ragged hierarchy in Figure 4-9 on page 97 shows a geographic hierarchy that has Continent, Country, Province/State, and City levels defined. One branch has North America as the Continent, United States as the Country, California as the Province or State, and San Francisco as the City. However, the hierarchy becomes ragged when one member does not have an entry at all of the levels, for example, another branch has Europe as the Continent, Greece as the Country, and Athens as the City, but has no entry for the Province or State level because this level is not applicable to Greece for the business model in this example. In Figure 4-9 on page 97, the Greece and United States branches descend to different depths, which creates a ragged hierarchy.

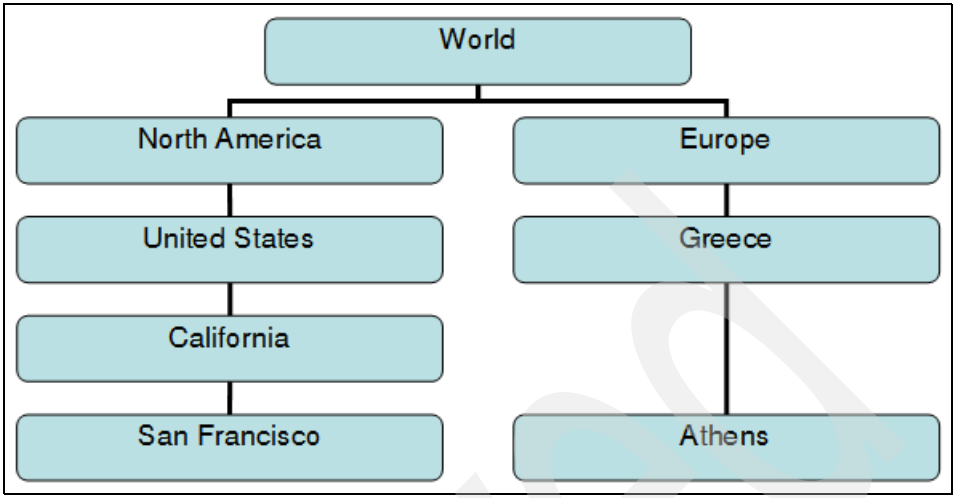


Figure 4-9 Ragged hierarchy

Before you can model the dimension, it is important to understand the format of the relational table that allows Cubing Services to create a ragged hierarchy, such as the one in Figure 4-9. To create an example of a ragged hierarchy, you must have a table that has a column for each level. Table 4-3 shows an example relational table. For the ragged hierarchy to display correctly in the cube it is necessary that missing levels, such as the State column for the record Athens, be NULL.

Table 4-3 Ragged hierarchy: sample relational table

ID	Region	Country	State	City
1	North America	United States	California	San Francisco
2	Europe	Greece	NULL	Athens

To create the ragged hierarchy:

1. Create a new hierarchy in InfoSphere Design Studio, such as the one shown in Figure 4-10 on page 98.
2. In the Properties sheet, click the Type/Deployment tab, and select **Ragged / Standard** as the type. Cubing Services creates a ragged hierarchy only if it finds cell values to be NULL in the underlying dimension table that corresponds to the default related attribute of the missing levels.

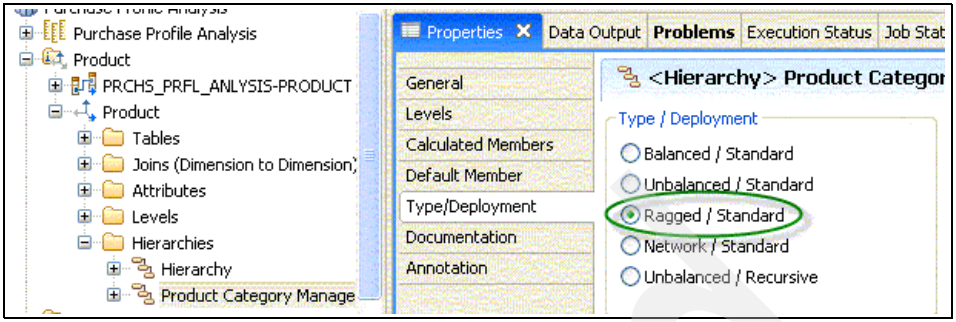


Figure 4-10 Ragged hierarchy - property set in Design Studio

4.1.6 Unbalanced recursive hierarchies

An unbalanced hierarchy using a recursive deployment is represented as parent-child level pairs. As an example, an organizational structure is typically expressed as a parent-child relationship through the relationship of an employee (child) to the manager (parent). Figure 4-11 on page 99 illustrates the resulting hierarchy of recursive deployment. It is the recursive deployment of an unbalanced hierarchy for an Organization dimension, describing an organization chart, which is created from a relational table as shown in Table 4-4 on page 99. Recursive deployment can be used only with an unbalanced hierarchy when the hierarchy consists of exactly two attributes with an inherent parent-child relationship.

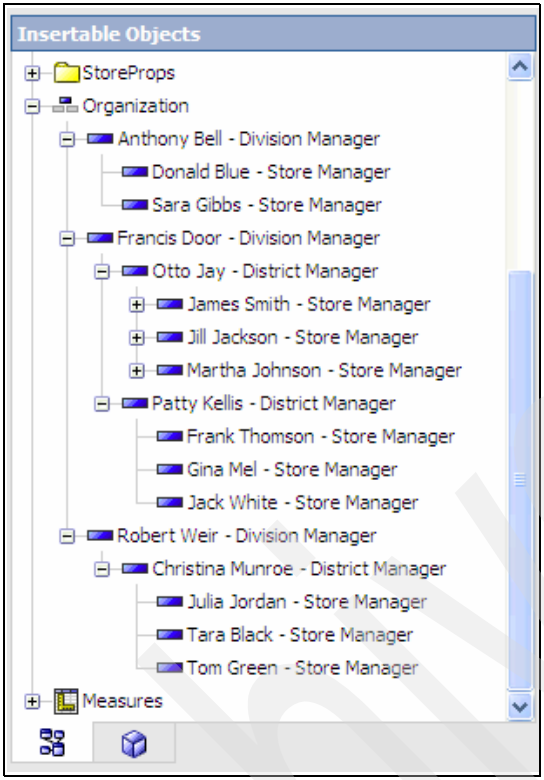


Figure 4-11 Recursive hierarchy

Before you create the *unbalanced recursive hierarchy*, you must correctly set up the relational table, similar to Table 4-4. Have a single table or view that presents the parent-child hierarchy. If several tables are required to build the dimension, you can create a view across the tables to flatten the structure. Additionally, it is also necessary for the root member to have a parent key with a NULL value.

Table 4-4 Recursive Relational Dimensional Table

EMP_ID	FIRST_NM	LAST_NM	TITLE	MGR_ID
1	James	Smith	Store Manager	12
2	Martha	Johnson	Store Manager	12
3	Jill	Jackson	Store Manager	12
4	Gina	Mel	Store Manager	13
5	Jack	White	Store Manager	13

EMP_ID	FIRST_NM	LAST_NM	TITLE	MGR_ID
6	Frank	Thomson	Store Manager	13
7	Tara	Black	Store Manager	14
8	Tom	Green	Store Manager	14
9	Julia	Jordan	Store Manager	14
10	Donald	Blue	Store Manager	15
11	Sara	Gibbs	Store Manager	15
12	Otto	Jay	District Manager	16
13	Patty	Kellis	District Manager	16
14	Christina	Munroe	District Manager	17
15	Anthony	Bell	Division Manager	19
16	Francis	Door	Division Manager	19
17	Robert	Weir	Division Manager	19
19	Paula	Brown	VP	NULL

To create a recursive hierarchy in InfoSphere Cubing Services, the hierarchy Type/Deployment property must be set to Unbalanced / Recursive. Select the hierarchy, and click the Type/Deployment tab in the Properties sheet, as shown in Figure 4-12.

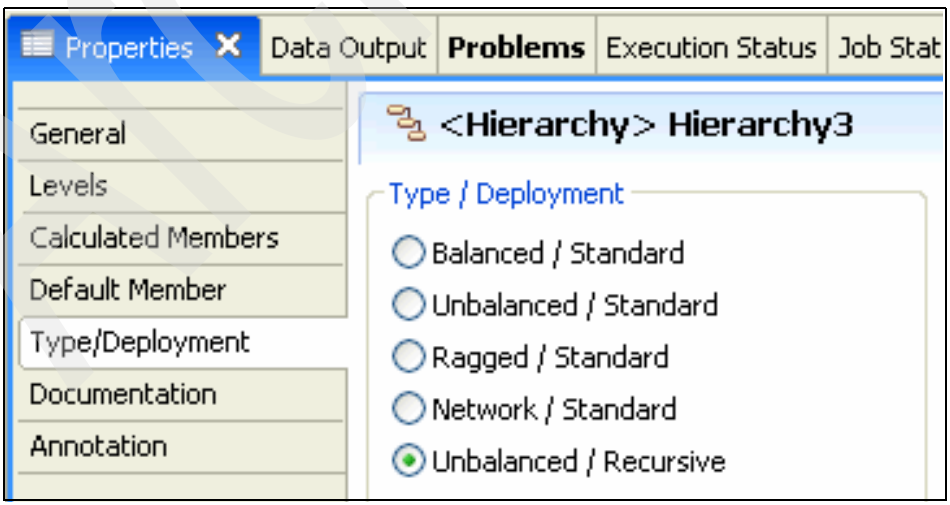


Figure 4-12 Unbalanced Recursive Type/Deployment property

An unbalanced recursive hierarchy must have exactly two levels: The top level uses the parent key as the level key and the bottom level the child key, for example, the organizational hierarchy has a manager and an employee level, as shown in Figure 4-13. For a table, as shown in Table 4-4 on page 99, the manager level uses the MGR_ID as the level key and a calculated attribute that concatenates the FIRST_NM, LAST_NM, and TITLE as the default related attribute, and the employee level uses the EMP_ID as the level key and the same default-related attribute as for the manager level.

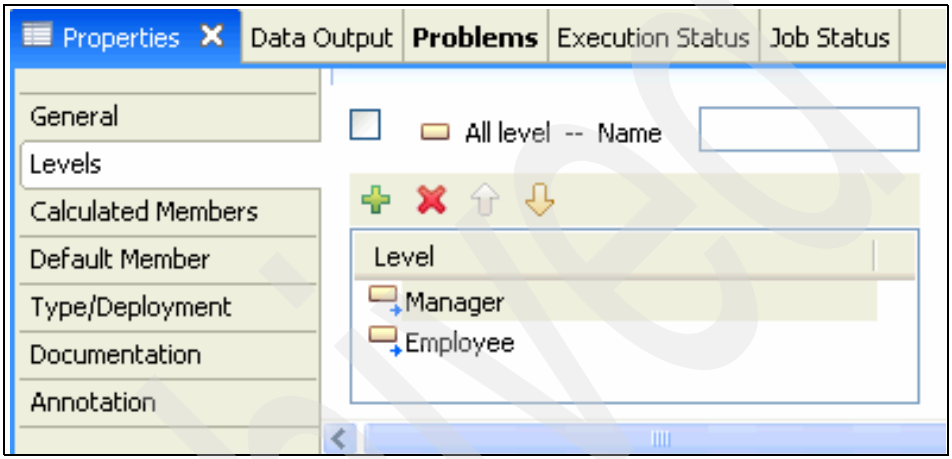


Figure 4-13 Unbalanced Recursive hierarchy: 2 levels

4.1.7 Facts

In this section, we discuss several types of facts.

Calculated facts

Calculated facts or measures are calculations that are based on existing measures or column values in the relational database, for example, consider a sales table that has the columns UNITS and AMOUNT. You can create a calculated measure called SALES = AMOUNT X UNITS. However, InfoSphere Cubing Services supports two types of calculations:

- SQL calculations
- MDX calculations

SQL calculations are expressed using an SQL function in the InfoSphere Design Studio. Cubing Services pushes those calculations to the data warehouse using SQL. The level where the calculation is executed depends on the aggregation type. In most cases SQL calculations are more efficient because the data warehouse provides the majority of the processing power, for example, to

calculate the Average Item Price Sold using an SQL calculation you can use the following calculation with the aggregation type = None.

```
Avg Item Price Sold = @Measure(MARTS.Sales Amount)/
@Measure(MARTS.Number Of Items)
```

MDX calculations are expressed using a MDX calculation in InfoSphere Design Studio, but the calculation is executed inside the Cubing Services Java™ Process. The MDX calculations specify at what level it is calculated. MDX calculations are a better choice when the SQL calculation gets complex, for example, the percentage of total and any time series calculations or multi-pass calculations.

To calculate the same Average Item Price Sold, that was previously shown using using SQL, with MDX, you can use the following calculation:

```
[Measures].[Avg Item Price Sold] = [Measures].[Sales Amount] /
[Measures].[Number Of Items]
```

In some cases, it is necessary to combine SQL and MDX calculations. An example is a dimensional calculation that calculates the number of patients, but excludes the patients in the current intersection. First, you must create an SQL function using a distinct count on PATIENT_ID, and then build a variance using an MDX function. By using the CurrentMember function, the calculation automatically adjusts for any patient set:

```
Patient Count = COUNT(distinct @col(DB2ADMIN.FACT.PATIENT_ID))
```

```
MEMBER [Patient].[Others] AS ([Patient].[All], [Measures].[Patient Count]) -
([Patient].CurrentMember, [Measures].[Patient Count]), SOLVE_ORDER=0
```

Using MDX calculation you can also create multi-pass calculations by nesting the MDX calculations and specifying a solve order. In this example, you want to calculate the variance for a location to the top 25% in terms of sales. Calculate the threshold for the stores that are in the top 25%, create a variance, and calculate the variance percentage.

```
MEMBER [Measures].[Min Top Qtl] AS MIN(TopPercent( {
[Locations].[Location].Members },25,[Measures].[Sales]  ),[Measures].[Sales]),
SOLVE_ORDER=1
```

```
MEMBER [Measures].[Min Top Qtl Var] AS ([Measures].[Sales] -
[Measures].[Min Top Qtl]), SOLVE_ORDER=2
```

```
MEMBER [Measures].[Var to Top Qtl] AS ([Measures].[Min Top Qtl
Var]/[Measures].[Sales]), SOLVE_ORDER=3
```

Cubing Services provides three ways to aggregate measures:

- Measures can be aggregated using a consistent aggregation function for Additive facts: Sum, Average, Count, Variance, Min, Max, and Standard Deviation.
- It supports Non-Additive facts where the calculation is calculated at each level and not rolled up.
- It supports Semi-Additive facts where the aggregation differs for levels using a custom aggregation script.

Additive facts

In *additive facts* the measure is rolled up across all dimensions and levels using a *consistent aggregation function*, the easiest kind of aggregation. An example of an additive fact is sales or cost of goods sold. To set the aggregation function:

1. Define the measure in InfoSphere Design Studio.
2. In the Properties pane, under the Aggregations pane, select the aggregation function, for example SUM, as shown in Figure 4-14, and set the aggregation type for an additive fact.

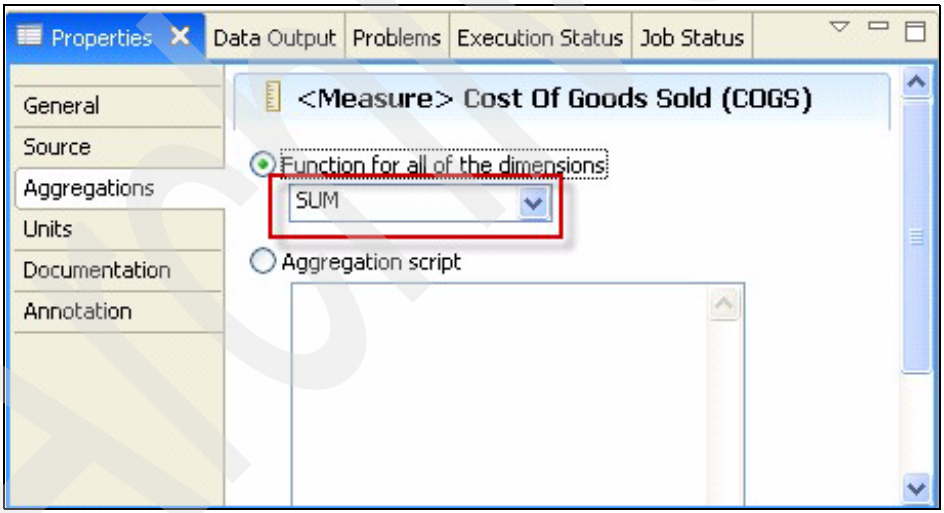


Figure 4-14 Set the aggregation type for additive fact

In this case, the calculation is executed on the bottom level and then rolled up using the summarization aggregation function. Cubing Services also supports average, count, standard deviation, max, min, and variance as aggregation types.

Non-additive facts

Non-additive facts cannot be rolled up through any of the dimensions in the cube model, which is typically the case for any measure calculations that result in ratios, averages, or variance percentages for example. Cubing Services supports non-additive facts for calculated measures by setting the aggregation function to None, which results in that calculation being calculated after the values are aggregated, as shown in Figure 4-15, for example for the calculation: Avg Item Price Sold = @Measure(MARTS.Sales Amount)/ @Measure(MARTS.Number Of Items). Sales Amount and Number of Items are first aggregated and then the ratio is calculated resulting in: SUM(@Measure(MARTS.Sales Amount)) / SUM(@Measure(MARTS.Number Of Items)).

To create a non-additive measure:

1. Define the calculation.
2. In the Properties pane, select the Aggregations tab, and under Function, for all of the dimensions, select **None**.

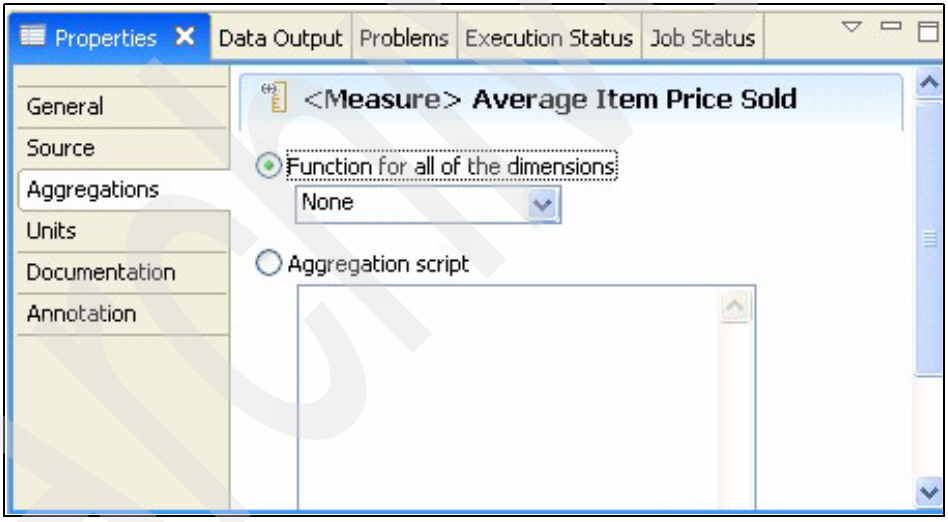


Figure 4-15 Defining a non-additive fact

Semi-additive facts

Semi-additive facts are rolled up through some of the dimensions of the cube model. Measures that are balances, for example, cannot be aggregated the same way across time as across other dimensions. If you want to calculate the Average Daily Balance for a relational table that looks like Table 4-5 on page 105, you must aggregate the balance for Customer and Account but perform an average for Time.

Table 4-5 Balance Table

Date	Customer	Account	Amount	Balance
1/1/2009	1001	30001	10	110
1/1/2009	1001	30002	2	42
1/2/2009	1001	30001	15	125
1/2/2009	1001	30002	-3	39
1/2/2009	1001	30003	43	103
1/3/2009	1001	30001	-5	120
1/3/2009	1001	30003	3	106
1/4/2009	1001	30001	20	140
1/4/2009	1001	30002	12	51

Cubing Services supports aggregation scripts that allow you to specify the aggregation type for each dimension for a measure. After you define the measure, go to the Properties pane, click the Aggregations tab, and select **Aggregation script**. Click **Edit Script**, as shown in Figure 4-16.

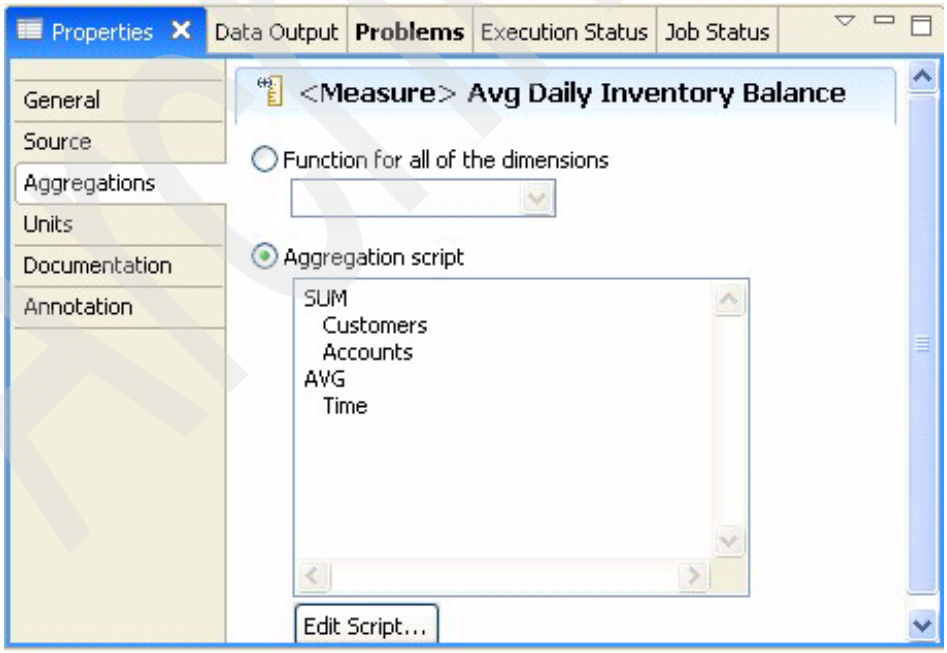


Figure 4-16 Using an aggregation script for semi-additive facts

Cubing Services allows you to specify a separate aggregation function for each dimension, as shown in Figure 4-17.

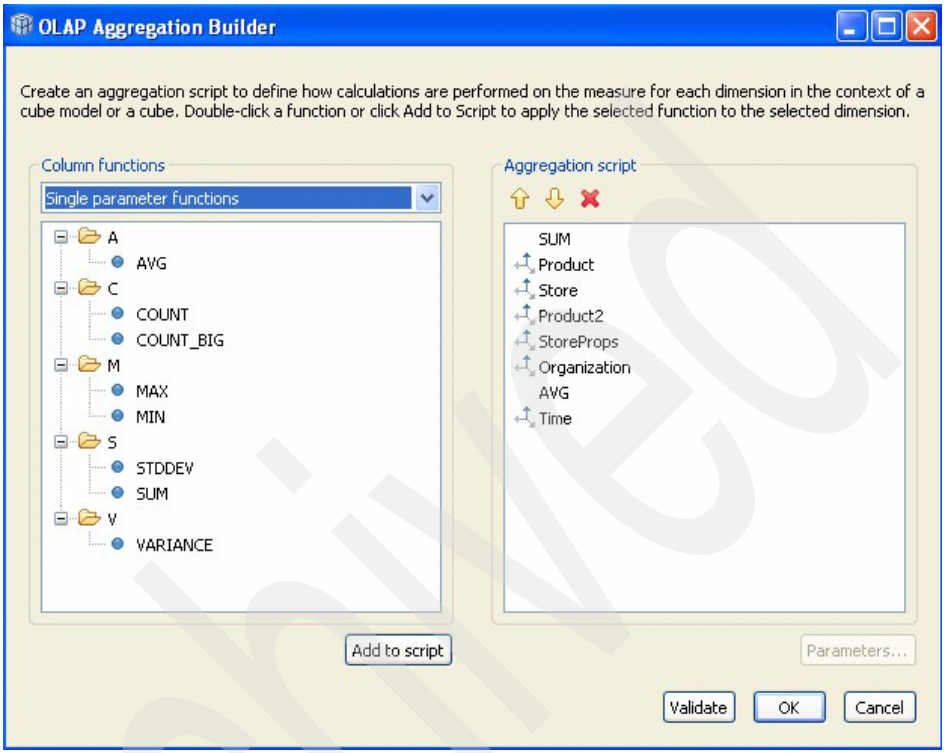


Figure 4-17 Custom aggregation

5

Reporting solutions

After you create a cube, you are ready to create the content that satisfies the business requirements that were identified at the start of your business intelligence project. In this chapter, we outline some of the reporting capabilities that are available within IBM Cognos 8 Business Intelligence that helps you to achieve these goals. The topics that we cover in this chapter are:

- Ad-hoc query
- Analysis
- Authored reporting
- Design approaches for performance
- Common calculations
- Building financial reports
- Dashboarding

5.1 Query styles

The initial business questions for your project can vary or they might focus on specific targets. While the data warehouse might now contain the answers, it is important to deliver the information in a way that makes sense to the end consumers. The information can contain the most fantastic things, but if users cannot adopt the new tools or cannot make sense of the information then the success of the project is at risk.

There are several query styles that can help address the varied needs of your consumers. To categorize these various approaches we put them into four groups:

- Ad-hoc query
- Analysis
- Authored reporting
- Dashboarding

Each of these styles satisfies a particular demand or usage pattern within your user community. IBM Cognos 8 Business Intelligence crafted different interfaces to satisfy these distinct business needs. You can save the reports and analyses created in these studios to the IBM Cognos 8 Cognos Connection portal for easy access and to share the content within your business.

Each of the studios also make use of the same underlying query engine and leverage the same metadata from your Cubing Services environment. This sharing ensures that consumers get a single version of the truth regardless of the methods that they use to display and deliver the underlying data.

5.2 Ad-hoc query

Ad-hoc queries are a less structured and formalized way of obtaining information from your data warehouse. They allow users to ask questions of their own and retrieve information about the current state of the business, for example, a sales representative might want to retrieve a listing of sales leads in their area that were generated in the last six months, or a human resources manager might want to see a list of open positions that took a long time to fill.

These one-of-a-kind questions can be addressed without a complex report that has intricate formatting and a fancy layout. IBM Cognos 8 includes Query Studio, shown in Figure 5-1 on page 109, to satisfy this type of demand.

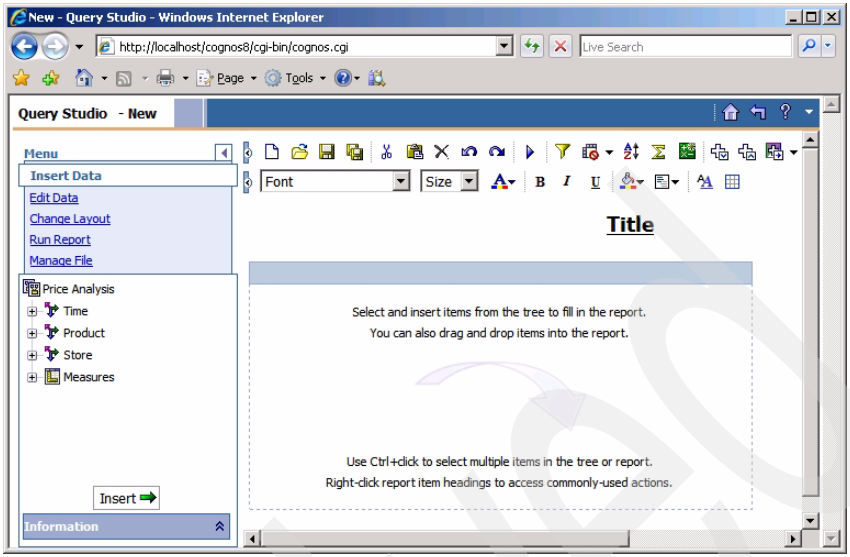


Figure 5-1 Query Studio

The metadata that is available within Query Studio is sourced directly from your Cubing Services OLAP model. By dragging and dropping elements from the model into the report, a user can get immediate results for their own business questions.

The full set of results can be focused into specific data sets in a number of ways. Users can either perform drill actions, as shown in Figure 5-2 on page 110, to drill down and see the members under a given parent, or apply filters to the data, which we show in Figure 5-3 on page 111, to limit the range of data that is displayed.

Drill actions can be initiated by left-clicking on a non-measure value in the results section of Query Studio or by right-clicking and selecting the appropriate direction to either drill up or drill down. The data that is retrieved after drilling makes use of the parent-child relationships and levels within your structured Cubing Services model.

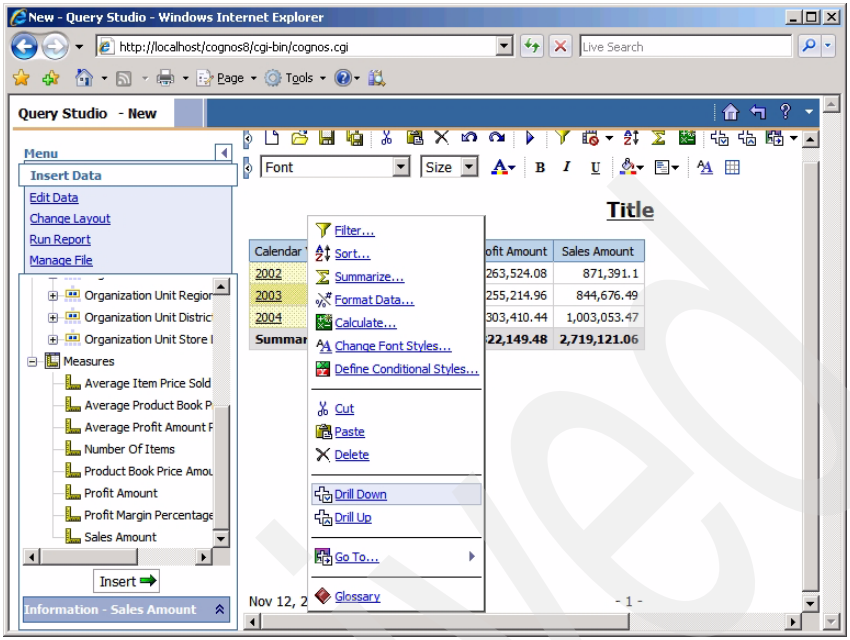


Figure 5-2 Initiating a drill down from year to quarter with right-click context menus

Filters can be applied in addition to using the drill actions. Filters can be added and combined to form powerful Boolean expressions. Individual filter conditions can be added by selecting columns in the output and using the filter button from the toolbar, as shown in Figure 5-3 on page 111.

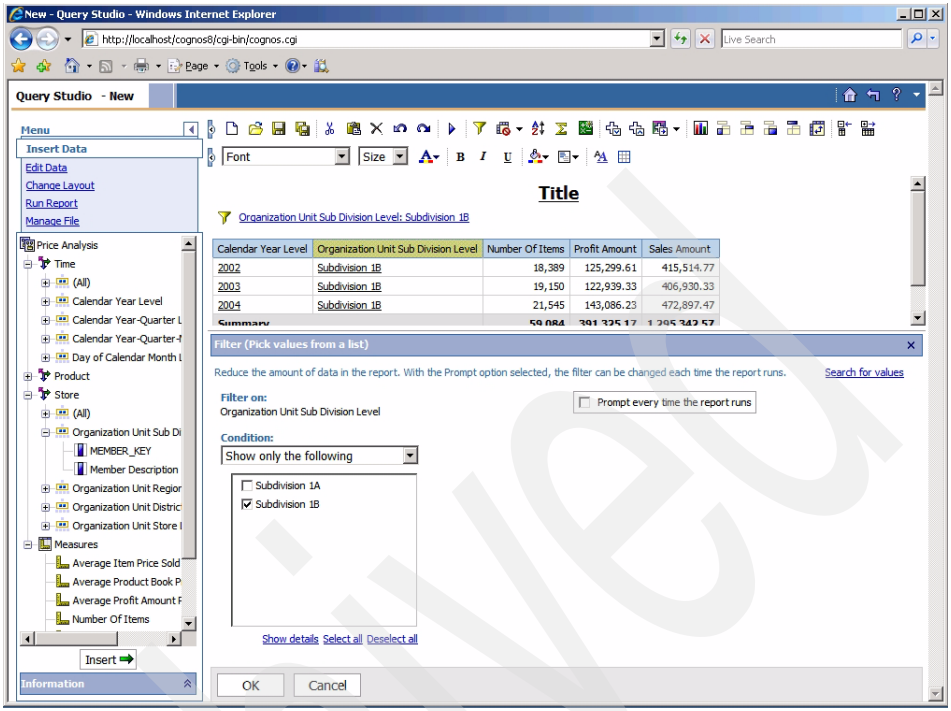


Figure 5-3 Adding filters to a query

There are powerful options that are available when creating a filter. The initial filter criteria can be filtered by selecting individual values from a list, searching for individual items, or by typing in known criteria. The type of input is initially dependent on the type of data that is filtered but can be overridden by an administrator; members initially present a list of values while measures use a type-in input option. By checking the **Prompt every time the report runs** option, a filter expression can be made dynamic and allow a user to select different filter criteria each time the report is executed, which encourages an element of reuse for ad-hoc queries because the same user business question can be applied to different portions of the underlying data.

The Edit Data menu, which is available at the left of the Query Studio window, gives you access to the Combine Filters dialog, Figure 5-4 on page 112. Here you can bring together the individual filter components, add new filters, and change the operators between filters and filter groups, for example, to review sales figures for different stores across different time periods, you might use a filter like the one shown in Figure 5-4 on page 112.

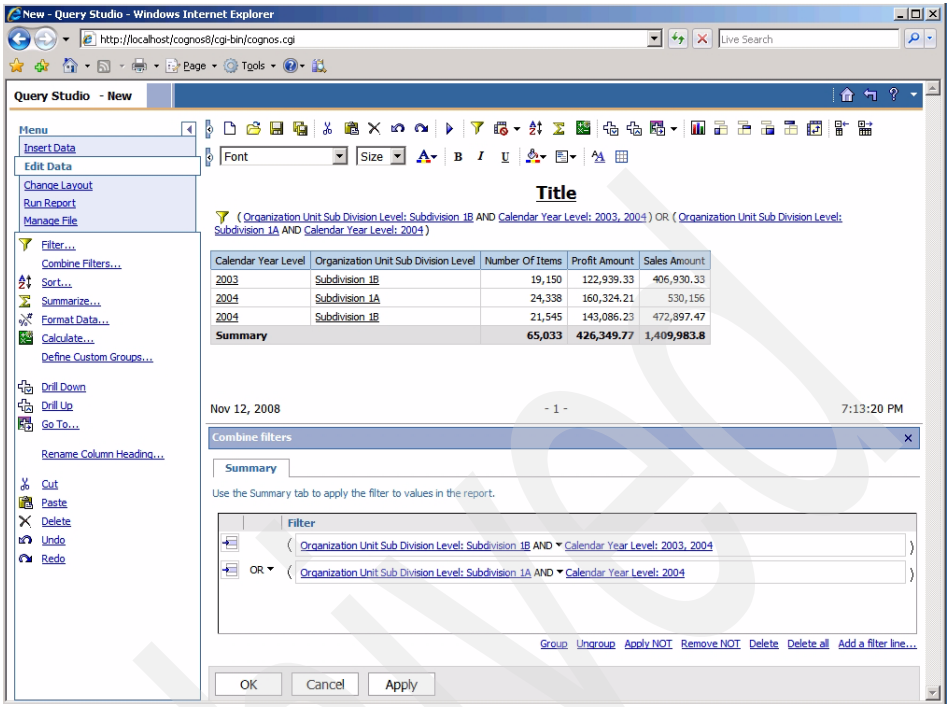


Figure 5-4 Combining filter expressions

On top of these abilities for focusing in on specific data, Query Studio offers conditional styling to bring greater attention to exceptional data. To continue the previous example of store sales, a calculation is created that determines the profit per item sold and highlighting shows values that are below a defined threshold value.

Calculations can be created by selecting the items in the output and using the toolbar button or the right-click option to create a calculation, as shown in Figure 5-5 on page 113. The default calculation options are context sensitive and depend on the type and number of items that are selected (although you can create custom calculations outside of the list provided by the selection context).

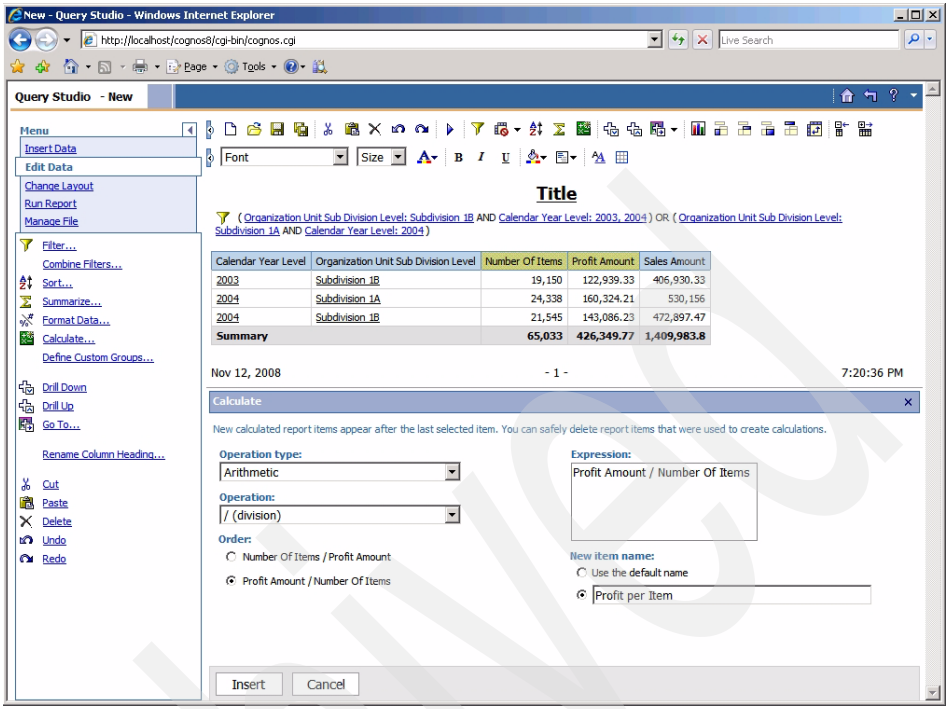


Figure 5-5 Creating a calculation

By selecting **calculation**, and then the **Define conditional styles** option from the Change Layout menu or using the right-click context menu, a conditional style can be applied to highlight the exceptional calculation results. Several types of data selection can be used for the conditional styles. The example in Figure 5-6 on page 114 shows a numeric range applied to the calculation where values below 6.5 are brought into greater focus by applying a predefined style, “poor”, to the list cell. Custom styles and formatting can also be applied by clicking the **Edit** icon to the right of the style example.

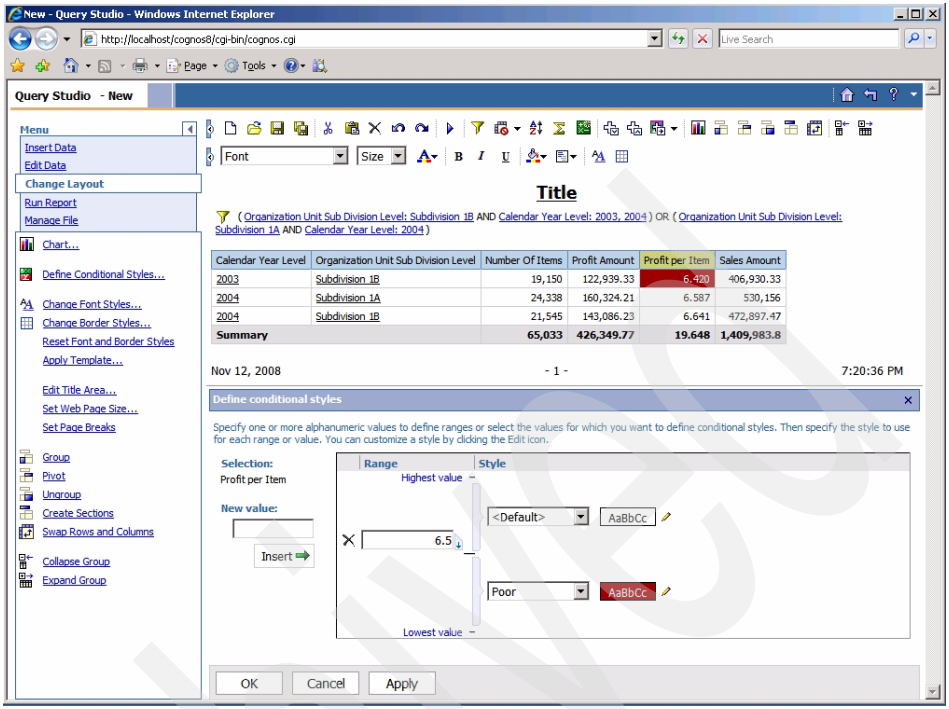


Figure 5-6 Applying Conditional Styles

Additional customization, such as grouping and templates, can extend the look and feel of the queries created in Query Studio. For enterprise projects, it is often useful to have a standardized look and feel to all your queries, and the template options and defaults that an administrator configures can help to achieve this type of formatting standardization. In Figure 5-7 on page 115, we can see the results of these additional formatting options to increase the appeal of the ad-hoc query results that a user created. If a default template was not assigned, you can manually apply a template using the Apply template option from the Change Layout menu.

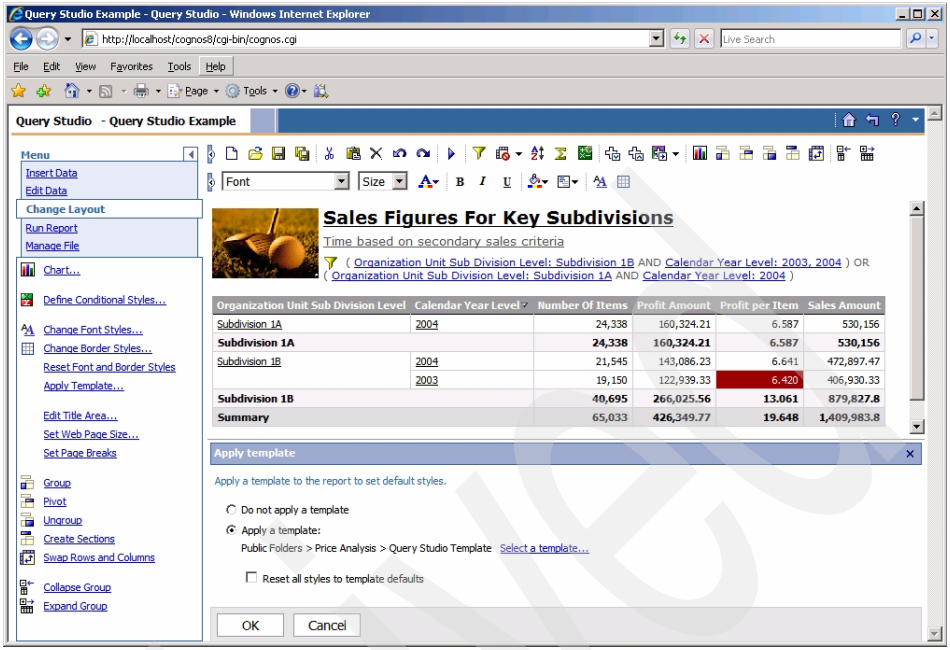


Figure 5-7 Applying grouping, templates, and formatting

Additional display options, such as charts and crosstabs, are available within Query Studio for displaying the query results. These display options are accessible from the toolbar at the top of the Query Studio interface, as depicted in Figure 5-8 on page 116.

A wide variety of charting options exist with Query Studio. While the configuration options in Query Studio are kept simple, you can create charts, such as:

- Column
- Bar
- Pie
- Line
- Column-Line
- Area
- Radar

These chart types come with many options for representing data. You can choose to apply 3-D effects, display your data as stacked (accumulated) across related series, or as a stacked percentage of the total.

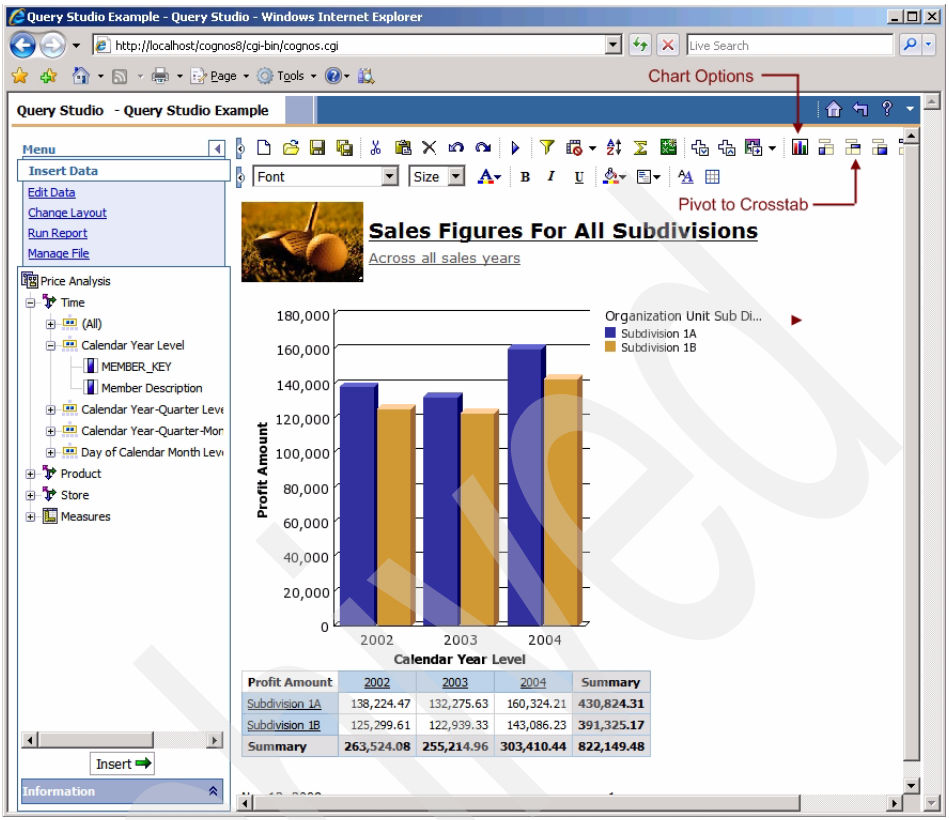


Figure 5-8 Charting and Crosstabs

5.3 Analysis

Analysis is similar to creating ad-hoc queries in that it is an on-demand process from the user community. However, instead of a simple listing of the data as it is, analysis involves exploration and comparisons that go beyond relatively simple data retrieval. IBM Cognos 8 provides Analysis Studio, shown in Figure 5-9 on page 117, to enable the easy slicing, dicing, and investigation of the core business metrics contained within your deployed cubes.

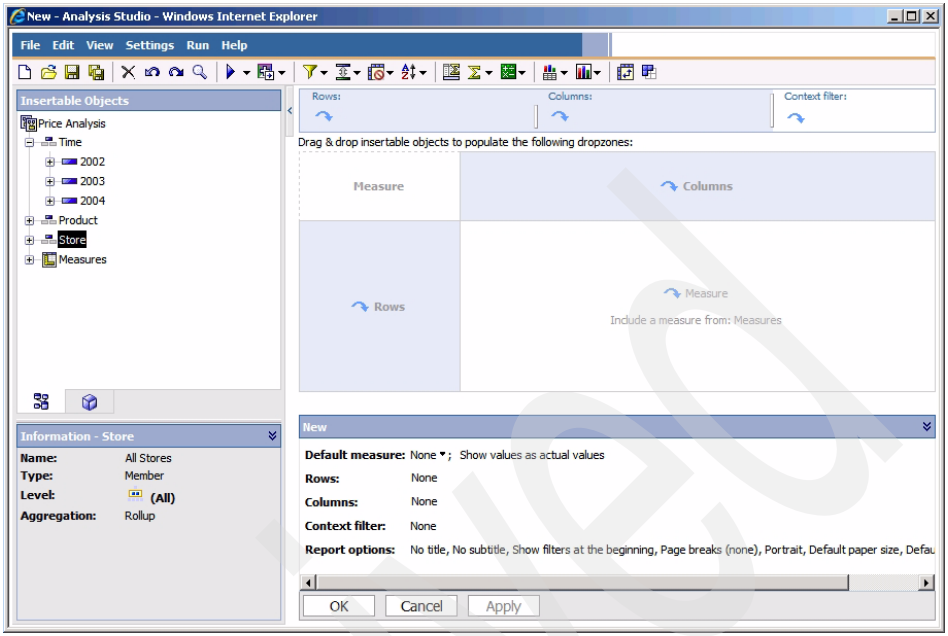


Figure 5-9 IBM Cognos 8 Analysis Studio

Analysis Studio makes use of a set-based approach to manipulating data. Whether you include an entire level or specific members of a dimension, the results are contained within sets that you can manipulate as a single unit and independently of the other sets that are available in your analysis. Working with sets you can focus one aspect of your analysis and compare the results to other independent dimensions in your cube.

The major aim of an analysis is to determine the exceptions and correlations that make up the key drivers of your business process. An exception is defined as a value or set of values that deviates from your established targets or falls outside of (above or below) the normal range of your data values. A correlation identifies patterns, trends, and relationships between the different elements of your cube.

Figure 5-9 shows some of the key components of Analysis Studio. The left pane shows the Insertable Objects from the cube and the expanded Information panel that provides additional information about the selected model items.

The right pane shows the current analysis and any data that is associated with the sets in use. You can create sets by dragging members or the hierarchy to the rows, columns, measures, or context areas of the crosstab or into the drop zones above the crosstab. The context area is used to slice the data by selecting

members that are used to restrict the measure values but is not displayed in the analysis itself.

The Properties pane below the crosstab shows all the properties of the current selection from the analysis. However, because Figure 5-9 on page 117 does not contain any sets in the crosstab, the selection defaults to the entire analysis so the displayed properties are for the analysis itself rather than for any selected set or calculation.

After you define the rows, columns, and measures, the analysis looks similar to the Figure 5-10. Select a set to display the set properties and modify the filtering and display properties.

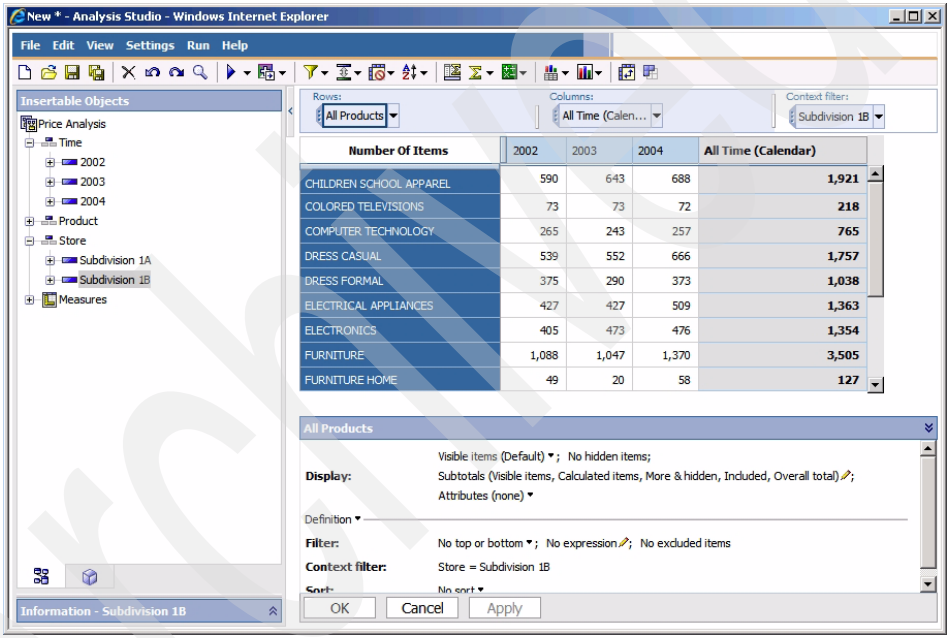


Figure 5-10 Defined analysis with the properties of a selected set

Like Query Studio, a cube data source also supports drill operations that allow you to easily navigate the dimension hierarchies and focus in on the underlying factors that are influencing your bottom line. Additionally, using the Analysis Studio interface you can build complex nesting scenarios, as shown in Figure 5-11 on page 119, that allow you to compare and relate multiple different factors within your data.

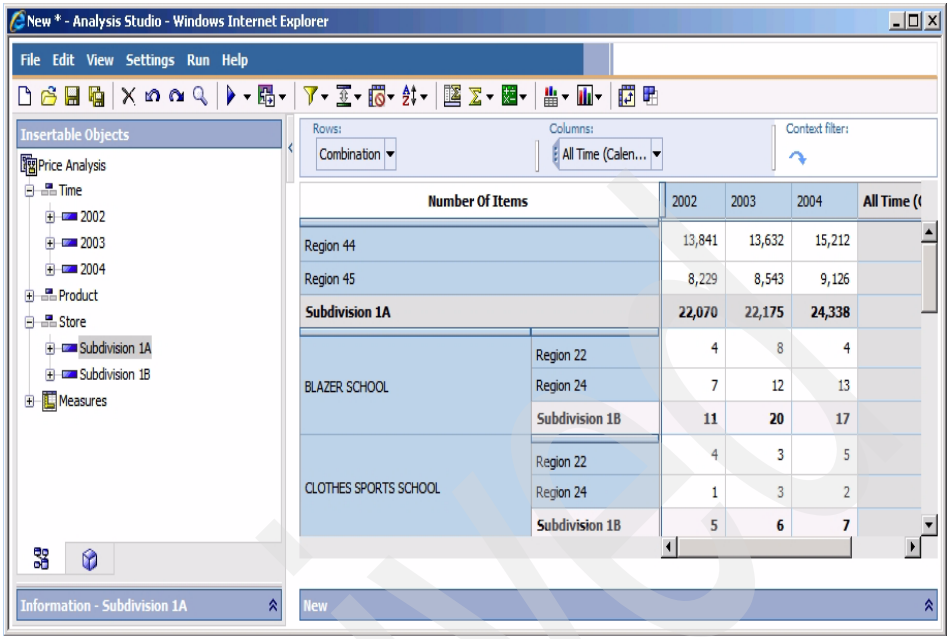


Figure 5-11 Nesting to compare different data sets

Along with these individual features, you can include calculations that become part of your sets, for example, your cube model might not include a calculated member that computes the measure growth across the last two years of your data. Fortunately, this is easily accomplished by selecting the required years in Analysis Studio and using the **Calculation** button from the toolbar or the right-click context menu to generate the growth expression. As in Query Studio, the default calculation options are context sensitive and adjust to the number and type of items selected.

5.3.1 Exceptions

One of the key factors in keeping a business operating is to identify when things are not going according to plan, either for the good of your business or the ones that are detrimental to your business. By identifying the factors of a successful event, you can promote or expand upon them to improve your business success. Likewise, understanding the source of a downturn allows you to adjust your business process to recover from the problem or avoid it in the future.

The top and bottom filters that are available on all sets in Analysis Studio give you a quick and easy way to identify these exceptions within your data. Figure 5-12 on page 120 shows two sets, both based on a large list of product

departments. The top set is modified to show the top three product departments and the bottom set to show the bottom three departments. With a small number of actions, the top performers and bottom performers were called to our attention. From here, you can drill down on these product departments or add in additional dimensions to investigate the reasons for the stellar or not-so-stellar performance.

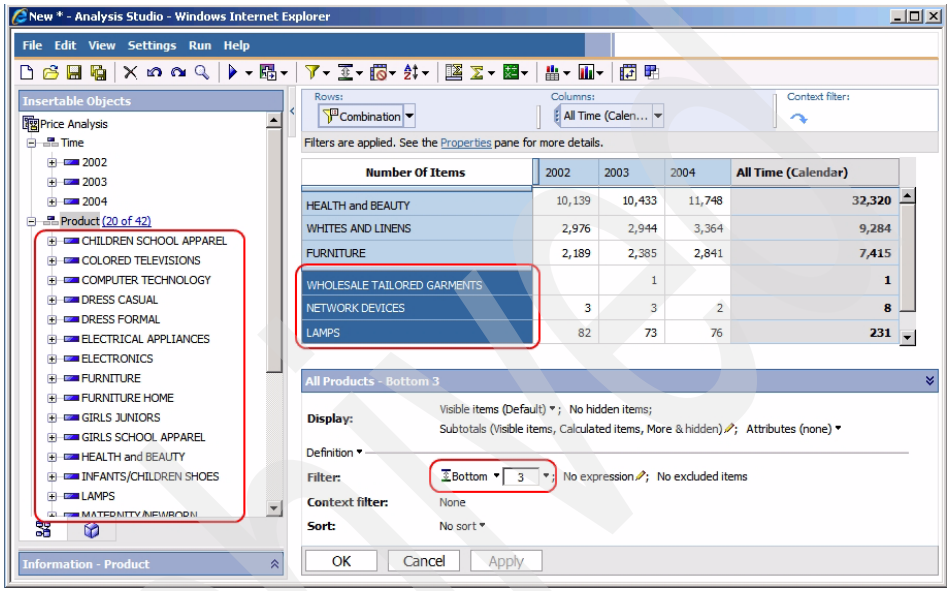


Figure 5-12 Finding exceptions using Top and Bottom filters

Similar to Query Studio, you can define additional filter expressions for each set from this Properties pane. In the filters area, click the Edit icon to add, group, and combine Boolean filter expressions into additional powerful criteria to focus in on key data.

Likewise, conditional styles are also available in Analysis Studio as they were in Query Studio. You can use this tool to call attention to key points in your data as you navigate through your cube, for example, use conditional styles to identify when the profit margin drops below 5%, and as you drill or swap dimensions the visual cue of the style draws immediate notice to the jeopardized profits.

5.3.2 Correlation

The flip side to identifying exceptions is to find combinations of factors that relate to each other. After an exceptional scenario is identified you might want to track

its development across other dimensions or compare within different areas of the same dimensions.

Say for example that you used a Top filter to identify your top five product departments in 2004 based on the Number of Items Sold (and all stores because the store dimension is not set in the analysis), as shown in Figure 5-13. You can then set the “Hold current context” to fix this set definition for the top five, which allows you to then add or replace the other sets in your analysis to view the performance of the top five product departments in different ways.

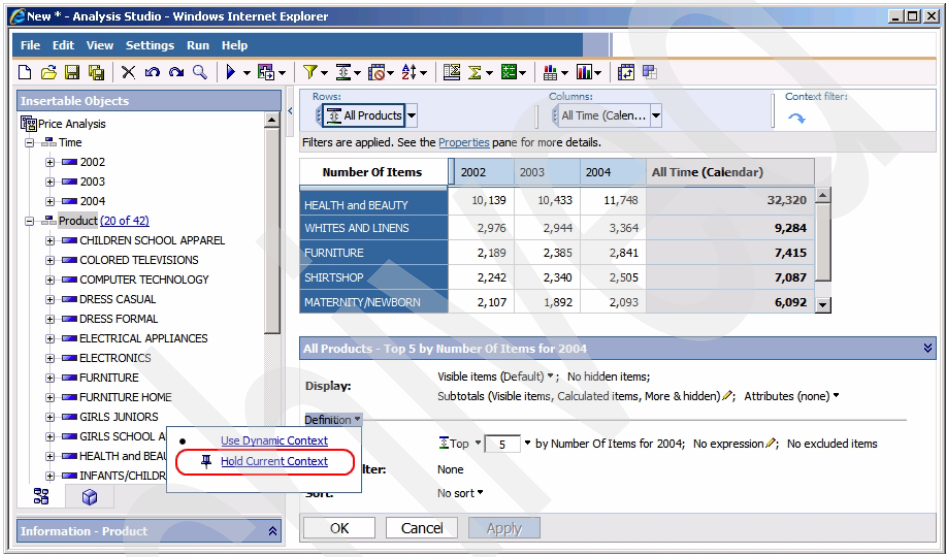


Figure 5-13 A custom top five filter with fixed context

By swapping the measure reference in the analysis and drilling down on 2004 to see the individual quarters of the year, we can see how the top five product departments by items sold in 2004 generated their profits by quarter, as shown in Figure 5-14 on page 122.

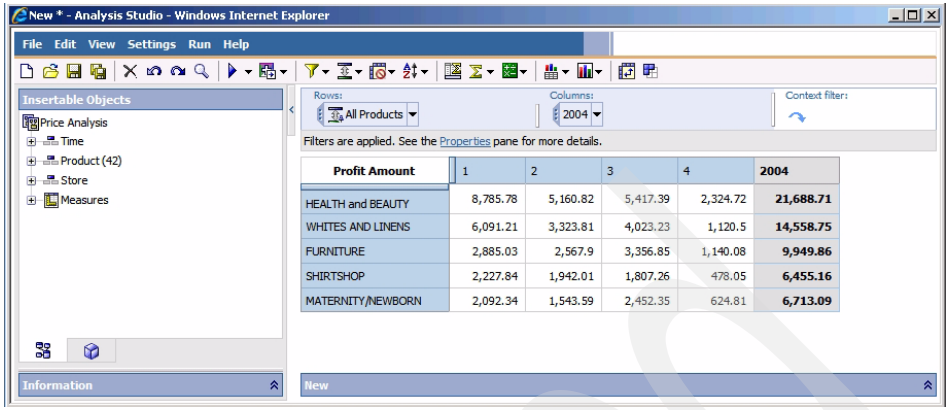


Figure 5-14 Navigation with fixed context

By the results, we can see that the top five product departments had strong first quarter profit that carried their performance for the year. Unfortunately, the end of the year profits seem to trail off, leading us to other questions as to the factors that contributed to the strong start and the weak finish of these business areas.

Just as easily, we can drag the fixed top five product departments into the Context navigation area, and then use this to restrict (slice) the measure values when viewing the data for other dimensions from the cube, as shown in Figure 5-15. Now we can analyze the Stores in our sample retail data based on their performance with the top five product departments in 2004.

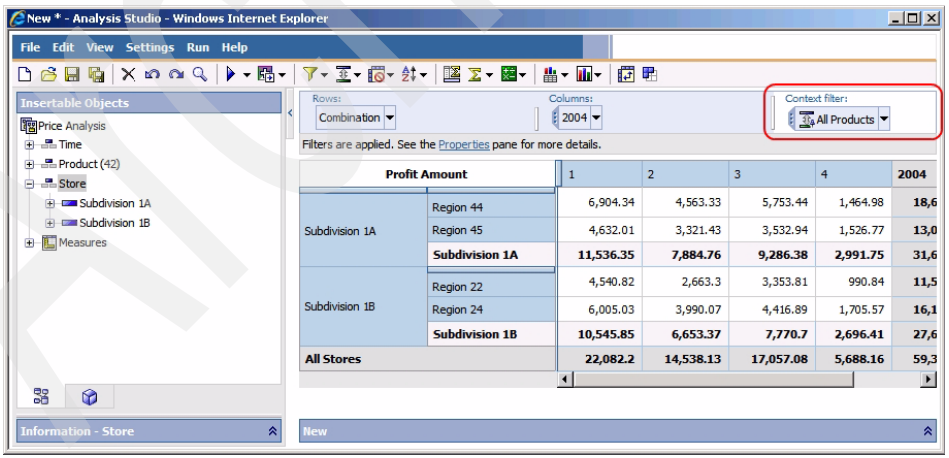


Figure 5-15 Using the fixed context set to restrict measure values

5.3.3 Charting

Although the tabular numbers in a crosstab are excellent for lookups and individual comparisons, there is great value in being able to represent the data graphically. Like Query Studio, powerful charting features are available to analyze the exceptions and trends within your data. Click the **Charting** button from the toolbar to transform the crosstab information into all of the same chart types that are available from Query Studio, pareto, and point charts.

Working with the previous top product departments scenario, by using a line chart we can easily see the trend of the data across the quarters of 2004, as shown in Figure 5-16. While the crosstab data shows us the base numeric values, the chart shows the magnitude and shape of the data in a way the simple numbers cannot manage.

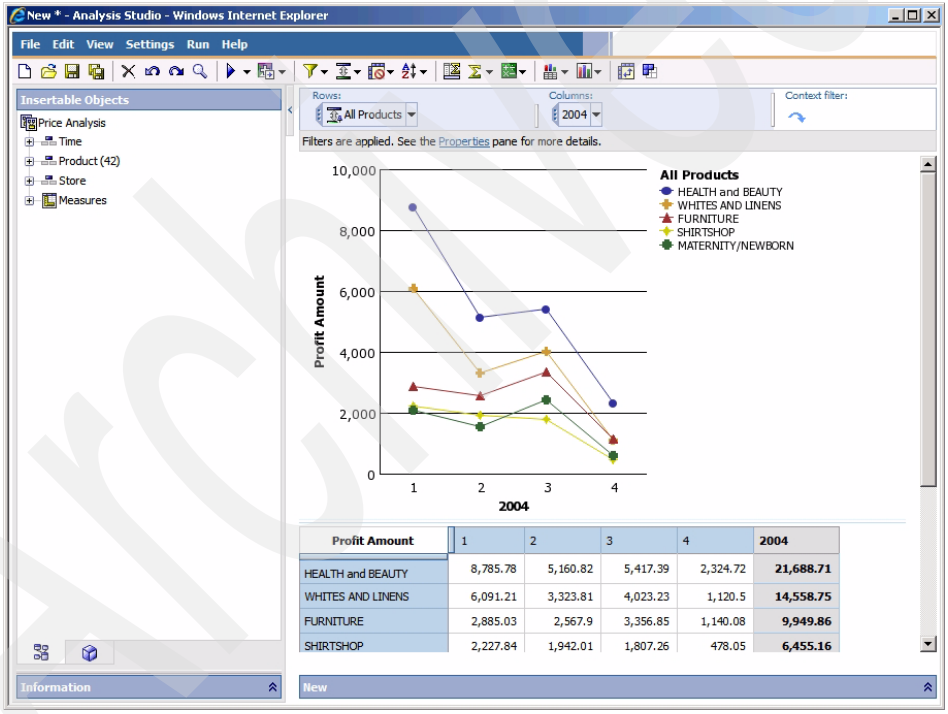


Figure 5-16 Charting in Analysis Studio

If we wanted to investigate the data further, the linked drill capabilities on both the chart and the crosstab brings us the detailed information that we require.

5.4 Authored reporting

Ad-hoc queries and Analysis satisfy the demand for exploration and data delivery that have no well-defined or specific business requirements. However, many of your project requirements might have specific delivery specifications that can include more involved or additional queries, calculations, and layout formatting than is necessary in an ad-hoc scenario.

Because the business requirements for this type of reporting are fixed ahead of time, you can tailor the report to this specific need, which introduces the concept of *Authored Reporting* where a report is developed one time and then executed on demand or when the cube data is refreshed to render the latest data values within the same report format.

Use IBM Cognos 8 provides Report Studio, Figure 5-17, as a means to achieve these more in-depth query and layout capabilities.

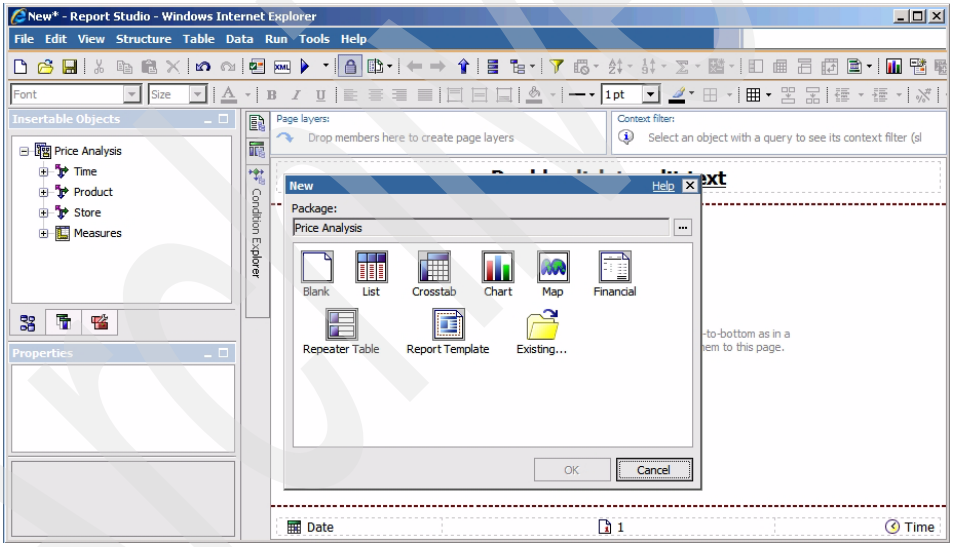


Figure 5-17 IBM Cognos 8 Report Studio

You can use Report Studio to open reports that you generated in Query Studio and to open analyses that you created by Analysis Studio. This flexibility gives you the option of using an existing investment from an ad-hoc query to generate a presentation-ready report with additional layout and formatting options. However, when you start new report development, we recommend that you begin within Report Studio. Query Studio and Analysis Studio might generate additional elements within a report that, although they support complex

operations in Query Studio and Analysis Studio, might not be necessary to the targeted report goals of Authored Reporting.

5.4.1 Layout and formatting

While a template defines the initial layout of a Report Studio report (either one of the default templates or a custom template), this does not restrict an author from creating additional layout objects or from applying additional styling. This is different from ad-hoc queries where the layout is fixed based on either a single list or crosstab with the option to display the same data in chart format.

The toolbox in Report Studio provides a large number of layout objects that can be dragged into any location of the report. These include objects such as:

- Tables
- Blocks
- Prompts
- Text
- Buttons
- Images

These are just a small subset of the layout options that are available to a report author. However, these basic few items can be used to create quite interesting and appealing report layouts for presenting data. After the layout is established, an author can proceed to include some of the many data containers also available from the toolbox. Refer to the Report Studio User Guide for additional information:

- List
- Crosstab
- Chart
- Map
- Repeater Table
- Repeater
- Singleton

These data containers can be added to show multiple different perspectives on your Cubing Services data.

On top of these additional objects, report authors in Report Studio have significant fine-grain control over the properties of each object. While Query Studio and Analysis Studio controlled much of the formatting, Report Studio exposes all these various properties for use in your reports, for example, when you select a crosstab, the properties pane at the bottom left of the Report Studio window updates to show information such as that depicted in Figure 5-18 on page 126 (a subset of the available crosstab properties).

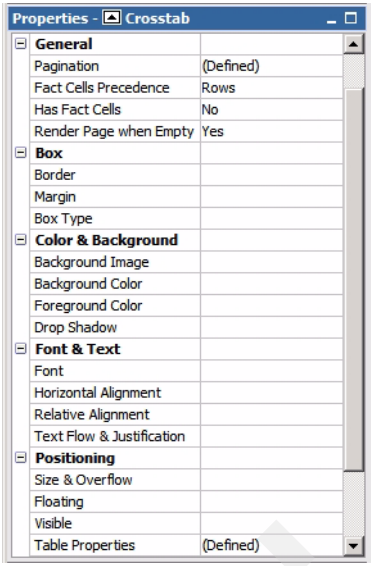


Figure 5-18 Crosstab formatting properties

The properties that are available on individual objects depend on the object type, for example, an image does not have font properties but it does have border, margin, and box type properties, as does a crosstab.

All of these properties can be customized by a report author. If you have many objects, this type of property manipulation might seem daunting at first. However, objects inherit formatting properties from higher levels within the same container, for example, applying font properties to a table applies the same properties to each cell of the table without needing to explicitly define each cell. Similar effects can be obtained with the other objects within a report, making formatting a relatively easy process.

Another feature to report authors are the style classes that are available within Report Studio. These classes bundle formatting properties together under a common reference, and they can be applied to the various objects that are included in the report layout. With a single style class, you can format multiple objects and because classes are dynamic, you can update the class definition and have all of the associated report object formats updated in a single step.

The end effect of all this layout and formatting is that the authored reports can be made quite attractive and appealing, such as the one in Figure 5-19 on page 127, which guides the report consumers to the data they must make decisions.

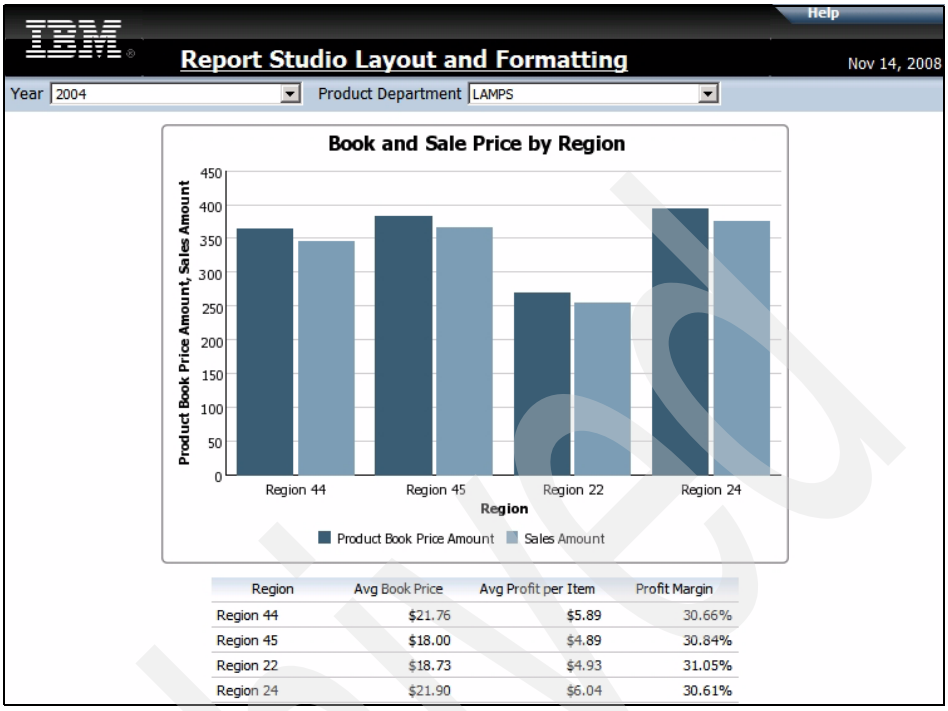


Figure 5-19 Layout and formatting with Report Studio

Likewise, an author can choose to define a dashboard with a series of charts, rather than lists and crosstabs, that provide the at-a-glance view of key business metrics.

5.4.2 Multiple queries

One of the key differentiators of Report Studio is that it allows multiple queries to be included in a single report. While a single ad-hoc question is answered by a single query, authored reporting might work with several different data sets and or views of the same data to deliver the necessary information to a report consumer.

Each list, crosstab, chart, and so forth, added into the Report Studio layout can share a single query or be set to use different queries. This sharing allows you to bring together multiple cubes or link relational data into your reports to expand on the data available from Cubing Services.

Strictly speaking, it is not necessary to use multiple queries in Report Studio when reporting from a single cube source. The set-based nature or a

dimensional query can be leveraged to apply different filter criteria in different data item expressions. These filtered items can then be used as required on your layout without impacting the filter expressions used by other data items from the same query object. At run-time, the report layout is evaluated and, based on the items used, the query engine sends the appropriate request to the underlying data source, which means that a single query object, using a common cube source, in Report Studio can be used to satisfy multiple components on the report layout.

It is also possible to establish links between different queries. These are known as Master-Detail Relationships. The idea is that one query generates a master set of query results that are then used as input criteria for a query containing more detailed information. When the master-detail relationship is used, each row from the master query is executes a new version of the detail query. Depending on how complex the queries are this might result in a single large query with the master data as an outer level of nesting or the detail queries might be executed individually to retrieve the required results, for example, if you want to generate a chart of the yearly sales trend for each store. You can generate a master query for the stores you are interested in and pass the store information to a detail query used to chart the sales figures over time. In Figure 5-20 a list that is associated with a query named Key Stores is linked to another query called Sales Trend, which contains the time and measure information. The linkage is defined by using the Master Detail Relationships property of the detail query.

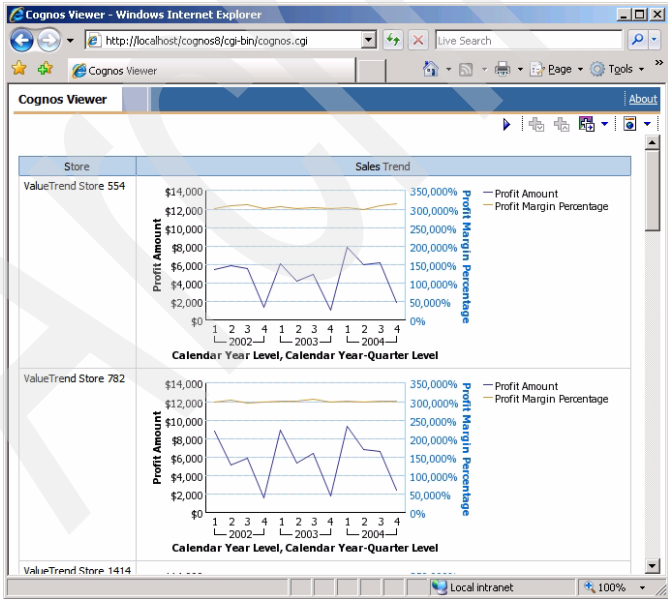


Figure 5-20 One chart query reused through a Master-Detail Relationship

Using the layout in Report Studio, you get significant freedom in where and how the multiple queries are rendered. Master-detail relationships are one way of linking the queries but there are other methods of coordinating the information.

IBM Cognos 8.4 allows you to link drill operations across conformed dimensions that are used in different queries, which means that when multiple queries share the same cube as their data source they also have the same dimensions available within each query. These shared dimensions can be linked for drill operations (configurable from the Drill Behavior option available from the Report Studio Data menu). The end result is that a drill down or drill up for one query can be used to change the corresponding dimension context for the other queries in your report. Figure 5-21 shows a drill down on 2003 from the chart in a report. The results of the drill, shown on the right, display the quarters of 2003 in both the chart and crosstab.

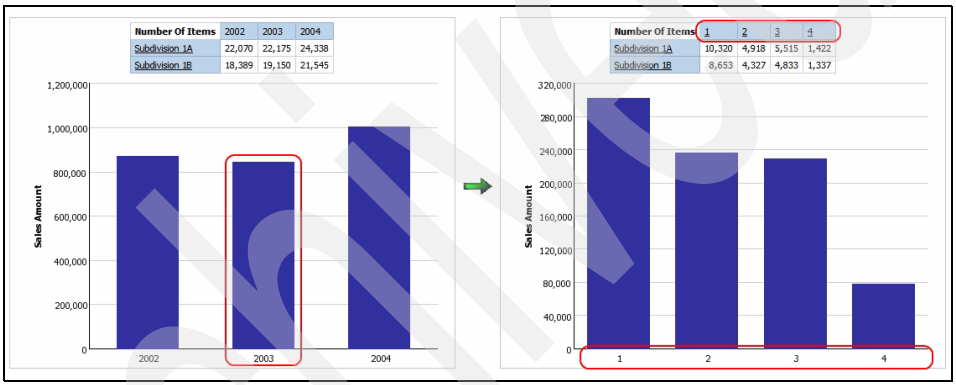


Figure 5-21 Linked drill between queries

Combining additional layout capabilities, you can group different queries into pages, define page breaks based on data values, and customize the report to meet a common business requirement for multiple different user communities.

5.4.3 Calculations

Similar to the extended control over the layout and formatting, Report Studio also offers greater control over calculations and the expressions used to define the data elements in your report. The expression editor, Figure 5-22 on page 130, provides an extensive list of functions that can be used and combined in powerful ways. The function list contains both dimensional functions that are passed through in the queries sent to Cubing Services and tabular functions, which are processed by the IBM Cognos 8 query engine to extend your query capabilities beyond the base expressions available from Cubing Services.

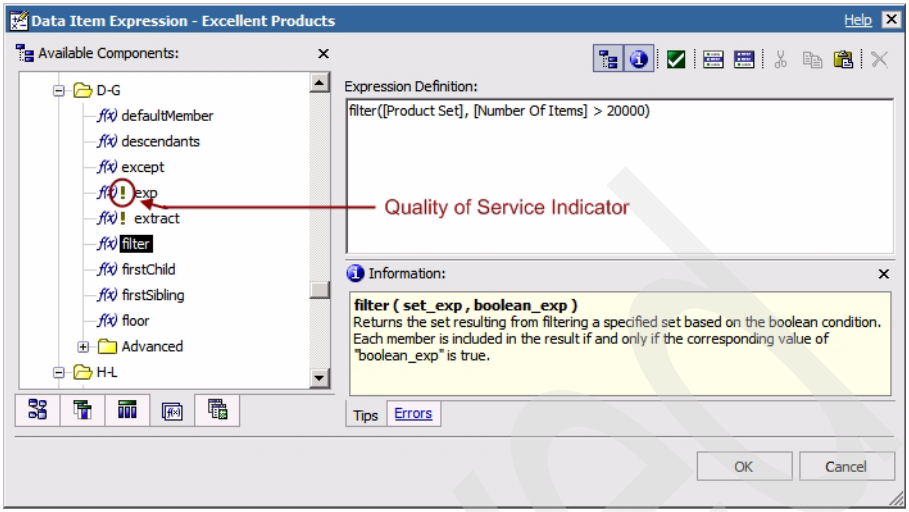


Figure 5-22 Report Studio expression editor

Quality of service indicators next to the function name in the function list indicate which functions are passed down to Cubing Services. When possible, it is better to leverage the high-speed caching and calculation capabilities of Cubing Services. Making use of the native dimensional functions that Cubing Services supports helps to streamline query planning within IBM Cognos 8 and often benefits report performance.

5.4.4 Prompting

Previously, with Query Studio and Analysis Studio, you define a prompt based on filter or context criteria that allows a user to apply the same query definition to multiple different portions of the complete data available from Cubing Services. Report Studio allows these same operations but extends these options by providing many additional prompt types and allows you to use them in any of your layout or query expressions.

Prompting with layout expressions allows you to conditionally style or conditionally display different portions of your report. Prompting with data item expressions allows nearly limitless freedom in accessing data. The prompt values can be used to slice or filter data, but they can also be used as arguments in the dimensional functions available in the expression editor.

The basic syntax for parameters that provide values in a query uses question marks to surround the name of the parameter, for example, a parameter named “Sales Threshold” would be used in an expression as *?Sales Threshold?*

If a prompt control is not created for the parameter by a report author, the type of prompt control and the values that are retrieved to populate it depends on the context in which the prompt is used. If, for example, the parameter were used in a Boolean condition with a measure then type-in prompt would be presented to the users. The available value prompt types are:

- Text Box
- Value
- Select & Search
- Date & Time
- Date
- Time
- Interval
- Tree

Additional options are available if a report author wants to construct their own prompt control using HTML or javascript. These types of controls can be inserted into a report page or a prompt page in Report Studio using an HTML object.

When defining sets or selecting individual members in a query, an variation on the parameter syntax can be used. For this type of prompting, you can supply a level or hierarchy reference before the prompt to identify that a member or set of members is to be returned. The set() function is used around the parameter reference to specify that multiple members, a set, is returned from the prompt control. These variations take the following form:

- Single member from a single level:
  - `[Cube].[Dimension].[Hierarchy].[Level]->?Parameter Name?`
- Multiple members from a single level:
  - `set([Cube].[Dimension].[Hierarchy].[Level]->?Parameter Name?)`
- Single member from the complete hierarchy:
  - `[Cube].[Dimension].[Hierarchy]->?Parameter Name?`
- Multiple members from the complete hierarchy:
  - `set([Cube].[Dimension].[Hierarchy]->?Parameter Name?)`

The default prompt control that is generated for these is a value prompt for level-based parameters and a tree prompt for hierarchy-based parameters, as shown in Figure 5-23 on page 132.

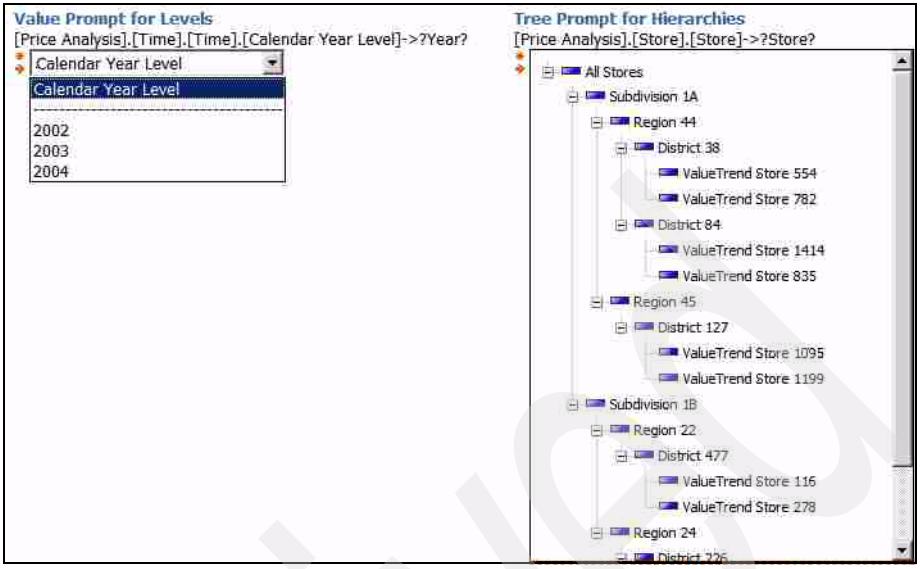


Figure 5-23 Dimensional prompt controls

With these prompting mechanisms, a report author can define a single report that satisfies a number of similar business requirements. The prompts can be combined and used to provide a series of pathways for users to navigate to the specific data that they require.

5.4.5 Bursting

Often, a single business requirement can become complicated by the delivery requirements among different consumer groups, for example, a project might require that a report be produced showing the sales figures for the business. However, each regional group might need to see their own figures rather than all the figures for all the regions in the organization.

In this type of situation, a report author can use a prompt to select the various regions, but this forces consumers to run the report and select their region each time. On top of adding unnecessary work for report consumers, the on-demand nature of the report places additional load on IBM Cognos 8 and Cubing Services to satisfy the requests.

A better option is to make use of bursting in Report Studio, which is similar in effect to the master detail relationships illustrated in section 5.4.2, “Multiple queries” on page 127. You can use a master query to define the different data groups that the single report supports. This master query is then used in the Burst Options settings from the Report Studio File menu. When scheduling this

report from Cognos Connection, select the bursting option to generate individual output files for each of the key data values in the burst query. Returning to the example scenario, the burst query can be defined with the list of regional groups and then a single output file is generated for each Region.

An additional recipients query, using either fixed references to security objects (groups or user accounts) or a query to a relational lookup table, can be defined in the burst options. The recipients query can be used to secure the report output so that each group has access to their own specific report output and also for delivery of the report output by e-mail notification.

In this way, one report satisfies multiple groups. On top of this, the bursting process reduces the server load by pre-generating the report output so that it is available when the users need it and avoiding the need to execute the queries each time a user accesses the report.

5.5 Design approaches for performance

Query performance can be impacted in many different ways. In the chain of events for executing a query, there are ways to improve performance from disk access all through database design, cube design, and cube server settings, to the query that is created by a report author.

While an administrator must be involved for many of these stages in the process, there are several concepts and approaches that a report author can use to keep queries performing well.

5.5.1 Crossjoins

A *crossjoin*, also known as a cartesian product, is a fairly common operation when working with a dimensional data source. Any time nesting occurs between different dimensions, a cross product occurs. In Example 5-1, a list that shows Year and Region generates a cross product to match each region to each year. A list projects all the columns onto a single query axis. The columns are nested from the left to the right so the right column is part of the inner most crossjoin in the query.

Example 5-1 Simple cartesian product

```
Consider two sets from two different dimensions (Time and Geography).

Set 1: Years
```
- 2008

- 2009

Set 2: Region
- Asia Pacific
- North America
- Europe

The crossjoin result will become:
- 2008, Asia Pacific
- 2008, North America
- 2008, Europe
- 2009, Asia Pacific
- 2009, North America
- 2009, Europe

The size of the result set is equal to the product of size of the Year and Region sets. The Year set contains 2 members and the Region set contains 3 members so this produces 2x3=6 tuples in the result.

The crossjoin itself is not necessarily bad. However, with extensive nesting the result of all the crossjoins can become large, for example, if an author were to nest three sets from different dimensions that each contained 100 members then the resulting crossjoin would produce 100x100x100= 1 Million tuples.

With this type of scenario it is quite easy to produce result sets that are considerably larger than required to satisfy a user's request. These large result sets must be generated by Cubing Services and transferred to the IBM Cognos 8 server for processing and rendering.

Being aware of this type of scenario, an author can implement filtering on the individual sets to reduce the size of the full cross product. Often, there is some degree of data sparsity that can be eliminated or the business requirements might only be focused on identifying the top or bottom elements in these sets. If the previous three sets of 100 members were filtered down to 20, 70, and 90 members, the result set would be 20x70x90=126000 tuples, which is considerably less than the previous 1 Million tuples from the unfiltered results.

5.5.2 Filtering

There are several ways to apply filters within IBM Cognos 8. In section 5.2, "Ad-hoc query" on page 108, we saw the filtering methods that are available for Query Studio, where Boolean expressions can be created and combined to restrict a tabular result set. In section 5.3, "Analysis" on page 116, we saw the filtering properties available on sets in Analysis Studio that would allow individual

components of an analysis to be restricted independent of the other sets that are used.

Report Studio offers these same methods, and more, to report authors in the form of detail filters, context filters, and dimensional functions. However, the added freedom available within Report Studio means that it becomes easier to apply expressions that can adversely affect performance.

Detail Filters

You can create a detail filter by selecting a data item in the layout and clicking the **Filter** button from the toolbar or by using Query Explorer to select the query and add a filter from the toolbox into the Detail Filters drop zone of the query, as shown in Figure 5-24.

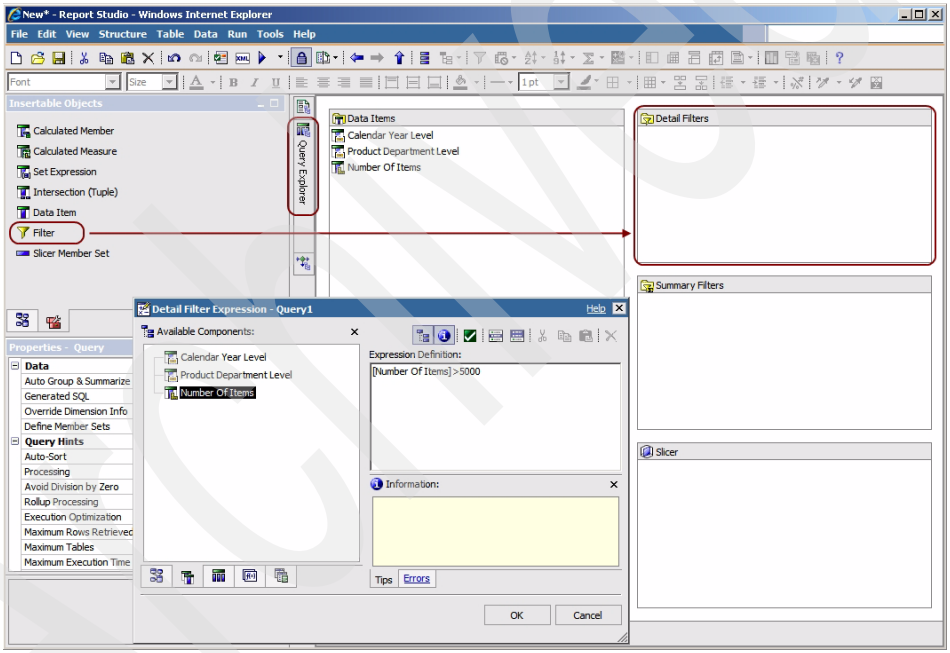


Figure 5-24 Detail Filters

Where possible, the filter is applied to the given dimension. However, there are some cases where the detail filter can add undue complexity to a query against Cubing Services. This happens because a detail filter is applied to the entire query rather than individual sets within the query.

When working with multiple sets, and a filter using the same dimension, the filter is applied to all sets using the same dimension, which limits all of the sets that use this common dimension to match the filter criteria.

Additionally, you must evaluate some detailed filter expressions after computing intermediate result sets. If, as in Figure 5-24 on page 135, a filter is applied to a measure in the query, the unfiltered result set must first be retrieved before the measure values are available. The final result is filtered but the intermediate result set still requires additional resources to transfer and apply the filter criteria. When working with large data sets, as can occur with nesting of several large dimensions, a detail filter might not provide any direct performance benefit when executing the query.

Context filters

You can use a *Context filter* or slicer to restrict the query results using members or sets. These are created by dragging a Slicer Member Set into the Slicer area, as shown in Figure 5-25, of a query or (for all queries in a report) by using the Context filter drop zone on the report layout. The Context filter area is available at the top-right of the page layout in Report Studio by default in Express Authoring mode and can be enabled in Professional Authoring mode by selecting Context Area from the Panes option in the View menu.

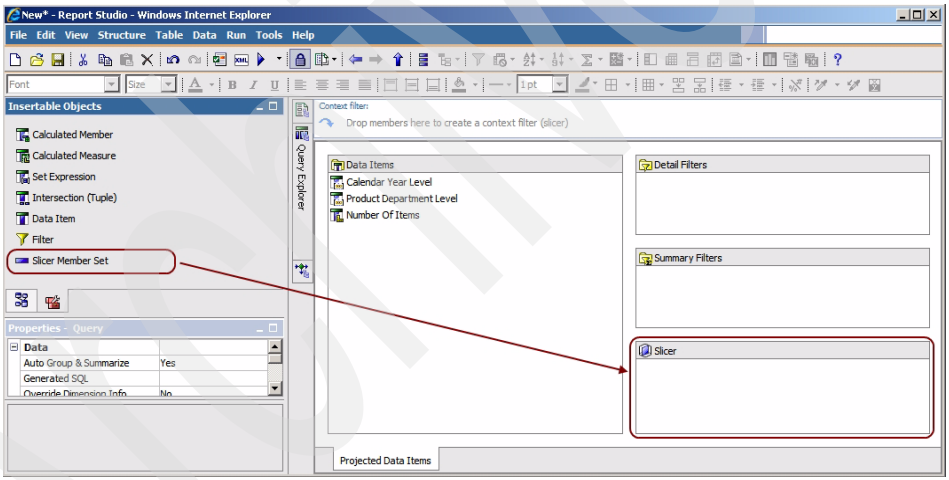


Figure 5-25 Slicers

After dragging a Slicer Member Set into the Slicer drop-zone, select the member(s) from the cube, or define an expression that produces a set of members. The member or set is used to restrict the measure values that are presented on the report layout or that are used within expressions.

Typically, a slicer is used to restrict figures based on a dimension that is not present in the report layout. Applying a slicer to a dimension that is on the layout restricts the measure values that are displayed, but this is often not a desirable result for reporting. Also, the expressions required to project the same dimension

onto an edge in the layout (such as the row or column areas of a crosstab) while also restricting the measure values can become complex. For this reason, it is often better to use slicers only when the members of the slicer set are not also used in the report layout.

Figure 5-26 illustrates the difference in results when the slicer dimension is on the layout and when the dimension is not on the layout. When the slicer dimension is on the layout, the measure cells are predominantly blank because the measure values are only displayed when they match the slicer members.

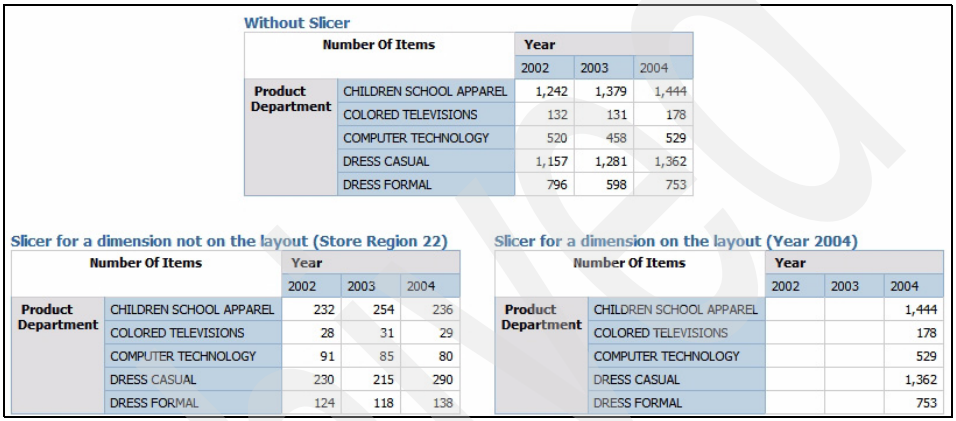


Figure 5-26 Slicer results

Filter functions

By far the most flexible method of filtering dimensional queries is to use one of the many functions that are available for Cubing Services to restrict sets defined within the query. Where detail and context filters are applied to the entire query, a function result is only applied within a single data item expression, which allows an author to create individual sets from the same dimension that have considerably different filter criteria.

Some of the key functions used to restrict a set are:

- Filter
- TopCount
- TopSum
- TopPercent
- BottomCount
- BottomSum
- BottomPercent

Each of these key functions takes a larger set as an input argument and returns a (potentially) smaller set created from the members that match the function

criteria. The filter criteria for the Filter function are based on Boolean expressions. TopCount and BottomCount retrieve a specified number of members associated with the largest or smallest measure values respectively. TopSum and BottomSum return the members that cumulatively form the largest or smallest measure sum, respectively, that meets a numeric threshold value. TopPercent and BottomPercent operate by retrieving a cumulative percentage of the total calculated from the greatest or least measure values, respectively, that meet a threshold percentage of the total set value.

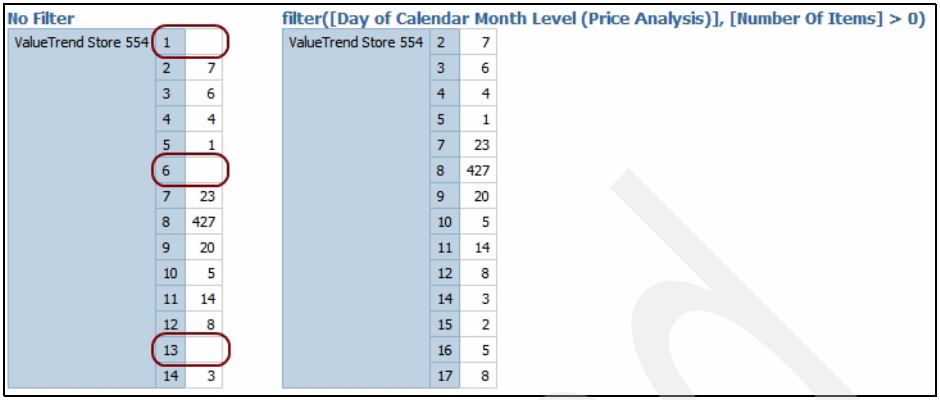
In Figure 5-27, two expressions are added to a crosstab. The first expression uses a TopCount function to retrieve the three stores that sold the most items. The second expression uses BottomCount to return the three stores that sold the least number of items.

Top 3 Stores	topCount([Organization Unit Store Level], 3, [Number Of Items])
ValueTrend Store 1199	13,147
ValueTrend Store 1095	12,751
ValueTrend Store 116	12,481
Bottom 3 Stores	**bottomCount([Organization Unit Store Level], 3, [Number Of Items])**
ValueTrend Store 554	9,577
ValueTrend Store 835	10,085
ValueTrend Store 875	11,149

Figure 5-27 Using TopCount and BottomCount in different sets in a crosstab

Applying these two functions reduced the set of stores from 11 down to the six that are the most interesting to our sales figures.

However, in most cases a report author uses the Filter function in Report Studio, which is the key to improving query performance for large result sets, as we described in section 5.5.1, “Crossjoins” on page 133, and is also useful in suppressing zero or empty measure cells. In Figure 5-28 on page 139, we can see that the ValueTrend Store 554 did not make sales on each and every day of the sample time dimension. By applying a filter function to the set of days, the empty cells can be eliminated from the query results. Due to the nesting of the function beneath the store member the measure values for the filter function are evaluated within the store context without needing to specify the store in the filter function expression, which makes the filter expression portable and reusable under other stores or even under members from other dimensions.



No Filter		
ValueTrend Store 554	1	
	2	7
	3	6
	4	4
	5	1
	6	
	7	23
	8	427
	9	20
	10	5
	11	14
	12	8
	13	
	14	3

filter([Day of Calendar Month Level (Price Analysis)], [Number Of Items] > 0)		
ValueTrend Store 554	2	7
	3	6
	4	4
	5	1
	7	23
	8	427
	9	20
	10	5
	11	14
	12	8
	14	3
	15	2
	16	5
	17	8

Figure 5-28 Filter function to remove empty cells

Attributes and members

When applying filters either as detail filters or filter functions it is possible to use either member references or member attributes in the Boolean expressions. Typically, member attributes are not structured for fast retrieval, which can have a performance impact if they are used for filtering, for example, a product dimension might have products that have a package size attribute. It is not unusual for users to want to use the package size for filtering. However, as an attribute, the query engine must perform extra processing to identify all the members with the matching attribute properties for the filter. The effects of this on query performance varies depending on the size of the dimension involved.

For this reason, where possible, perform filtering using member references rather than using member attributes. By using member references the query can make better use of the indexes and the member cache within Cubing Services. If filtering or grouping based on attribute values is common, there might be justification for redefining the attribute as either a new level in the existing dimension or as a new dimension within the cube model.

5.5.3 Local processing

When an entire query cannot be submitted as a single request to Cubing Services there is some amount of processing that IBM Cognos 8 performs. Processing at the IBM Cognos 8 report server is known as local processing, and it occurs after the data is retrieved from Cubing Services.

Because the data is no longer contained within Cubing Services or the underlying data warehouse there are some elements, such as indexes and materialization, that are not accessible during local processing. Without these

structures there is additional effort required to process large result sets when local processing is involved.

A good rule of thumb when working in Report Studio is to set the properties of the query to prevent local processing, which you do using the Query Explorer to select the query object, and then set the Processing property of the query to Database only, as in Figure 5-29. This action causes any local processing to return an error message during report execution or validation and serves as an early warning to report authors that the approach they are using might involve additional query processing outside of Cubing Services.

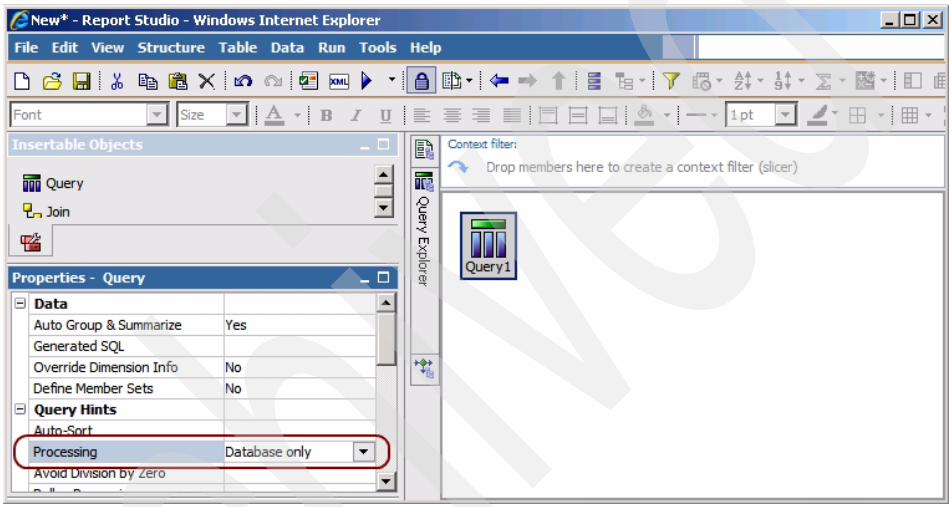


Figure 5-29 Database only processing

The most likely cause of local processing in a report is the use of functions that Cubing Services does not support, for example, many of the character and date manipulation functions that are available in IBM Cognos 8 do not have an equivalent within the multidimensional expression (MDX) language. These functions are identified within IBM Cognos 8 with a quality of service indicator, as shown in Figure 5-22 on page 130.

After local processing occurs it is no longer possible to return the data to Cubing Services for processing. If a locally processed function is used, this can prevent the application of dimensional functions on top of the function or expression that requires local processing.

When possible, it is best to use functions that are supported directly by Cubing Services so that local processing can be avoided.

5.5.4 Ad-hoc queries

When the user community has access to Query Studio and Analysis Studio there are many free-form queries issued to Cubing Services. Because the business questions that are asked might not be part of the original project scope there might be some query scenarios that do not match the indexing or materialization optimizations applied to your data warehouse. This type of situation can result in poor performance for some queries.

As part of the ongoing tuning and optimization of your Cubing Services environment, we recommend that you monitor the workload and use it to periodically evaluate the indexing and tuning settings applied to the data warehouse. For more information about tuning and the Optimization Advisor, refer to section 6.4.1, “Optimization Advisor” on page 192.

5.6 Common calculations

While the reporting in each business environment is targeted towards their specific business processes there are some common operations that arise regardless of the business. Invariably, there is some component of time involved and this shows up either as a point in time for producing results or in evaluating growth or metric comparisons between different time periods. After the data for the report is identified, there are usually requirements to view the aggregated or summarized results.

By applying appropriate techniques to these two areas, relative time and aggregation, a report author can ensure that the reports are easy to maintain and satisfy performance goals.

5.6.1 Syntax changes

Many of the functions that are available in Cognos 8 are familiar to anyone working with MDX and Cubing Services. However, there are differences in syntax that are worth mentioning.

The MDX language contains two syntax forms for functions and methods. Simple method calls are performed using a “dot” syntax where the method is appended to a member, set, or tuple using a “.” character, for example, to obtain the members directly beneath a given member in a hierarchy you use:

member.*children*

The ".children" is a method for retrieving the set of members beneath a single source member.

The other syntax form applies to functions, which are typically operations that require multiple input parameters to perform an action, for example, you can use the descendants function to obtain a set of members at an arbitrary depth beneath a single member or set of members:

descendants(member, 2 *)*

An experienced author of MDX has little trouble remembering which form each function or method requires, but for general usability a business focused report author would not be expected to remember this type of information. Fortunately, IBM Cognos 8 simplifies the expression syntax to a single common function syntax. This means that the "dot" functions become proper function calls within input arguments. The previous example with the children of a member becomes the following in IBM Cognos 8:

children(member *)*

5.6.2 Aggregates

Aggregate or member summary functions allow a measure value to be rolled up across a set of members. The most common member summary functions are:

- Total
- Minimum
- Maximum
- Average
- Count
- Aggregate
- Rank

The first five functions in the list function according to their name. The Aggregate function operates a little differently in that the method used to roll up the data depends on the properties of the data itself, for example, the Aggregate function can be applied on top of multiple different measures or measure calculations and the rollup of each measure is determined by the properties of the individual measures themselves.

The last one in the list of common member summaries is the Rank function, which assigns an integer ranking to each of the members in the set based on their relative measure value. The default is to assign the highest value as number one. The second highest is then number two and so on through the set of members. In the event of a tie, the numbers are assigned based on the order

of the members themselves rather than assigning the same rank value to both members.

The syntax of each of these functions is the same, and there are three main forms of each summary function:

- function(<Measure Expression> within set <Set> [, <Set>])
  - "within set" causes the summary to be computed based on the measure values obtained for the members in <Set>
- function(<Measure Expression> within detail <Set> [, <Set>])
  - "within detail" calculates the summary based on the detail values actually displayed in the report. This might differ from "within set" if there are lower levels of nesting that restrict the measure values to a subset of the values associated with the members in <Set>.
- function(<Measure Expression> within aggregate <Set> [, <Set>])
  - "within aggregate" aggregates the detail values in the report up to the level of the members in <Set> and then the summary function is applied to these aggregate values.

Figure 5-30 illustrates the differences in the different forms of aggregate functions by calculating an average of a Sales Amount measure using within set, within detail, and within aggregate.

		Sales Amount
Subdivision 1A	Region 44	872,423.17
	Region 45	551,355.32
	Subdivision 1A	**1,423,778.49**
Subdivision 1B	Region 22	520,386.42
	Region 24	774,956.15
	Subdivision 1B	**1,295,342.57**

		Sales Amount
Subdivision 1A	Region 45	*551,355.32*
Subdivision 1B	Region 22	*520,386.42*
	Region 24	*774,956.15*
Average Subdivisions - set		**1,359,560.53**
Average Subdivisions - detail		**615,565.96333333**
Average Subdivisions - aggregate		**599,513.3025**

Figure 5-30 Within set, detail, and aggregate

The source values for the entire data set are on the left side, and the modified data set and the average calculations are shown on the right side. The data on the right was modified to remove Region 44, which is nested below Subdivision 1A.

The three different average functions are:

- average(currentMeasure within set [Organization Unit Sub Division Level])
- average(currentMeasure within detail [Organization Unit Sub Division Level])
- average(currentMeasure within aggregate [Organization Unit Sub Division Level])

[Organization Unit Sub Division Level] is a reference to the outer set containing Subdivision 1A and Subdivision 1B.

The average “within set” uses the Sales Amount values for the subdivision set containing Subdivision 1A and Subdivision 1B. From the values on the left of Figure 5-30 on page 143, we can see that the values for these members are 1,423,778.49 and 1, 295342.57. This does not take into account the fact that at the nested Region level the Region 44 member was removed. The value for the specific member, Subdivision 1A, remains the same.

The average “within detail” operates a little bit differently. Instead of using the values for the subdivision members, the average makes use of the detail values included on the report. These values are 551,344.32, 520,386.42, and 774,956.15. The resulting average does not include the value for the excluded Region 44 member.

The average “within aggregate” also uses the detail values on the report, but the first thing that it does is to roll the lower-level detail values up to the subdivision level. Subdivision 1A uses the value 551,355.32 while Subdivision 1B uses 647,671.285, the average of the two detail region values. The final average is then calculated using the two aggregated subdivision values.

In most cases the “within set” operation is the most appropriate for calculating member summaries. This also allows the summary values to be sourced from Cubing Services directly rather than calculated by IBM Cognos 8 from the detail values retrieved for the query.

Another important property of member summary functions is the ability to supply multiple sets within the function arguments. When the sets from different dimensions are used they form a crossjoin and the member summary function operates over the entire cross product set of measure values. In Figure 5-31 on page 145, we can see two average calculations. The first average operates over the Region set only and the average is evaluated for each year based on the single set of Region members. The second average includes a reference to both sets so the average is computed across the cross product of the year and region measure values (all 12 values rather than 4 for each year).

		Sales Amount	Average - Region	Average - Year, Region
2002	Region 44	274,795.68	217,847.775	226,593.42166667
	Region 45	181,080.65	217,847.775	226,593.42166667
	Region 22	169,433.41	217,847.775	226,593.42166667
	Region 24	246,081.36	217,847.775	226,593.42166667
2003	Region 44	271,770.76	211,169.1225	226,593.42166667
	Region 45	165,975.4	211,169.1225	226,593.42166667
	Region 22	168,229.51	211,169.1225	226,593.42166667
	Region 24	238,700.82	211,169.1225	226,593.42166667
2004	Region 44	325,856.73	250,763.3675	226,593.42166667
	Region 45	204,299.27	250,763.3675	226,593.42166667
	Region 22	182,723.5	250,763.3675	226,593.42166667
	Region 24	290,173.97	250,763.3675	226,593.42166667

Figure 5-31 Multiple sets in an aggregate function

5.6.3 Relative time

Time information is the key to effective trend analysis. By using the appropriate functions and techniques you can create many different kinds of relative expressions and aggregates that allow you to compare your measure results across these different time periods.

Note: If you are creating the same expressions often, then it might be more effective to create calculated members within your cube model to accommodate the relative time period calculations, which ensures the accuracy of the expressions and makes the authoring experience easier by allowing simple drag and drop operations.

The main functions that are used for relative time calculations are:

- lastPeriods:

  LastPeriods returns a defined number of members either forwards or backwards from a target member in the same level of a hierarchy, which is useful for calculating rolling period ranges.

- parallelPeriod:

  ParallelPeriod finds the same relative member under a different branch of the time hierarchy. A common example is finding the same month in the prior year or two years ago.

- periodsToDate:

  PeriodsToDate works to return all the members up to and including a target member that are all contained within the same branch of a hierarchy. This is useful for calculating “to date” periods such as a year-to-date summary value.

You can combine these functions with other expressions but these three are the core to many of the time-based calculations that a report author needs on a regular basis.

Target Time Period

There are two primary ways of working with time expressions:

- A single time period basis:

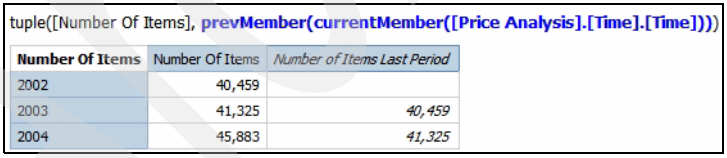
  A single time period is selected when the report is meant to display data for a user-selected period (5.4.4, “Prompting” on page 130) or the report is meant to show the latest figures available from the cube.

- A time range:

  A time range is used to show data trends over time as a sequence.

The two approaches are important because they dictate how the relative time functions are applied. When using a single reference period the expressions can simply make use of this member reference in lastPeriods, parallelPeriod, or periodsToDate. However, when using a time range, then there is no single reference member that can be used in all the other functions. In this case, the relative time functions must be evaluated for each member within the time period range.

Fortunately, there is a function that allows a you to identify each of the members in a set and use them individually within other functions. This function, currentMember, can be used to retrieve the member context from the query for additional relative time calculations. In Figure 5-32, the currentMember function used in a column calculation allows the expression to be evaluated for each of the years listed on the crosstab rows.


tuple([Number Of Items], prevMember(currentMember([Price Analysis].[Time].[Time])))

Number Of Items	Number Of Items	Number of Items Last Period
2002	40,459	
2003	41,325	40,459
2004	45,883	41,325

Figure 5-32 currentMember function

By evaluating the expression from the inside out we get the following sequence of operations:

1. Identify the current member of the Time hierarchy.

2. Find the member located immediately before the current member from the same level of the Time hierarchy.
3. With the previous member, lookup the value for the Number of Items measure.

So, if a single time period is used as the basis of the calculations in the report, then this single reference can be used in expressions. Otherwise, the currentMember function is required to evaluate the appropriate time period based on the query context.

Growth from prior period

One of the more common applications for relative time is to compare the current period results to a prior period, which you can accomplish in a few easy steps. In this example, a fixed time period reference is set by user selection. The calculations that are added to the query are aliased, as shown in Table 5-1.

Table 5-1 Calculation of Growth

Alias	Expression	Description
TargetPeriod	[Price Analysis].[Time].[Time].[Calendar Year-Quarter-Month Level]->?Select a Month?	Allow the user to select the target month.
PeriodLastYear	parallelPeriod([Price Analysis].[Time].[Time].[Calendar Year Level], 1, [TargetPeriod])	Find the same month from 1 year prior.
Growth	([TargetPeriod] - [PeriodLastYear])/[PeriodLastYear]	The change in measure value from the target month to the same month from the prior year.

Figure 5-33 shows the result of these calculations. Because the month members have the same caption value, additional text labels were added to the query to identify the different months, which can also be handled by including the year ancestor member as an identifier for the data.

Number Of Items	TargetPeriod	PeriodLastYear	Growth
	8	8	
Region 44	1,277	1,157	10%
Region 45	748	701	7%
Region 22	709	664	7%
Region 24	1,113	1,035	8%

Figure 5-33 Calculating growth with parallelPeriod

You can easily modify the parallelPeriod function to retrieve a month from two or three years prior by increasing the integer argument supplied to the function. Alternately, you can use the same function to retrieve the same relative month from a previous quarter instead of the year by adjusting the first level argument to use the quarter level in place of the year level.

Rolling 3 Months

Another typical time calculation is to compute a rolling range. Often this date range is summarized to calculate a total or average of a measure value. Figure 5-34 shows a simple application of the lastPeriods function to calculate a range of three quarters from a fixed target of Q1 of 2004. Typically, the result of the lastPeriods function is incorporated into a member summary function to create a single aggregate value for the set. This aggregate can be seen in the final row of the crosstab.

		Number Of Items
Q1 of 2004	1	20,188
lastPeriods(3, [Q1 of 2004])	**3**	**10,348**
	4	**2,759**
	1	**20,188**
aggregate([Number Of Items] within set lastPeriods(3, [Q1 of 2004]))	**Summary**	**33,295**

Figure 5-34 Simple use of lastPeriods

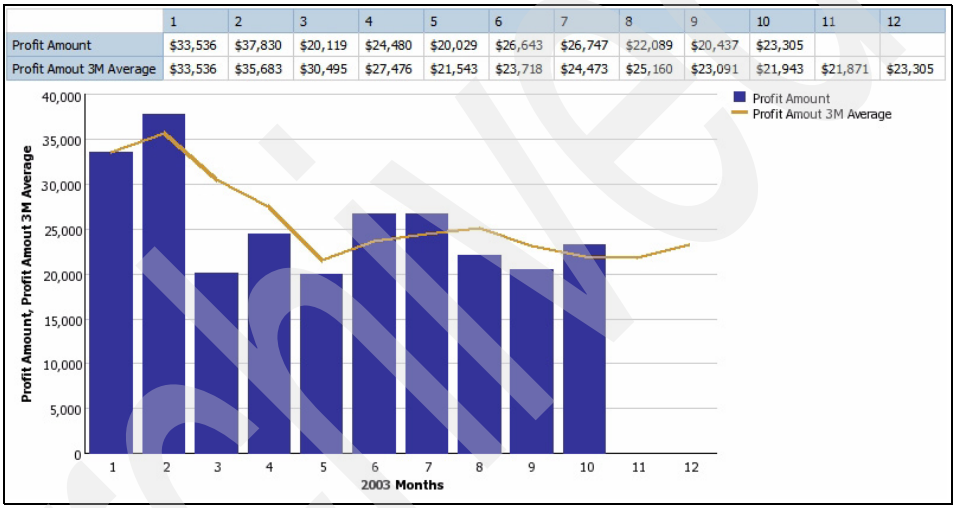
Note that the lastPeriods function rolls over the boundary of the year and retrieves quarters three and four from the previous year, 2003, to create the 3-month range.

In a more interesting application, the lastPeriods function can be used with a range of months to calculate the rolling three month average for each month. In this case, the calculations use the months of 2003 to plot the Sales Amount and the rolling three month average Sales Amount. The expressions used for the report are listed in Table 5-2, and the results are shown in Figure 5-35 on page 149.

Table 5-2 Calculating a Rolling 3 Month Average

Alias	Expression	Description
2003 Months	descendants([2003], 2)	Retrieves the members two levels below 2003. In the current cube model these are months.

Alias	Expression	Description
Rolling 3 Months	lastPeriods(3, currentMember([Price Analysis].[Time].[Time]))	Uses the current member of the time dimension to find the last three periods. In this scenario, the current member is from the Month level.
Profit Amount 3M Average	average([Profit Amount] within set [Rolling 3 Months])	Calculates the average of the Profit Amount measure over the last three months based on the current time context.



	1	2	3	4	5	6	7	8	9	10	11	12
Profit Amount	$33,536	$37,830	$20,119	$24,480	$20,029	$26,643	$26,747	$22,089	$20,437	$23,305		
Profit Amout 3M Average	$33,536	$35,683	$30,495	$27,476	$21,543	$23,718	$24,473	$25,160	$23,091	$21,943	$21,871	$23,305

Figure 5-35 lastPeriods to calculate a rolling 3 Month average

Year/Quarter/Month to date

One final relative time application is calculating a period to date rollup, which is an aggregate operating over a set of periods under a common ancestor up to and including a target or current period, for example, if the target period were November of 2004 then the Year-To-Date range would include the months from January of 2004 (the start of the year) up to and including November of 2004 (the target period). Similarly, if this were modified to calculate a Quarter-To-Date then the range would include October of 2004 (the start of quarter 4) and November of 2004.

Building this into a reporting scenario, we examine the Year-To-Date, YTD, sales amounts of several retail store regions and compare the current YTD values with the YTD calculated for the prior year. Table 5-3 on page 150 lists the expressions

to build this report, and Figure 5-36 shows the results. The list of expressions might seem more lengthy than prior examples but this is a result of calculating two YTD aggregates rather than due to any greater level of complexity in the individual expressions.

Table 5-3 Year to date sales amount comparison expressions

Alias	Expression	Description
TargetPeriod	[Price Analysis].[Time].[Time].[Calendar Year-Quarter-Month Level]->?Select a Month?	Allows the user to select the target month.
PeriodLastYear	parallelPeriod([Price Analysis].[Time].[Time].[Calendar Year Level], 1, [TargetPeriod])	Finds the same month from one year prior.
YTD range	periodsToDate([Price Analysis].[Time].[Time].[Calendar Year Level], [TargetPeriod])	The range of month members from the start of the current year to the target period.
Prior YTD range	periodsToDate([Price Analysis].[Time].[Time].[Calendar Year Level], [PeriodLastYear])	The range of month members from the start of the prior year to the matching target member from the prior year.
YTD Total	total(currentMeasure within set [YTD range])	A total of the measure values for the months of the YTD range.
Prior YTD Total	total(currentMeasure within set [Prior YTD range])	A total of the measure values for the months of the Prior YTD range.
YTD Growth	([YTD Total] - [Prior YTD Total]) / [Prior YTD Total]	The growth of the year to date measure total compared to the prior year.

Sales Amount	YTD Total	Prior YTD Total	YTD Growth
Region 44	$325,857	$271,771	20%
Region 45	$204,299	$165,975	23%
Region 22	$182,724	$168,230	9%
Region 24	$290,174	$238,701	22%

Figure 5-36 YTD and Growth calculations

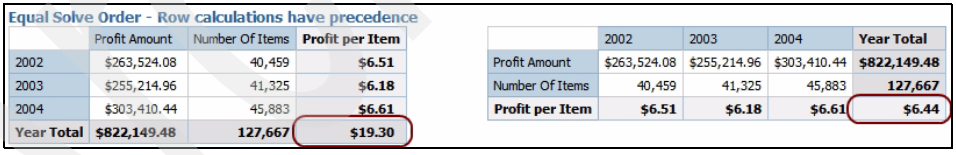
5.6.4 Solve order

When creating calculations in a query, there are sometimes certain operations that you must calculate in sequence. The way to control the order of calculations is by using the Solve Order property of a data item in Report Studio. A higher value for the solve order forces the expression to be evaluated later, after expressions with lower solve orders.

Regular calculations that are added to a query do not have a value assigned for their solve order, which means that they use a default solve order of zero (0). Aggregate functions that are created by using the Aggregate button on the Report Studio toolbar have a default solve order of one (1) so that they are evaluated after any of the other expressions in the query.

In the event that expressions on opposite edges of the query (such as rows and columns) have the same solve order, the rows are given precedence by default. However, this is configurable using the Fact Cells Precedence property of a crosstab object.

In Figure 5-37, two calculations were added to a query. The first calculation, Year Total, uses a member summary function to aggregate the values across the set of Years. The second calculation, Profit per Item, divides the Profit Amount measure by the Number of Items measure. Both calculations have a solve order of zero. By swapping the rows and columns it becomes apparent that there are differences in the solve order precedence. When the Profit per Item calculation is on the crosstab columns the result becomes incorrect and is displayed as the total of the Price per Item detail values rather than evaluating the division across the other measure totals.



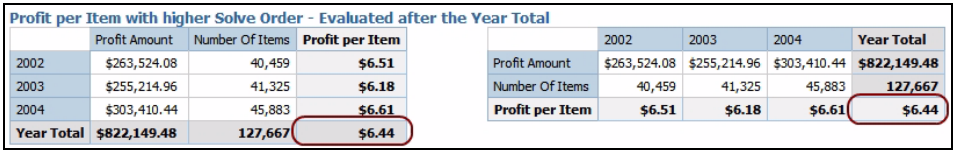
Equal Solve Order - Row calculations have precedence

	Profit Amount	Number Of Items	Profit per Item
2002	$263,524.08	40,459	$6.51
2003	$255,214.96	41,325	$6.18
2004	$303,410.44	45,883	$6.61
Year Total	$822,149.48	127,667	$19.30

	2002	2003	2004	Year Total
Profit Amount	$263,524.08	$255,214.96	$303,410.44	$822,149.48
Number Of Items	40,459	41,325	45,883	127,667
Profit per Item	$6.51	$6.18	$6.61	$6.44

Figure 5-37 Default Solve Order precedence

By adjusting the solve order property of the Profit per Item calculation it is possible to push the evaluation to after the Year Total is performed, which means that the appropriate value for the Profit per Item can be obtained based on the aggregated Profit Amount and Number of Items measures. The result of this change is illustrated in Figure 5-38 on page 152.



Profit per Item with higher Solve Order - Evaluated after the Year Total

	Profit Amount	Number Of Items	Profit per Item
2002	$263,524.08	40,459	**$6.51**
2003	$255,214.96	41,325	**$6.18**
2004	$303,410.44	45,883	**$6.61**
Year Total	**$822,149.48**	**127,667**	**$6.44**

	2002	2003	2004	Year Total
Profit Amount	$263,524.08	$255,214.96	$303,410.44	**$822,149.48**
Number Of Items	40,459	41,325	45,883	**127,667**
Profit per Item	**$6.51**	**$6.18**	**$6.61**	**$6.44**

Figure 5-38 Using Solve Order to control order of operations

5.7 Building financial reports

IBM Cognos 8 supports two authoring modes in Report Studio for satisfying authored reporting applications:

- Professional authoring

  The Professional authoring mode is the full set of Report Studio capabilities which are outlined in section 5.4, “Authored reporting” on page 124, and allow for a report author to generate advanced calculations and report functionality.

- Express authoring

  The Express authoring mode provides a streamlined set of capabilities catering to financial and management style statement reporting.

You can change the choice of authoring mode using the Authoring mode selection from the Report Studio View menu. You can bring content that you created in either authoring mode from one mode to the other.

Extended Data Item properties also allow the Express authoring mode to present data dynamically during report authoring and allow for drill operations on live data when creating the report layout. Figure 5-39 on page 153 shows the Express authoring mode of Report Studio using the Financial report template to display live data within the Report Studio window.

The Financial report template provides a common set of formatting options typical of financial and management statement style reports. This template is the default template for the Express authoring mode but is also available when creating new reports in Professional authoring mode.

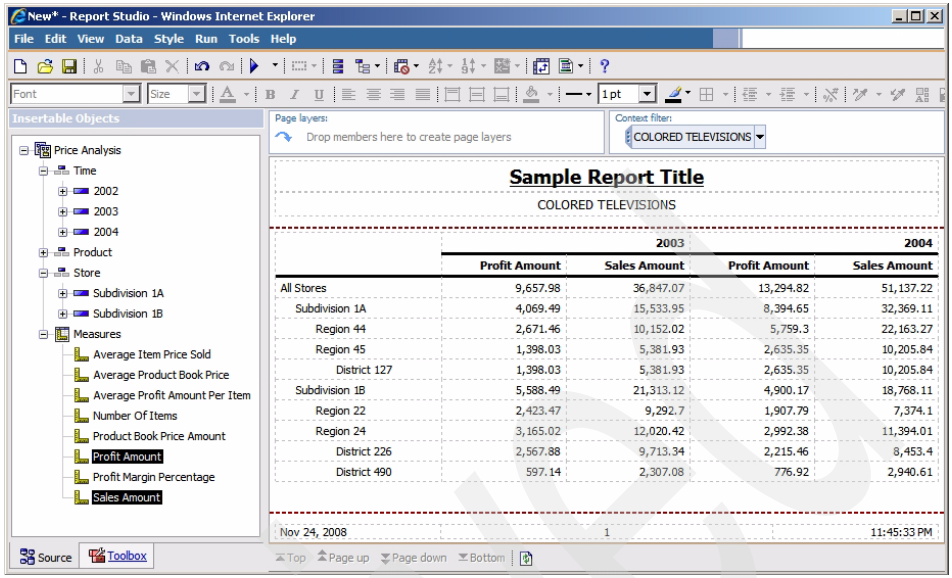


Figure 5-39 Express Authoring mode with live data

5.7.1 Managing sets and members

Several options exist for creating the rows and columns in Express authoring mode. Individual members can be added to the layout with or without their child members or multiple members can be grouped into sets. The method used when dragging items from the source model to the report layout is determined using the toolbar buttons shown in Figure 5-40.

Figure 5-40 Toolbar buttons for controlling member/set insertion options

Inserting sets allows multiple members of the same dimension to be grouped together, which makes formatting and nesting operations easier as one action can be used on the entire set rather than having to apply properties to each individual member.

In Figure 5-39, the years 2003 and 2004 are grouped as a set, which allows the two measures, Profit Amount and Sales Amount, to be nested under the entire set rather than having to nest the measures individually beneath the separate year members.

In the same report the root member of the Store dimension, All Stores, was added to the rows. Double-clicking on the All Stores member was then used to retrieve the child members, Subdivision 1A and Subdivision 1B, which were then also expanded to navigate deeper into the Store hierarchy.

These simple options allow a financial business user to quickly create statement style reports. The intuitive data manipulation actions guide the report creation and handle many of the underlying functions and expressions automatically while still providing the sophisticated layout and formatting options required for authored reporting.

5.7.2 Managing calculations and summary members

Express authoring mode allows calculations and summary members to be created similar to Query Studio and Analysis studio while also providing the full expression editor capabilities of the Professional authoring mode, as depicted in Figure 5-22 on page 130.

By selecting one or more members within the work area, the calculation button on the toolbar or the right-click context options for calculations become available. The options that are available depend on the number and type of data items selected. The appropriate expression is generated based on the user selections, and solve order is assigned to evaluate expressions in the following order:

- Addition or subtraction
- Multiplication or division
- Aggregation (rollup)
- Remaining arithmetic functions: absolute, round, round down, average, minimum, maximum, median, count
- Percentage, % difference (growth), or % of total
- Rank, quartile, quantile, or percentile

In the case of a tie in solve order for calculations on different edges, the precedence is granted to the crosstab row calculation.

In most cases, it is recommended that you use the aggregation or rollup calculation instead of the other available rollup functions, such as average or minimum. By explicitly defining the aggregation method, the default rollup defined within the cube model might be overridden, which can produce results different from those intended by the underlying data model.

However, often it is not necessary to use the Rollup calculation at all. When displaying a complete set of child members it is easier to use the common parent

member to display a summary value rather than creating a calculation to obtain the same results. A parent member can be added either as a single member by dragging it onto the layout next to the existing members or it can be added to a defined set by right-clicking the set and selecting the Edit members option. The resulting dialog, shown in Figure 5-41, allows a report author to include the higher-level members as part of the existing set.

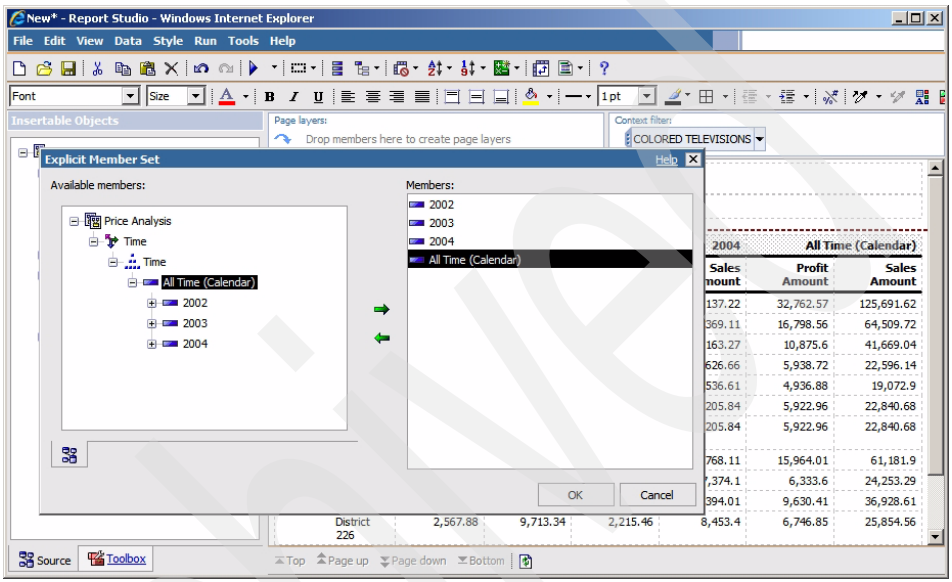


Figure 5-41 Adding a parent member as a summary member

Over large sets, using a single parent member for the summary can help improve performance by leveraging existing aggregates within the data warehouse rather than forcing a calculation across the full set of detail members.

5.8 Dashboarding

New in IBM Cognos 8.4 is an optional component named IBM Cognos 8 GO! Dashboard. This new component addresses another report consumption scenario that progresses from authored reporting.

Dashboards are another reporting type that provides an at-a-glance view of key business precesses and metrics. In the end effect, this is an authored reporting scenario, but the quick summary method of consuming this information has additional requirements for authors.

You can leverage Report Studio to create the summary chart or table content for dashboarding; likewise, portlet views within the Cognos Connection portal can be used to display and link information between multiple different report sources. IBM Cognos 8 GO! Dashboard extends these capabilities using a Flash interface to generate highly graphical data representations with data interactivity.

The initial release of IBM Cognos 8 GO! Dashboard allows components of authored IBM Cognos 8 content to be included in a dashboard framework. However, the architecture of this component is not distinctly tied to Cognos 8 content and future releases might see extensions to the available input sources for dashboarding, including third party input sources.

An entire report or pieces of a report can be used in dashboard viewers. By working with the report elements as data fragments the display can be customized to show the data in different formats, for example, Figure 5-42 on page 157 shows a sample dashboard with three panels. The Financial report template provides a common set of formatting options typical of financial and management statement style reports. This template is the default template for the Express authoring mode but is also available when creating new reports in Professional authoring mode.

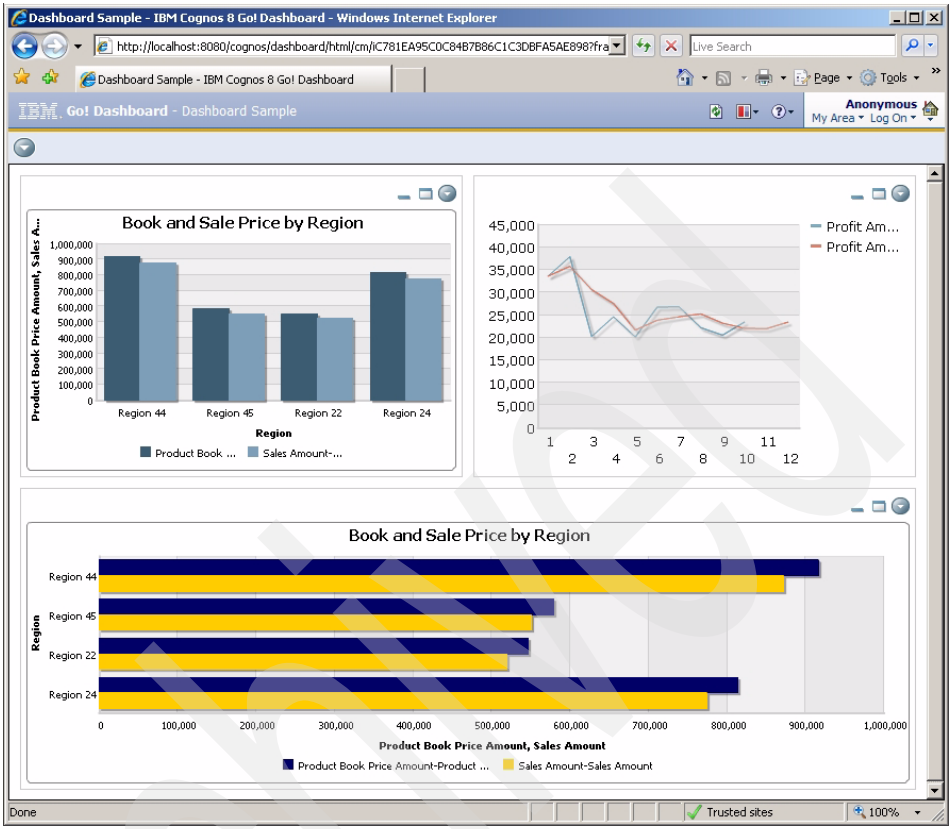


Figure 5-42 IBM Cognos 8.4 GO! Dashboard

The chart at the top left is sourced directly from the report shown earlier in Figure 5-19 on page 127, without including the list query from the same report. The chart shown at the bottom of the dashboard is the exact same chart data, but the display is customized within the dashboard viewer to show a different chart type and uses a different palette. Users of the dashboard can be allowed to make the same changes when viewing the final dashboard results to explore different data relationships.The third chart at the top-right of the dashboard displays content sourced from the crosstab shown in Figure 5-35 on page 149. Again, the underlying data can be displayed as crosstab data or transformed into a graphical format using a chart.

Like other reporting formats, if drilling up or down is permitted in the source report, then the same actions are possible within the dashboard. On top of these capabilities, the dashboard interface introduces user interface components such as sliders and check boxes that can interact with the portlets to restrict the retrieved data that is rendered in the flash viewers.

Additionally, GO! Dashboard offers many of the same portlets that are available within Cognos Connection, such as the Cognos Navigator and Cognos Search portlets, which can be tied to a viewer portlet that displays an existing report selected through either of the first two portlet types.

In all, IBM Cognos 8.4 GO! Dashboard provides an intuitive interface for an at-a-glance view of the top level data that drives the business. The flash rendering components provide attractive data representation and the flexibility to select different representations of the underlying data.

5.9 Reporting summary

IBM Cognos 8.4 Business Intelligence offers considerable flexibility in allowing authors to tailor data access and distribution to suit different consumers. The single point of data access through the common Cubing Services cube model ensures that all reporting works with a single version of the business truth. Likewise, the common Cognos Connection portal provides a single point of access for all consumers to author and consume reports.

By leveraging the ad-hoc and analytical query capabilities of Query Studio and Analysis Studio a user can satisfy their own unstructured business questions. This lightens the load from a dedicated reporting or authoring team and puts the data directly in the hands of the business users who both need the resulting data and understand the business processes in greater detail than any isolated reporting team can.

The sophisticated authoring capabilities of Report Studio build on the wide range of business intelligence capabilities by offering a structured reporting environment. Here a report author can customize report content for presentation, create advanced calculations, and set up reports to be delivered to broad audiences through bursting.

IBM Cognos 8.4 also provides several additional capabilities for reusing the investment in existing reports. IBM Cognos 8 GO! Dashboard allows report content to be reused in dynamic dashboarding applications that satisfy the at-a-glance data access requirements for quick monitoring of critical business metrics. Also, IBM Cognos 8 GO! Office gives users familiar with desktop office applications a means of integrating dynamic report content into such applications as Microsoft Word, Excel, and PowerPoint. Additional integration into Excel allows users to create report content in Excel and make use of the extensive Excel function libraries when analyzing their data.

In the end effect, the reporting possibilities and applications are wide open for users to create the content they need to drive their business forward. With direct

access to the data investment in Cubing Services there can be significant business improvements by lowering the time it takes to access, analyze, and make decisions based on the available corporate data.

6

Cubing services performance considerations

Performance is one of the most important things to be considered during the entire development process of a data warehousing environment. There are also many other factors to be considered as you build your Cubing Services environment. In this Redbooks publication, we described a number of them. As examples, there is the design of the relational and the multidimensional infrastructure, and the building of the Cubing Services model. But in fact, all of those elements are interrelated, that is, when designing and building an environment, all the decisions must be made keeping in mind their impact on performance.

Doing this enables you to design an environment that performs well. Otherwise, you might spend significant time and money building the environment only to find that the performance is not acceptable. Then comes more time and money, and user disappointment, as you try to improve the performance. So plan and design your systems, always keeping in mind the importance of the resulting performance.

In this chapter, we discuss and describe performance considerations in a multidimensional environment, and particularly as it relates to InfoSphere Warehouse Cubing Services.

6.1 Impact of the data model on performance

The data modeling phase is a very important part of the design process for a data warehouse, because it specifies how the data is structured and how it is accessed. It is the base building block of any data environment, and it is becoming increasingly important because of the dramatic increases in the volumes of data flowing into and out of businesses today.

The good news is that this increase in data volume provides significant information from which to gather business intelligence. But now businesses must step up to the challenges of gathering, organizing, storing and making available all that data in a format that is fast to access and easy to analyze by management and data analysts. Relative to the subject of this book, that means there is an increasing demand for a highly performing infrastructure to make all that possible, and all that starts with well-defined relational and cube models.

In other chapters of this book, we describe how to define and build such models. Now we present a number of considerations that have an impact on the performance of your data warehousing environment.

6.1.1 Referential integrity

As you build your data model, you define referential integrity, which you must do even if you only create the informational constraints but do not enforce them, which is because when using referential integrity, it provides extremely valuable hints to the DB2 optimizer for optimizing access plans. In addition, you can use it to determine accurate MQT routing during the query processing.

6.1.2 Normalization

Most everyone is familiar with data normalization. So, we do not discuss or describe how to normalize the relational database model. We simply want to remind you that you must apply the implementation rules of normalization (1st, 2nd, and 3rd normal form) to the development of a well-defined data model, because it is important to avoid redundancies and inconsistencies in your data.

Furthermore normalization maximizes the flexibility of the system for the future growth in your data structures and has a huge impact on the performance of your system. However, for some database models, also think about using denormalization, because it can be useful in reducing, for example, the number of joins that might be required in a query, and can minimize the complexity of a database by reducing the number of tables. Both of these considerations can add to your goal of better performance.

Note: Typically the relational database model of a Data Warehouse is not fully normalized.

6.1.3 Design of dimensions

There is no inherent limit of the numbers of dimensions that can be included with a cube. But, more dimensions means that more intermediate memory is required for tuples, the SQL generated by the Cube Server is longer and more complex, and the routing of the MQTs can become an issue. Furthermore, when you use static caching all members from each dimension are stored in the cache and it takes more time until they all are loaded there. So analyze how many dimensions are really needed for the reports, for example. When you use dynamic caching the number of members can be significantly higher than it is with static caching, but the resulting reports and the startup time of the cube takes more time.

A good recommendation is to avoid large, flat dimensions if possible, for example, when your dimension members have thousands of children they also require more resources. If this becomes an issue for you, explore the creation of artificial levels in the hierarchy.

Tip: Dimension table foreign keys must be integer columns.

6.1.4 Levels

When specifying level keys, keep single column level keys where possible or at the least use as few columns as absolutely necessary. Furthermore, avoid redundant information in the level keys. As examples:

- Instead of using a level key that is compromised of four columns, such as country, state, county and zip (for example: USA, CA, Santa Clara County and San Jose), use a level key that is comprised of a single level key, such as city_id of San Jose.
- In the Time dimension, the Month level key does not need to contain the Quarter level. The month level key can just be comprised of 2 columns, such as year and month (for example: 1997, January).

Furthermore, it is preferable to use integer values for level keys, because due to the SQL that Cubing Services generates the level keys are used for the join and the group-by operations. So if the level keys are just one integer column then the joins, comparisons and group-by operations using an integer are faster than joins, comparisons and group-by operations on multiple columns and other types of columns.

6.1.5 Numbers of cubes

It is possible to include more than one cube in a Cube Server, but be careful. This is because every cube needs its own resources and each Cube Server runs in a single process. So all cubes within a server compete for resources in that process. Theoretically the limit of the number of cubes is the maximal integer value supported by the particular Java™ Virtual Machine.

However, it is your choice whether or not you want to have a large cube or several smaller cubes. When using a single cube the queries posed against the cube can share a single instance of metadata, but it might not be practical to cache the single large instance. So in general, use a few smaller cubes because you can better manage the cached data, even if some of the metadata is duplicated.

When more than one cube exists also consider distributing the cubes across multiple Cube Servers that are located on different servers. That provides you with the opportunity to assign the hardware resource to each cube.

6.1.6 Measures

In general all relevant measures must be modeled in a cube model, but the more measures that are used for your cube, the more disk space that is required. So for each cube, only take the measures that are needed for the special subject area for which the cube was created. In addition, your MQTs are affected when using a high number of measures because more resources are needed for the aggregation.

So, try as often as possible to use distributive measures (as examples, SUM and COUNT) because they can be aggregated from one level to the next. Using non-distributive measures (as examples, AVG and STDDEV) can affect the MQTs because the data must be generated from base data and not from an underlying level. If you use non-distributive measures the Optimization Advisor and the DB2 Advisor is limited with their recommendations for new performance improvements, such as the possibility to create *refresh immediate* MQTs. Furthermore MQT recommendations become more complex in the presence of non-distributive measures and they often result in larger MQTs with less overall coverage of the cube model, which means routing is more difficult and resulting queries against these non-distributive measures are generally longer running than their distributive counterparts, even when the query gets routed to an MQT.

6.1.7 Small fact tables

A fact table can be used for one or more cube models, which in turn can be used to create one or more cubes. This does not mean that we must have a single, big fact table, with all imaginable dimensions (keys) at hand, from which to allow many cube models to be created.

Suppose, for example, we have a fact table with six dimensions (and some columns for measures) and millions of rows, and that we must create a new cube with five dimensions that are already in that fact table, plus a new one that has only 12 distinct members. The number of rows that a fact table has depends on how many different primary key value combinations actually exist. Adding this new key column to that fact table can potentially increase the row count by 12, should every new possible combination exist for each of the original keys!

Because of that huge increase in fact table size, any query sent to it is probably much slower than before. Also the MQTs created for the original cube models would take much longer to be refreshed because the RDBMS now has to perform a GROUP BY clause for all 6 original keys to retrieve the same numbers from the new, expanded fact table.

It might be easier and faster to create a new, independent fact table and end up having two 6 dimensions fact tables rather than having a single fact table with seven dimensions, and they most likely have a combined size much smaller than that single seven dimensions fact table. The message here is that you must try to use as few dimensions as possible in fact tables. However, adding more columns for measures does not have a big impact.

6.2 Where to tune performance

After the implementation of the relational model in the relational database and the implementation of the cube model in the Cube Server, the next consideration to understand is how the tuning process functions in the relational database and in the Cube Server.

6.2.1 Relational database tuning

For sure there are already many documents that deal with the optimization of the IBM DB2 relational database. So in this section, we summarize the primary factors that you must take into account when you want to optimize DB2 for processing Cubing Services SQL queries. Most of them must be adjusted to every individual environment, so here we give you some advice on optimizing your relational database and show you how to do that.

DB2 configuration properties

In the configuration of DB2 there are many properties that you can adjust to optimize performance. In DB2, there are three levels on which you can change the settings, the DB2 registry and environment variables, the Database Management settings, and the Database settings:

- DB2 registry and environment variables:
  - To view your current settings:

    ```
    db2set
    ```

  - To change the current settings:

    ```
    db2set registry_variable_name=new_value
    ```

Tip: More information about DB2 registry and environment variables is located at the following Web site:

http://publib.boulder.ibm.com/infocenter/db2luw/v9r5/index.jsp?topic=/com.ibm.db2.luw.admin.regvars.doc/doc/c0007340.html

Performance relevant registry and environment variables are at the following Web site:

http://publib.boulder.ibm.com/infocenter/db2luw/v9r5/index.jsp?topic=/com.ibm.db2.luw.admin.regvars.doc/doc/r0005665.html

- DB2 Database Manager configuration settings:
  - To view the current settings:

    ```
    db2 list dbm cfg
    ```

  - To change the current settings:

    ```
    db2 update dbm cfg using <config-keyword> <value>
    ```

  - Table 6-1 on page 167 lists some important performance configuration parameters

Table 6-1 Database Manager configuration parameter

Parameter	Description
aslheapsz	The application support layer heap represents a communication buffer between the local application and its associated agent. This buffer is allocated as shared memory by each database manager agent that is started.
sheapthres	This parameter is an instance-wide soft limit on the total amount of memory that can be consumed by private sorts at any given time. When the total private sort memory consumption for an instance reaches this limit, the memory allocated for additional incoming private sort requests is considerably reduced. The SQL statements that are submitted from Cubing Services can contain ORDER BY clauses and so this parameter can be become necessary.

- DB2 Database configuration xettings:
  - To view your current settings:

    ```
    db2 list db cfg for <db-name>
    ```

  - To change the current settings:

    ```
    db2 update db cfg for <db-name> using <config-keyword> <value>
    ```

  - Some important performance configuration parameters are listed in Table 6-2.

Table 6-2 Database parameters

Parameter	Description
catalogcache_sz	This parameter specifies the maximum space in pages that the catalog cache can use from the database heap.
dbheap	This parameter determines the maximum memory used by the database heap.
logbufsz	This parameter allows you to specify the amount of the database heap (defined by the dbheap parameter) to use as a buffer for log records before writing these records to disk.
maxappls	This parameter specifies the maximum number of concurrent applications that can be connected (both local and remote) to a database. Because each application that attaches to a database causes some private memory to be allocated, allowing a larger number of concurrent applications potentially uses more memory.
maxlocks	This parameter defines a percentage of the lock list held by an application that must be filled before the database manager performs lock escalation.

Parameter	Description
mincommit	This parameter allows you to delay the writing of log records to disk until a minimum number of commits are performed, which helps to reduce the database manager overhead associated with writing log records.
num_poolagents	This parameter sets the maximum size of the idle agent pool.
sortheap	This parameter defines the maximum number of private memory pages to be used for private sorts, or the maximum number of shared memory pages to be used for shared sorts.
sheapthres_shr	This parameter represents a soft limit on the total amount of database shared memory that can be used by sort memory consumers at any one time.
stmtheap	This parameter specifies the size of the statement heap, which is used as a work space for the SQL compiler during compilation of an SQL statement. Cubing Services can submit long SQL statements. Because those statements are processed the stmtheap parameter must be increased.

Running statistics

It is important to keep the statistics up-to-date for your tables and indices, especially when there are changes in the volume of the data, or even if the content of the data itself changes. It is better when periodic updates of the statistics are made, so that the DB2 Optimizer can choose the best access plan for your query.

There are also other possibilities for updating your statistics:

- The easiest way is to connect to the database and execute the following command for every table to be updated:

```
RUNSTATS ON TABLE <schema-name>.<table-name> ON ALL COLUMNS WITH
DISTRIBUTION ON ALL COLUMNS AND INDEXES ALL ALLOW WRITE ACCESS;
```

Tip: More options for the **runstats** command are at following Web site:

http://publib.boulder.ibm.com/infocenter/db2luw/v9r5/index.jsp?topic=/com.ibm.db2.luw.admin.cmd.doc/doc/r0001980.html

- If you are using the SQL Warehousing Tool (SQW), which is a component of the InfoSphere Warehouse, there is an operator included for executing runstats at the control flow level. So every time this control flow is executed, or

a modification to data is made, your statistics are updated, as shown in Figure 6-1.

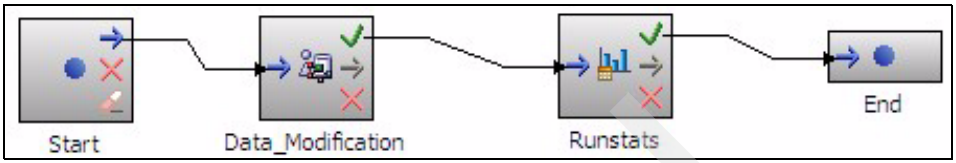


Figure 6-1 Runstats with SQW

Tip: More information about the InfoSphere SQL Warehousing Tool is at the following Web site:

http://www-01.ibm.com/software/data/infosphere/warehouse/sql.html

- Automatic Maintenance: Updating the statistics can also be performed automatically with the DB2 Automatic Maintenance Facility. This can be enabled within the Control Center. Right-click a database, choose the **Configure Automatic Maintenance** option, and follow the directions of the Wizard.

Reorganization

Periodically reorganize your data to be sure that the data is well-structured for an optimal data access. Furthermore, it frees up disk space. You can reorganize your data in a number of ways:

- On the command line, as examples, execute the listed commands for the following types of reorgs:
  - Reorg for a table:

    ```
    db2 reorg table <table-name>
    ```

  - Reorg for indices of a table:

    ```
    db2 reorg indexes all for table <table-name>
    ```

  - Reorg for a table in a partitioning group:

    ```
    db2 reorg table employee index empid on dbpartitionnum (1,3,4)
    ```

Tip: For more information about the **reorg** command visit the following Web site:

http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp?topic=/com.ibm.db2.udb.admin.doc/doc/r0001966.htm

If you are not sure whether or not to reorganize your data, you can use the **reorgchk** command to make that determination. This command is described at the following location:

http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp?topic=/com.ibm.db2.udb.admin.doc/doc/r0001971.htm

- As with the **runstats** command, you can also execute the reorg command with the SQL Warehousing Tool. As shown in Figure 6-2, there are a number of possibilities for reorganizing your data.

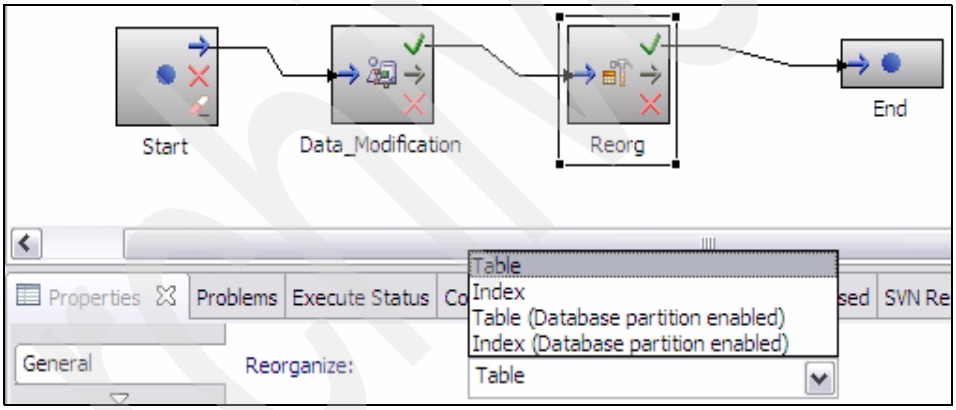


Figure 6-2 Reorg with SQW

- Automatic Maintenance: The reorganization of the data can also be done automatically with the DB2 Automatic Maintenance Facility. This can be enabled within the Control Center. Just right-click a database, choose **Configure Automatic Maintenance** option, and follow the directions of the Wizard.

Indexes

The use of indexes can significantly improve the performance of your queries. They are used by a database for the following purposes:

- Apply predicates to provide rapid look up of the location of data in a database, which can reduce the number of rows navigated
- To avoid sorts for ORDER BY and GROUP BY clauses
- To induce order for joins
- To provide index-only access, which avoids the cost of accessing data pages
- As the only way to enforce uniqueness in a relational database

However, determine if the DB2 Optimizer uses the index that you created, and check to determine if you do in fact better optimize the performance because indices need additional hardware resources. As examples, they:

- Add extra CPU and I/O cost to UPDATE, INSERT, DELETE, and LOAD operations
- Add to prepare time because they provide more choices for the optimizer
- Can use a significant amount of disk storage

An example command to create an index for one column of a table is:

CREATE INDEX <UNIQUE_NAME> ON <TABLE-NAME>
(<COLUMN-NAME>)

The runstats command: Do not forget to execute the **`runstats`** command after creating an index.

If you are not sure which index is most helpful for optimization, you can use the DB2 Optimization Advisor to gain advice for creating indices. How to use Design Advisor is be described in section 6.4, “Tools for tuning” on page 191.

Tablespaces and bufferpools

Defining the correct bufferpools and tablespaces is different for every environment, but it is important for the performance of DB2. Normally a Database Administrator must examine the environment and determine whether or not the definitions of the tablespaces and bufferpools are optimal for the requests that are posed against the database.

Logging performance

The DB2 logs can have a big impact on the performance of the database. As examples:

- It is possible to disable autocommit for the execution of queries. If you do this, every commit causes a synchronous log write to occur. The exception to this recommendation is if the application requires a commit after every SQL statement.

  On the DB2 command line it is possible to disable the autocommit function by using the **+c** option before executing the query, as demonstrated by the following command example:

  ```
  db2 +c <query>
  ```

  When doing this, you can commit or rollback the queries at the end.

  To disable the autocommit globally, you can set a DB2 Registry and Environment Variable by using, for example, the following command:

  ```
  db2 set db2options=+c
  ```

- Large database objects, such as clobs and blobs, must not be logged.
- DB2 logs must reside on dedicated disk storage, where no other I/O work occurs. In particular, separate the log files from the database itself. To change the location of the log file you can use the following command:

  ```
  db2 update db cfg for <database_name> using newlogpath
  <fully_qualified_path>
  ```

  Furthermore the used disk storage must have a fast write cache for the logs.
- The log buffer size must be set to 256 pages or larger.

Query optimization level

The DB2 query optimization level can be useful to adjust the amount of work and resources that DB2 puts into optimizing the access plan. The value for the optimization is between 0 and 9, for example, with level 9 all available statistics are considered to optimize the access plan. However, this takes a lot of time. To set the query optimization level for dynamic SQL, you can use the Control Center or the following command:

```
db2 set current query optimization = <0..9>
```

This level can be set for every database. When the optimization level is not set, the default value of 5 is used. For static SQL, the optimization that was specified during the preparation and bind phase is used.

Tip: Set the optimization level for the needs of the application. Use high levels only when there are complicated queries.

6.2.2 Cube server tuning

In the book *InfoSphere Warehouse: Cubing Services and Client Access Interface*, SG24-7582, you can find more information about gaining better Cube Server Performance.

In this section, we describe how you can set the various performance properties of the cube. In addition, we show a continuous sizing example for the primary cubing services components. The components of the cubing engine for InfoSphere Cubing Services, are depicted in Figure 6-3.

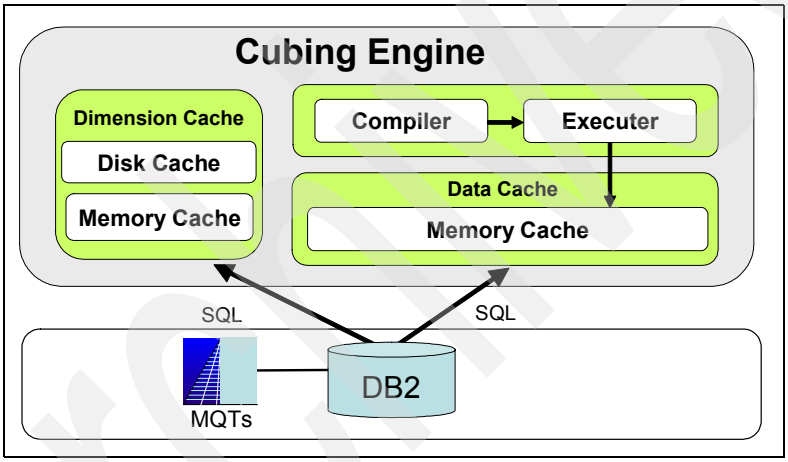


Figure 6-3 Cubing Engine

Member cache

The Member Cache, also called *Dimension Cache* or *Metadata Cache,* stores dimension metadata (members) and can be tuned to either completely or partially cache members. For member caching, there are two modes available: static caching (default) and dynamic caching.

When static caching is used, dimension members are read from the relational data source and preloaded into memory during cube start-up.

When dynamic caching is selected, dimension members are read from the relational data source and stored in compressed data files in a user-specified location on the system. Disk space usage is proportional to the number of

members in the cube. The amount of memory used for the member cache is proportional to the number of members and the size of each member.

When a cube is started or refreshed, the cube member cache files are generated and any previously existing files are overwritten. If the cube is running, dimension members are dynamically read into memory from the dimension files on an as-needed basis. If the member cache grows larger than the user-specified parameters, the space is managed according to cube cache settings.

Member Cache Properties

On the InfoSphere Warehouse Console, perform the following actions to set up the Member Cache:

1. Log into the InfoSphere Console, and select **InfoSphere Warehouse** → **Cubing Services** → **Manage Cube Servers.**
2. Select your Cube Server, click the related Hyperlink, and open the Cube tab.
3. Select the cube you want to change, and click its linked name.
4. Open the Cache tab, and a window is displayed, as shown in Figure 6-4 on page 175.
5. Now you have the following setting options:
   a. The default setting is that the Static Member Cache is enabled.
   b. To enable the Dynamic Cache, and implicitly disable the Static Cache, you must check the option **Enable dynamic caching**.
   c. For the Dynamic Cache you have the following options:
      - Do not limit the number of cached members
      - Limit number of cache members and specify amount of members

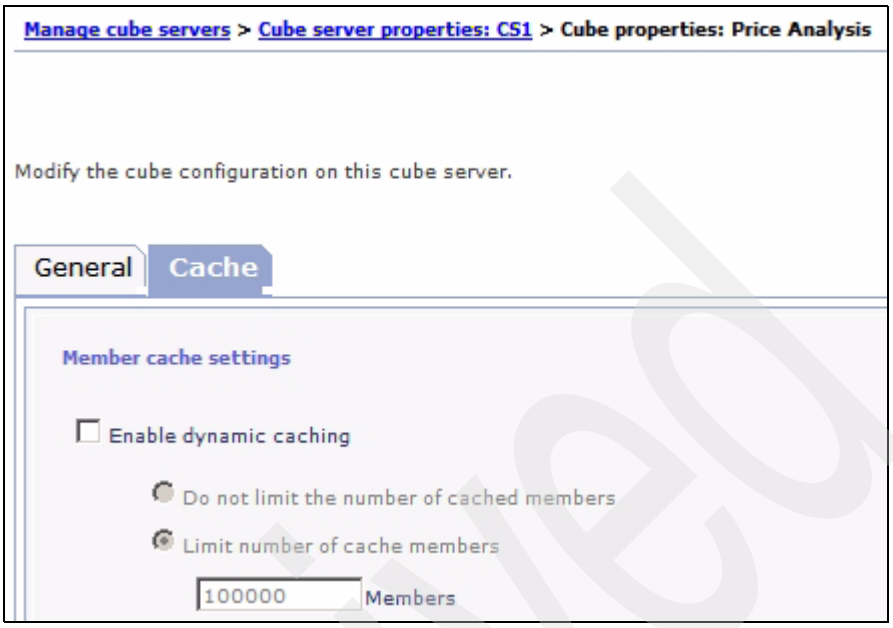


Figure 6-4 Settings for the Member Cache

Tip: As often as you can, use Static Caching so that the members of your dimension are kept in cache. Otherwise, the data needs to be read by the file system, which takes more time, particularly for huge dimensions. However, if you do use Dynamic Caching, try to minimize the dimensions members. Suggestions for how to do that are described in the section 6.2.3, “Tuning scenarios” on page 184.

Member Cache sizing example

In this example, we demonstrate how to calculate the Member Cache cost.

- You can classify costs into fixed and variable costs per member:
  - The fixed cost of a member consists of the fixed cost of that member plus the fixed cost of the indexes for that member. Calculate the costs as depicted in the following examples:

Fixed cost per member = 270 bytes per member for a 64-bit JVM

Fixed cost per member = 160 bytes per member for a 32-bit JVM

Fixed cost of indexes per member = 240 bytes per member for a 64-bit JVM

Fixed cost of indexes per member = 120 bytes per member for a 32-bit JVM

- Variable costs are of the following five types:
  - Member name: The cost of the member name is twice the length of the member name in bytes. In cases where a dimension level corresponds to a single column in a table, you can use a simple SQL query to estimate the average member name length for all members at that level. The following is an example of such an SQL query:

    ```
    SELECT AVG(LENGTH(colname)) from tablename
    ```

  - Children hash table size. A hash table is created for each member if the number of children is greater than 256.
  - Member property: The cost per member property is approximately 8 bytes of fixed overhead plus a variable cost that depends on the data type of the property. The primitive type of the property is typically converted into a Java object of the corresponding type, for example, if a property in the underlying relational table is of type INTEGER, it is converted into an Integer Java object (32 bytes); if a property is of type DOUBLE, it is converted into a Double Java object (32 bytes). Properties can also be strings (such as CHAR or VARCHAR), in which case they are converted to the corresponding String Java object. The number of bytes in a String object is 48 bytes of fixed overhead plus twice the length of the string.

    An overview for the Member Property and Level Key Cost by data type is shown in Table 6-3.

Table 6-3 Costs by Data type

RDB Type	Java Type	Size
Integer	Integer(Object)	16bytes
Double	Double(Object)	16bytes
Char, Varchar	String(Object)	38bytes + 2(length of string)

  - Level key: The level key is an array of values that can be comprised of 3 columns, such as country, state and store. Similar to the cost of a

member property, the cost of a level key consists of 8 bytes per key element of fixed overhead plus a variable cost that depends on the data type of the key element.

- Java overhead: Java overhead is incurred because of the way that Java grows its heap and deferred garbage collection. Based on testing we performed, the overhead was about 5%.

– Given these fixed and variable costs, you can estimate the size of a member and index structures as follows:

Size of a member = Fixed cost per member + 2 * name length
+ SUM(8 + sizeof(property))
+ SUM(8 + sizeof(level key))) bytes

Size of indexes per member = 240 bytes

► Static caching cost

– You can approximate the base cost of the members in the metadata cache under static caching as follows (on 64-bit JVMs):

Base cost = (size of a member + size of indexes per member) * n
+ sizeof(children hash table)

(where n is the number of members)

– The rest of the metadata (dimensions and levels) uses a few extra MB. The final cost must also take into account the Java overhead, but that cost can be difficult to determine. However, based on data from our testing, you might add another 5% memory to account for the cost of non-member metadata plus the Java overhead. Thus, you can approximate the final cost as follows:

Final cost = base cost * 1.05

► The following example shows how to calculate the cost under static caching. Assume that a cube is started on a 64-bit system and that the cube has the following characteristics:

– It has 10 dimensions totaling approximately one million members.

– The average member name length is 20 characters.

– One percent of members have more than 256 children.

– There are four integer properties per member.

– The length of the level key array varies depending on the depth of the level, but there is an average of two elements in the key. The key for this example is comprised of two elements, an integer ID and a name with string data type. Note that because name refers to an existing member name, you do not need to add the size of the name again here. You must add only the 8-byte cost of reference.

Given these characteristics, you can calculate the member and index sizes and the base cost as follows:

Size of a member = 270 + 2 * 20 + (8 + 32) * 4 + (8 + 32 + 8) = 518 bytes

Size of indexes per member = 240 bytes

Base cost = (518 + 240) *1 000 000 + (10 000 * 4096)

= approximately 799 MB

You can calculate the approximate final cost as follows:

Final Cost = 799 MB x 1.05 = 839 MB

Tip: In cases where detailed information about members, such as name lengths, numbers of properties, or numbers of member keys is unknown, use the following 64-bit coarse estimate: 1000 bytes per member.

Data cache

The data cache stores cube cells fetched from the relational database. After it is loaded into the data cache, the stored data is shared among concurrent and subsequent queries when available. The loaded data is kept in the Data Cache until the cube is restarted or flushed. The size of the data cache is configurable.

Data cache properties

For setting up the properties for the Data Cache, navigate to the same position that you did for the Member Cache settings (**InfoSphere Warehouse** → **Cubing Services** → **Manage Cube Server** → **Cube Server Name** → **Cube Tab** → **Cube Name** → **Cache Tab**), where you have the following properties, as shown in Figure 6-5 on page 179:

- The Default setting for the Data Cache is that all cells are cached: **Cache all cells**
- To limit the number of the cells that are cached, select the option: **Cache a limited number of cells**, and specify the number of the cached cells.

Figure 6-5 Settings for the Data Cache

In the context of the Data Cache properties, a cell is an address in the cube, which includes all measures associated with that address. So for example, if you have Time, Product, Store, and Measures dimensions in your cube a cell in that cube would be identified as ([1997], [Cookies], [Target]) and the cell includes all the measures. In this example, [1997] represents a member at the Year level in the Time dimension, [Cookies] represents a member at the Product Category level in the Product Dimension, and [Target] represents a member in the Store dimension.

Traditionally when you think of a cell as similar to the above, you only think of a single measure. So ([1997], [Cookies], [Target], [Sales]) is one cell, and ([1997], [Cookies], [Target], [Cost]) would be another cell (where [Sales] and [Cost] are both members in the Measures dimension). While in the usage of the Data Cache properties, the cell really corresponds to the idea of row in the calculation and that includes all measures for that intersection of members from other dimensions.

The Cache all cells option: Be careful when using the Cache all cells option because it can happen that you run out of memory when there are high volumes of data to be stored in the cache. That is especially important in a 32-bit environment, where a limitation of the cache size exists.

Data cache sizing example

In this example, we demonstrate how to calculate the Data Cache cost.

- There are three components of the memory usage cost of a row/cellset:
  - Each row/cellset is identified by a set of n integer values that correspond to dimension member IDs in the n non-measure dimensions.
  - Each row/cellset consists of a set of m measures represented by m values whose data types correspond to the data types of the measures. Note that the primitive type of a measure is typically converted into an object of the corresponding type, for example, if a measure in the fact table is of type INTEGER, it is converted to an Integer object (32 bytes); if a measure is of type DOUBLE, it is converted to a Double object (32 bytes). Measures can also be strings (such as CHAR or VARCHAR in the relational tables), in which case they are converted to the corresponding String object. The number of bytes in a String object is 48 bytes of fixed overhead plus twice the length of the string.
  - On 64-bit JVMs, there is an additional fixed cost that is associated with a group of related cellsets. A group of related cellsets consists of cellsets of the same granularity. This cost is approximately 20 additional bytes per cellset.

    Thus, you can calculate the approximate number of bytes used per cell as follows (on 64-bit JVMs), where n is the number of non-measure dimensions and m represents each measure:

    n * 5 + SUM(sizeof(m)) + 20

- Assume that there are 10 dimensions and five measures for a cube, of which three have type DOUBLE and two have type INTEGER. Based on those values, you can calculate the number of bytes that each row/cellset uses on 64-bit JVMs as follows:

  (10 * 5) + (3 * 32) + (2 * 32) + 20 = 230 bytes

- For the starting size of the data cache, 10% of the total size of the cube member cache data is reasonable, for example, if your cube member cache data size is 345 MB (using the calculation described earlier for member cache sizing), do the following calculations for the data cache:

  Estimated data cache size = 10% of 345 MB = 34.5 MB

  Data cache setting = estimated data cache size in bytes/ bytes per cell

- Therefore, for the previous example, where there are 10 dimensions and five measures, you can calculate the data cache setting as follows:

  34.5 MB / 230 bytes = approximately 150,000 cells

  Validate this value as you run your workload by examining the hit rates that you get in the data cache. Note that hit rates in the data cache depend on factors such as the size of the data cache, the size of a cellset, the amount of sharing among queries (interquery locality), and the number of times that the same data is accessed within a query (intraquery locality). The hits and misses information is stored in the log file. To get per-query data cache statistics, set the logging level to VERBOSE.

Tip: As a reasonable setting for the data cache size, calculate the data cache size setting after calculating the total size of the cube metadata (members). Then, start with a value of 10% of the metadata size, and further refine that setting based on the actual query performance and hit rates that you see in the log file.

Intermediate work area

The intermediate work area is needed for processing queries; thus, its size depends on the workload, for example, a large crossjoin (1000 members x 1000 members) requires a large intermediate work area because the crossjoined set of one million members is maintained in memory. Also, a query that has hierarchical ordering uses more intermediate memory than a query that has non-hierarchical ordering. Note that if such queries are submitted by multiple users simultaneously, the Cube Server reaches the limit of its resources quickly.

Remember that the default maximum Java heap size is usually quite small. Depending on the workload, you might need to increase the heap size.

DB2 connection pooling feature for Cubing Services

Since InfoSphere Warehouse Version 9.5.1, the DB2 Connection Pooling feature on the Cubing Services server has been available. With it you can reduce the Java Virtual Machine (JVM) memory consumption and improve MDX query performance.

Without the DB2 Connection Pooling feature enabled, the Cubing Services server establishes a JDBC™ connection for every SQL statement that is submitted to the DB2 back-end. The JDBC connection is dropped after the result is returned. With the DB2 Connection Pooling enabled, JDBC connections are never dropped, but are instead returned to the connection pool. When there is an SQL statement to be submitted to the DB2 back-end database, the Cubing Services server uses a free connection from the pool if there is one available.

The server creates a new connection only if no free connection is available in the connection pool.

A side effect of the connection pooling is that the Cubing Services server maintains a number of open connections to the DB2 back-end database at all times. The number of JDBC connections allocated by the Cubing Services server would gradually grow depending on the load and the number of concurrent data cache misses. This number can potentially increase to the maximum number of concurrently processed MDX queries.

Maximum number of concurrently processed MDX queries: The maximum number of concurrently processed MDX queries really means the number of query threads running in the server. So this defines the maximum number of queries that are actively processed at any point-in-time. Additional queries can be submitted to the server, but they wait in a queue for processing until a running query is completed and its thread becomes available.

To set up the maximum number of concurrently processed MDX queries:

1. Log into the InfoSphere Console by selecting **InfoSphere Warehouse → Cubing Services → Manage Cube Servers.**
2. Click the Cube Server that you want to adjust.
3. Under performance, choose the number of the maximum concurrently processed MDX queries. The default value is 100, as shown in Figure 6-6 on page 183.

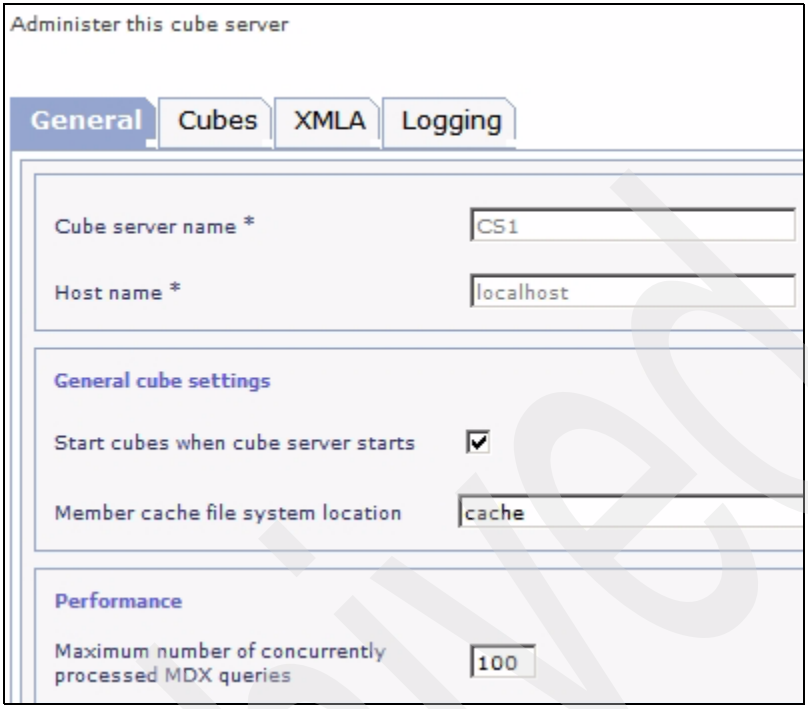


Figure 6-6 Connection Pooling for Cubing Service

The connection pooling feature is enabled for Cubing Services 9.5.2 and 9.5.1 refresh release. When you are using the original Cubing Services 9.5.1 release you must activate it by setting the csdb2maxconpool parameter. To enable the DB2 Connection Pooling feature, you must manually edit the Cubing Services server start-up script on UNIX and Linux® platforms, or a batch file on Windows platforms, and add the csdb2maxconpool property to the JAVA_OPTS environment variable as follows:

- UNIX and Linux platforms: $CUBINGSERVICES_HOME/ cubeserver.sh
  - Old content:

    JAVA_OPTS="-Duser.dir=${CUBE_SERVER_WORKDIR}"
  - New content:

    JAVA_OPTS="-Dcsdb2maxconpool -Duser.dir=

    ${CUBE_SERVER_WORKDIR}”
- Windows platforms:% CUBINGSERVICES_HOME%\ cubeserver.bat
  - Old content:

set JAVA_OPTS=-Duser.dir=%CUBE_SERVER_WORKDIR%

– New content:

set JAVA_OPTS=-Dcsdb2maxconpool -Duser.dir=
%CUBE_SERVER_WORKDIR%

6.2.3 Tuning scenarios

There can be several scenarios that require you to tune your cube. Often it can be that you are having problems with *huge* cubes. But what exactly is a *huge* cube? It might be that you have a lot of data in your fact table, it might be that you have many members in your dimension, or it even might be that you have a lot of hierarchies. So these things can have an impact on different performance related elements such as the Advisors, the MQTs, and so on. In this section we discuss several scenarios and how you can get better performance when using a *huge* cube. But be careful, each Cubing Services environment differs and you must determine the best solution for your system.

Drill-through

In section 6.2.2, “Cube server tuning” on page 173, it was said that the member cache loads all members of all dimensions into the cache if static cache is adjusted. When you have big dimensions with a lot of members it can happen that it takes quite some time until the cube is started, or it might even happen that there is not enough memory available to start the cube, for example, this can happen when you want to analyze a large company with millions of customers and you have a dimension where the lowest level contains all customers as a member. The best solution is to think about adjusting the design of the cube model and reducing the members of dimensions. The easiest way is to delete the lowest level of a big dimension, so the number of members decreases significantly. However, you probably still want to see the data at the lowest level. To solve this problem you can use a Drill-through. With a Drill-through it is possible to access the data directly from your relational database, so that these data need not be loaded into the cube. Furthermore, when you implement such a Drill-through you know exactly how the data is accessed, and you can tune the query as it was described in section 6.2.1, “Relational database tuning” on page 165, so that it has a limited impact on performance.

Drill-through: There are a number of ways to implement Drill-through, and an easy example of how to do this with Alphablox is described at:

http://publib.boulder.ibm.com/infocenter/ablxhelp/v9r5m0/index.jsp?topic=/com.ibm.db2.abx.dev.doc/abx-t-develop-226.html

Splitting cubes for different business issues

If you have a lot of data in your fact table and you want to store it all in the cache of the cube, think about splitting the data into special topics, for example, you can divide the business data into different geographies. So if your company can be segmented into three geographies, such as East, West and Central, and you have a large number of reports that belong to each geography, then a good solution would be to create a cube for each geography. So for every cube with its related geography, only the data needed must be loaded into the cache. But do not forget to build up a specific view that contains a combined view of the data of all the geographies.

Using one cube on multiple cube servers

There also exists the opportunity to deploy the same cube on multiple Cube Severs, which reside, perhaps, on different hardware platforms. So it would be possible to split the workload onto each cube, and enable users to be connected with multiple cubes. Another possibility is that you can analyze the workload that each report generates.

If there are reports that execute similar queries, then there is the opportunity for those reports to use the same cube, that there is a better chance that the right data resides in the cache of that cube, and that the data does not have to be loaded from underlying relational database (which has an impact on performance). For connecting special reports or special users to multiple cubes, there are many options given by your frontend and your Web/Application Server.

When thinking about such a solution, you just develop the cube one time. But both cubes can access their data from the same database. Furthermore, both cubes can profit from performance tuning optimizations, such as MQTs.

Production statistics in a development environment

In a development environment, it might be interesting to have the same table statistics that you have in the production environment. That way, the DB2 optimizer can generate the same access plans in both environments, which allows for simulating the indexes and MQT usage at production, while still in the development environment.

To collect the statistics, use the DB2 db2look utility in the production environment with the -m option. The -d option lets you specify the database name. An example db2look statement is:

```
db2look -d sample -m > stats.dml
```

Connect to the corresponding database in the development environment, and run the script generated in the previous example. Remember to edit the script to

alter the database name, and any other parameters needed, before running it. An example statement is:

```
db2 -tf stats.dml
```

To restore the real statistics, just run the RUNSTATS utility for the desired tables in the development environment.

Logs

During the test phase, performance tuning, or problem debugging, some logs can be turned on. Use the InfoSphere Warehouse Admin console to manage the Logs:

- In the navigation tree open **InfoSphere Warehouse** → **Cubing Services** → **Manager Cube Servers**.
- Click on **Cube Server name** and then select the Logging tab.

See detailed instructions in the book, *InfoSphere Warehouse: Cubing Services and Client Access Interfaces*, SG24-7582. In particular, see Chapter 6, section 6.3.7 “Logging and Tracing.”

6.3 Impact of using MQTs and MDCs

In a Data Warehouse environment, it is common that a huge amount of data exists. But the Cubing Services cube only extracts that data that is required. To support this type of environment, and to enable good performance during the initial filling process of the cube, we can take advantage of MQTs and MDCs. In this section, we discuss the design possibilities for using MQTs and MDCs.

6.3.1 Materialized Query Table basics

A Materialized Query Table (MQT), is a special type of a table that can be created and used to enhance query performance. It is created based on the results of a query that joined a group of tables. The data resulting from the query is directly stored in the MQT. This data then already exist in a pre-aggregated form to be accessed by the queries that are used to fill the cache of the cube, as shown in Figure 6-7 on page 187.

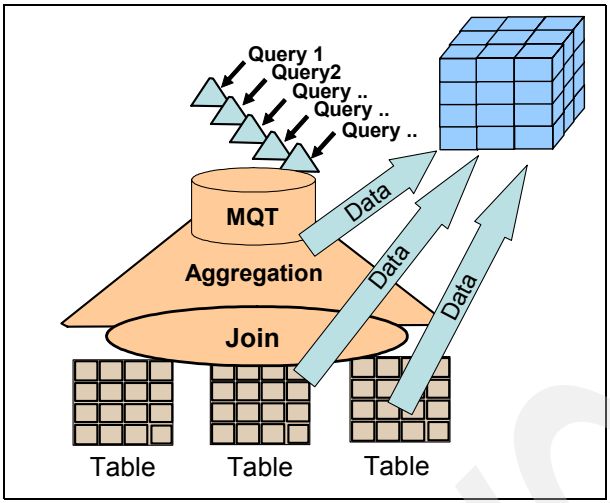


Figure 6-7 Materialized Query Tables

You can use the same guidelines for optimizing an MQT as are used for a normal table. As examples, runstats, indexes, and table space definitions.

An example of creating an MQT is depicted in Example 6-1. You can see that the table definition begins just as when using DDL for creating a normal table. However, the columns and data types are not defined. No, there is an SQL query that describes the structure of the MQT, and furthermore the MQT is filled with the data that the query returns.

Example 6-1 MQT example

```
CREATE TABLE MY_MQT AS (
   SELECT SUM(T1.Sales) as Total_Sales,
      SUM(T1.COGS) as Total_Costs,
      SUM(T1.Expenses) as Total_Expenses,
      T2.Prd_Family as Prd_Family,
      T3.Years Year,
      T3.Quarter as Quarter
   FROM SALES_FACT T1, PRODUCT T2, TIME T3
   WHERE T1.Product_ID = T2.Product_ID AND
      T1.Time_ID = T3.Time_ID
   GROUP BY T2.Prd_Family, T3.Year, T3.Quarter
)DATA INITIALLY DEFERRED REFRESH DEFERRED
ENABLE QUERY OPTIMIZATION MAINTAINED BY SYSTEM
```

Based on the MQT DDL from Example 6-1 on page 187, we show how the DB2 Optimizer decides whether to use the base table or the MQT. In Figure 6-8, we show how the routing mechanism of the MQT works. When a query is posed against the database, the DB2 Optimizer would rewrite the query if the query matches the MQT definition and if using the MQT would reduce the query costs. That means when the access plan has a better result with the MQT, then the query is posed against the MQT to access the data. To determine whether or not the DB2 Optimizer uses the created MQTs, you can use the DB2 Explain facility. For an explanation of that facility, refer to section 6.4.5, “DB2 Explain Facility” on page 200.

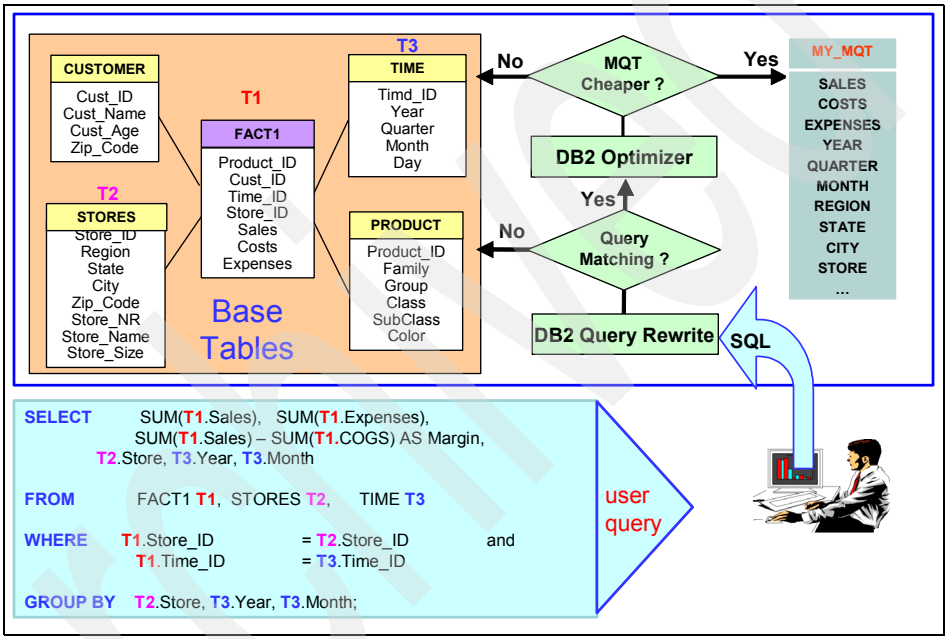


Figure 6-8 DB2 MQT Routing Mechanism

- Some considerations when creating an MQT are:
  - Create MQTs for often used queries, and those needing a lot of resources.
  - Create indexes for the MQTs (test if the indexes are used).
  - Update the statistics of MQTs periodically (with Runstats).
  - MQTs must have the same, or a higher, isolation level than the query posed against the MQT uses.
  - MQTs often need a lot of disk space (test if the MQTs are used).

– Determine a limit for the amount of disk space available for MQTs. In general, do not allocate more than 10% to 20% of the total system storage of a data warehouse for MQTs.
– Add more resources for the Sortheap, Bufferpool, CPU, and tempspace.
– Use the DB2 Design Advisor for MQT recommendations.

When the underlying data of an MQT changes, the data in the MQT is also affected. In section 6.5.2, “Update scenarios” on page 205, we discuss how an MQT can be refreshed.

6.3.2 MDC basics

Multidimensional Clustering (MDC), is concerned with physically clustering a table on one or more keys, to improve the performance when accessing the data by using multiple dimensions. In other words, it groups all rows with similar values on multiple dimensions in the same physical location in the table, called a block. It organizes the data for fast retrieval, particularly for queries that involve ranges of multiple predicates. The clustering with an MDC is done automatically and is maintained dynamically. Therefore, no reorganization is necessary for re-clustering. In Figure 6-9 we show an example of how a table can be divided into multiple blocks, such as in the example where region and time are grouped in one block. Those blocks are accessed using *block indexes*, where each of them points to one block.

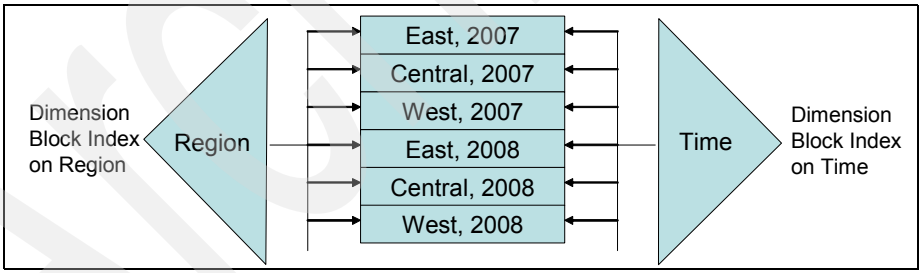


Figure 6-9 Multidimensional Clustering

When you want to change a regular table into an MDC table, you must export the data of the table, drop the table, create the table as an MDC table, and load the exported data into the new table. The data is automatically clustered in the new table.

MDC: An MDC provides a unique and powerful solution for large databases, such as in a Data Warehouse environment. It extends the performance advantages of clustering to multiple dimensions.

Example 6-2 is an example for creating a table with MDCs.

Example 6-2 MDC example

```
CREATE TABLE sales(
   Product_Number INTEGER,
   Region CHAR(10),
   Date DATE,
   Revenue DOUBLE,
   Order_Month INTEGER generated always as month(order_dt))
ORGANIZE BY (Region, Order_Month)
```

- Some considerations for creating an MDC are:
  - Look for columns that are used as predicates for equality, inequality, range, and sorting.
  - Leverage MDCs with optimal space utilization, and strive for densely filled blocks.
  - Keep the number of cells reasonably low to limit how much additional storage the table requires when converted to an MDC form.
  - Use generated columns to create coarsifications (reduced cardinality of a clustering dimension) of a table column that has much lower column cardinality, for example, create a column on the month-of-year part of a date column, or use (INT(column_name))/100 to convert a DATE column with the format Year-Month-Day to Year-Month table size so that, in most cases, you can ignore the storage required for them.
  - Do not select too many dimensions. It is rare to find useful designs that have more than three MDC dimensions without unreasonable storage requirements. As you have more dimensions, the cardinality of cells increases exponentially.
  - Consider single-dimensional MDCs. A single-dimensional MDC can still provide massive benefits compared to those of traditional single dimensional clustered indexes.
  - It might take trial and error to find an MDC design that works really well.

Tip: Use the MDC selection capability of the DB2 Design Advisor with a representative workload to find suitable MDC dimensions for an existing table. The Design Advisor is described in section 6.4.3, “DB2 Design Advisor” on page 196.

6.3.3 Using MQTs and MDCs together

It is also possible to combine the advantages of MQTs and MDCs. On one hand, you can build an MQT on tables that are partitioned with MDCs. On the other hand, you can create an MQT with an MDC. Unfortunately the Optimization Advisor does not suggest MDCs on MQTs, but here are some hints for selecting MDCs on MQTs:

- Consider using a time dimension level key member:
  - Example: Year, Quarter_ID, Month_Number
  - Time is a dimension that is common for having selective predicates
- Consider adding additional level key members with lower cardinality

Example 6-3 shows one way to create an MQT as an MDC.

Example 6-3 MQT with MDC Example

```
CREATE TABLE sales(
   SELECT SUM(T1.Sales) as Total_Sales,
      SUM(T1.COGS) as Total_Costs,
      T3.Year Year,
   FROM SALES_FACT T1, TIME T3
   WHERE T1.Time_ID = T3.Time_ID
   GROUP BY T3.Year)
DATA INITIALLY DEFERRED REFRESH DEFERRED
ENABLE QUERY OPTIMIZATION MAINTAINED BY SYSTEM
ORGANIZE BY (Year)
```

6.4 Tools for tuning

For optimizing the Cubing Services environment, there are many tools that are provided. These tools reside directly in Cubing Services, either in the InfoSphere Warehouse or in DB2. You can use those tools for recommendations of MQTs, indexes, MDCs, or other elements that can be helpful for performance optimization. With some tools, it is possible to detect the queries that are

executed during the population process of the cube cache, and moreover it shows potential problems that can occur during a query execution.

6.4.1 Optimization Advisor

After the implementing phase of the cube model, create MQTs that are helpful in populating the cache of the Cube much faster. The Optimization Advisor helps you to determine the best MQTs:

Opening the Optimization Advisor:

1. Open the Design Studio.
2. In Database Explorer, open the database where the cube was implemented.
3. Right-click the cube model, which resides under the following tree path: Schema → {Schema-Name}/OLAP Objects/Cube Models, and choose the **Optimization Advisor** option.

- The Optimization Advisor is a model-based advisor and, as is shown in Figure 6-10 on page 193, it bases its analysis on a number of sources:
  - The metadata of cube model, for example:
    - How many cubes reside in the cube model
    - Dimensions, Hierarchies, Levels, Attributes per cube
  - DB2 statistics
  - Data sampling of the base tables
  - Time and space constraints
  - Query types

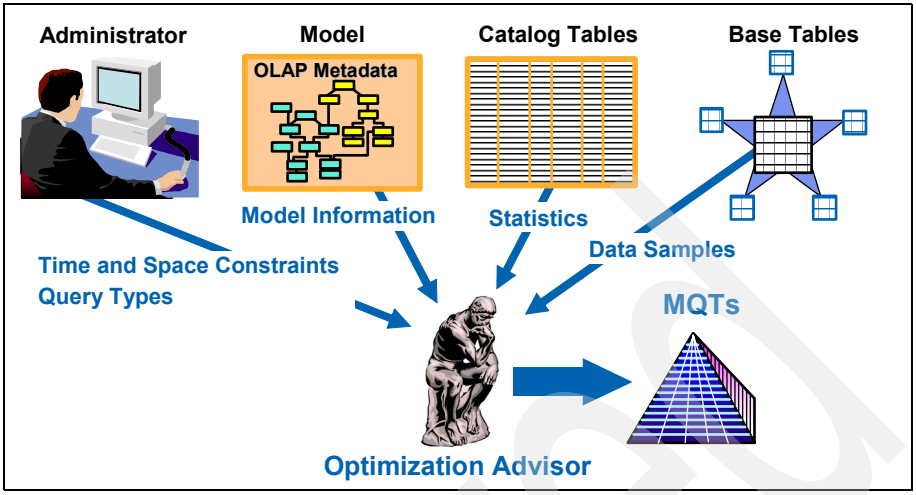


Figure 6-10 Optimization Advisor - model-based

- When you run the Optimization Advisor Wizard, you can set up the following properties:
  - Adjusting the Optimization Slices is helpful when you know exactly which level from each hierarchy in your reports is used most often. You can define as many slices as you want. Use the following guidelines when specifying an option for each cube dimension:
    - Specify a specific level in a cube dimension, such as Month in the time cube dimension, if you know that the specified level is important or frequently queried.
    - Specify *All* in a cube dimension only if the highest aggregation of the cube dimension is important or frequently queried.
    - Specify *Any* in a cube dimension if no level is significantly more important than any other level in that cube dimension, many levels in that cube dimension are queried, or you do not know how often each level in that cube dimension is queried.

Tip: In general, specify the *Any* option for most cube dimensions, and select a specific level only when that level is involved in intense query activity. If you are not sure how to set up these Optimization Slices, do not define any. The Optimization Advisor gives you the best available setting even without your adjusting the Optimization Slices.

  - You can choose if the MQTs are created with the deferred option or with the Immediate option. In section 6.3, “Impact of using MQTs and MDCs” on page 186 we explain how these options influence the MQT.
  - Define in which table space the MQT is created.
  - If you want to limit the maximum amount of disk space available for the MQTs, it is a selectable option.
  - It is a selectable option to specify in which time the Optimization Advisor gives you advice on how to create the MQT.
  - Choose whether or not you want the Optimization Advisor to make a data sampling for its analysis.
- When you use the Optimization Advisor, you get these benefits:
  - Smart aggregate selection
  - Smart index selection for the created MQTs
  - SQL generation/Hide MQT complexity
  - DB2 exploitation
  - Constraints and rollups

The Optimization Advisor generates DDLs for you. You can adjust the DDLs, and you must deploy them on the database and on the Cubing Services Server.

There can be some effects when you are limiting elements, such as size and time, within the Optimization Advisor. In general, the longer the advisor runs, the better the quality of MQT recommendations. When using sampling, first the analysis is done using a small sample size. After the analysis is complete, if there is still time remaining, the sample size is increased and analysis runs again, which repeats until the time limit is expired. So, in general, longer running times provide higher sampling rates, which results in more accurate analysis and higher quality MQT recommendations. For the size limit, that just means that after doing the analysis and determining some objects that are useful, we only recommend the most valuable objects that fit within the specified size limit. So if the analysis finds that three MQTs are useful, but based on the limit they can only fit two MQTs, then we only recommend the two most valuable MQTs.

6.4.2 Optimization Wizard

In the InfoSphere Warehouse Admin Console you can also optimize the cube model. To open the Optimization Wizard:

1. Open the InfoSphere Warehouse Console, and select **InfoSphere Warehouse** → **Cubing Services** → **Manager OLAP Metadata**.
2. A window opens, as shown in Figure 6-11 on page 195. Select a cube model, and click **Optimize**.

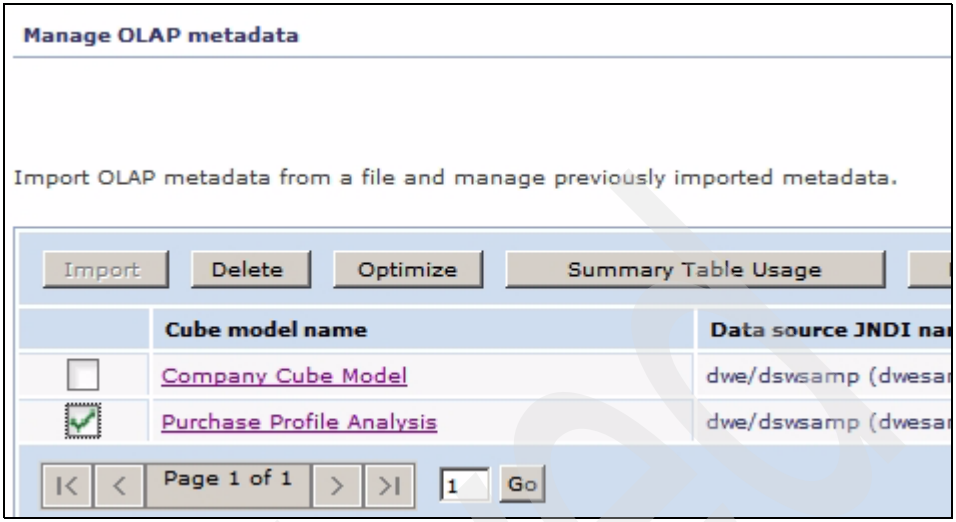


Figure 6-11 Optimization Wizard

In the next steps, you select the parameter values of the optimization process, for example, should the MQTs be created as *deferred* or *immediate,* and in which table space should the MQTs be created. When the Optimization Wizard finishes with its recommendations, it provides two DDLs that you can download, as shown in Figure 6-12 on page 196. One DDL is for creating the MQTs, and the other DDL is used to refresh the MQTs.

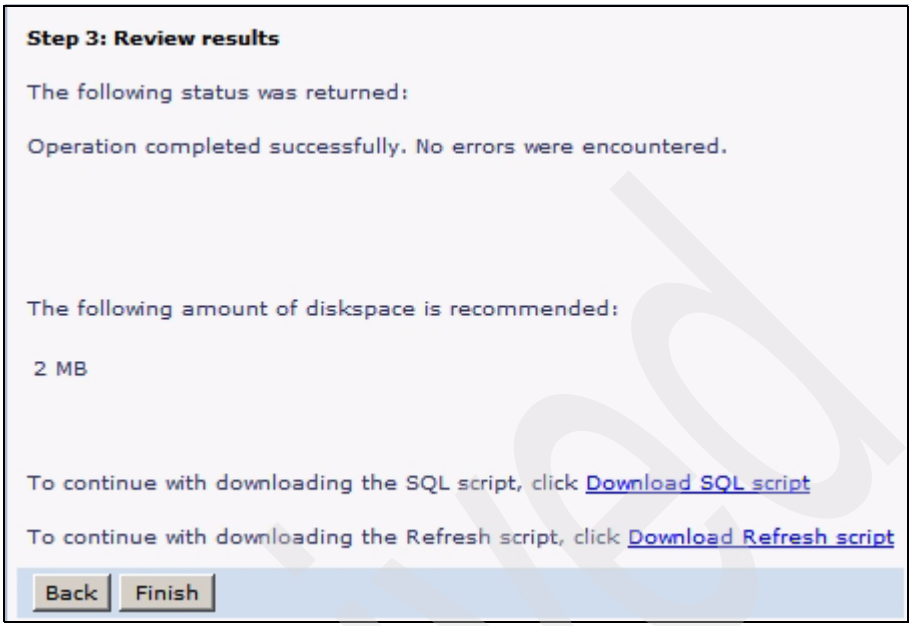


Figure 6-12 Optimization Wizard recommendations

6.4.3 DB2 Design Advisor

With the DB2 Design Advisor query performance can be significantly improved. It helps you to determine the best solution for creating:

- Indexes
- Materialized Query Tables
- Multidimensional Clustering Tables

It also gives you the opportunity to delete existing recommendations if they are no longer used.

Analyze and determine which queries are used during the populating phase of the cube cache. You can obtain that information using the DB2 Explain Facility, which we describe in section 6.4.5, “DB2 Explain Facility” on page 200.

You can use the DB2 Design Advisor at the command line or as a graphical tool:

- When you want to use the command line interface, use the following DB2 commands:
  - To analyze just one query, use the following command:

    ```
    db2advis -d <db-name> -s “<sql-command>“
    ```

– When there is more than one query to be analyzed, insert all queries into one file and deliver the file-name within the **db2advis** command, as shown in the following example:

```
db2advis -d <db-name> -i <file-name>
```

Tip: More information about the **db2advis** command is at the following Web site:

http://publib.boulder.ibm.com/infocenter/db2luw/v9r5/index.jsp?topic=/com.ibm.db2.luw.admin.cmd.doc/doc/r0002452.html

- The graphical interface of the DB2 Design Advisor resides in the DB2 Control Center, which you enter by selecting a database and choosing the DB2 **Design Advisor** option.

  With the Design Advisor, you can perform actions and specify parameter limits, such as:

  – Create indexes, MQTs (immediate or deferred refresh), and MDCs
  – A specific workload (specify a set of SQL queries)
  – Maximum disk space for the recommended objects
  – Tablespace and Schema names for MQTs

  Furthermore, you must set the following properties:

  – Tables whose statistics must be updated
  – Start time for the recommendations process
  – Maximum time needed for recommendations

Tip: To get the workload that is needed for Design Advisor recommendation, you can collect the queries that Cubing Services submits in the Cubing Services log.

After that, a recommendation is made to improve the query, as is shown in Figure 6-13 on page 198. It is then your decision as to which of those recommendations you want to implement and which old recommendations to delete.

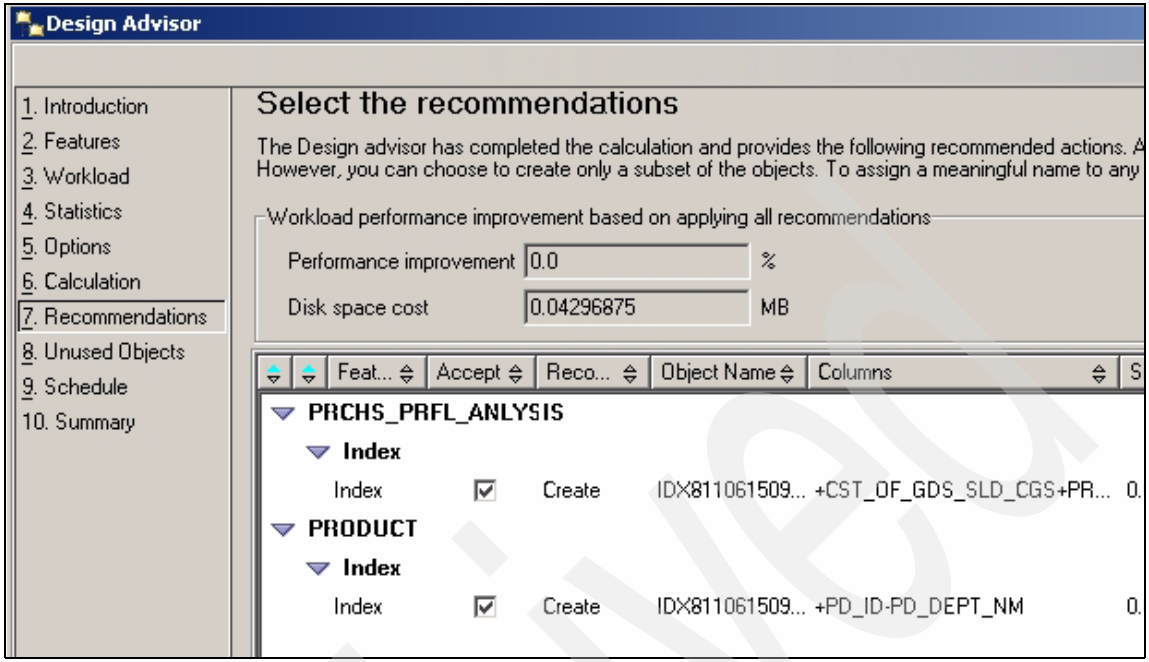


Figure 6-13 Design Advisor recommendations

Checking recommendations: Run the DB2 Advisor periodically to examine whether or not the recommendations are still correct because the data basis might have changed. You do not need to specify the workload again because it was saved for you the first time that you specified it; however, statistics for the base tables must be updated.

6.4.4 DB2 SQL monitoring

In this section, we explain how you can use the DB2 Monitoring tools to get information that is helpful for Cubing Services to analyze which SQL queries to execute and when the cache of the cube is filled with data from the relational database.

DB2 Snapshot

There are many DB2 snapshot functions, but here we only consider the SNAPSHOT_DYN_SQL table function, which provides a simple method to retrieve selected information from the dynamic SQL statement section of the package cache.

Execute the following command to remove all dynamic SQL statements from the package cache before you analyze the cube filling process:

```
flush package cache dynamic
```

Execute the following statement after the cube cache is filled to see which dynamic SQL was executed:

```
SELECT STMT_TEXT, NUM_EXECUTIONS,ROWS_READ,STMT_SORTS,TOTAL_EXEC_TIME
      FROM TABLE( SNAPSHOT_DYN_SQL('<db-name>',-1)) as SNAPSHOT_DYN_SQL
      WHERE NUM_EXECUTIONS > 0 AND ROWS_READ > 0
      ORDER BY ROWS_READ DESC;
```

DB2PD

DB2PD is an easy-to-use command line tool that determines database problems and analyzes activities that occur on DB2. So it is possible that you can monitor the workload that is executed on DB2 during the initial filling process of cube cache. It is quite easy to do this on the command line, and moreover you have a number of options to monitor DB2 on the database and instance level. For our Cubing Services environment, we show you how it is possible to monitor the executed SQL with the DB2PD tool. Use the following command to get the current executed dynamic SQL and to save the output in a text file:

```
db2pd –db <db-name> –dynamic -file <filename>
```

Tip: More options that can be specified for the db2pd tool are at:

http://publib.boulder.ibm.com/infocenter/db2luw/v9r5/index.jsp?topic=/com.ibm.db2.luw.admin.cmd.doc/doc/r0011729.html

DB2 Activity Monitor

The DB2 Activity Monitor is a graphical monitoring tool that assists you in improving the effectiveness of database performance and problem determination and resolution. It provides easy access to relevant and well-organized monitor data through a set of predefined reports, for example, with this tool you can discover the queries that need the most time and have the most impact on poor query performance, or just use it to analyze which queries execute during the cubes cache initialization. With this information, you know where you must improve the performance.

Tip: More information about the DB2 Activity Monitor is at:

http://publib.boulder.ibm.com/infocenter/db2luw/v9r5/index.jsp?topic=/com.ibm.db2.luw.admin.mon.doc/doc/c0021593.html

6.4.5 DB2 Explain Facility

With the DB2 Explain Facility you can analyze queries and locate where potential performance problems exist. Furthermore you can determine whether or not the MQTs and indexes that you already created are used for the query execution. But be sure that the statistics of the tables are up to date so that the DB2 Optimizer chooses the best execution plan. The DB2 Explain Facility reminds you if the statistics of the used tables are not up-to-date. Here again, you have the choice to use a command line interface or a graphical one:

- The commands to use for the command line interface are:
  - This command example saves the access plan in a file:

    ```
    db2expln -d <db-name> -schema % -package % -q <statement> -output
    <file-name>
    ```

  - To show the access plan directly in the command window, execute the following command:

    ```
    db2expln -d <db-name> -schema % -package % -q <statement>
    -terminal
    ```

    Tip: More information about the **db2expln** command is at the Web site:

    http://publib.boulder.ibm.com/infocenter/db2luw/v9r5/index.jsp?topic=/com.ibm.db2.luw.admin.cmd.doc/doc/r0005736.html

- The graphical interface resides in the DB2 Control Center, and you right-mouse click to open. Select the **Explain SQL** option. An SQL statement window opens, and you can enter the statement to be analyzed, and then the access plan for the query is shown.
- You can also use the graphical interface of the Visual Explain Facility within Data Studio. Use it to collect and save all queries that are executed during the initialization process of the cube in a Data Studio Project. So when you want to control the access plan of cache workload, just open the related project, and right-mouse click the query to be analyzed. Choose the Visual Explain option, as shown in Figure 6-14 on page 201.

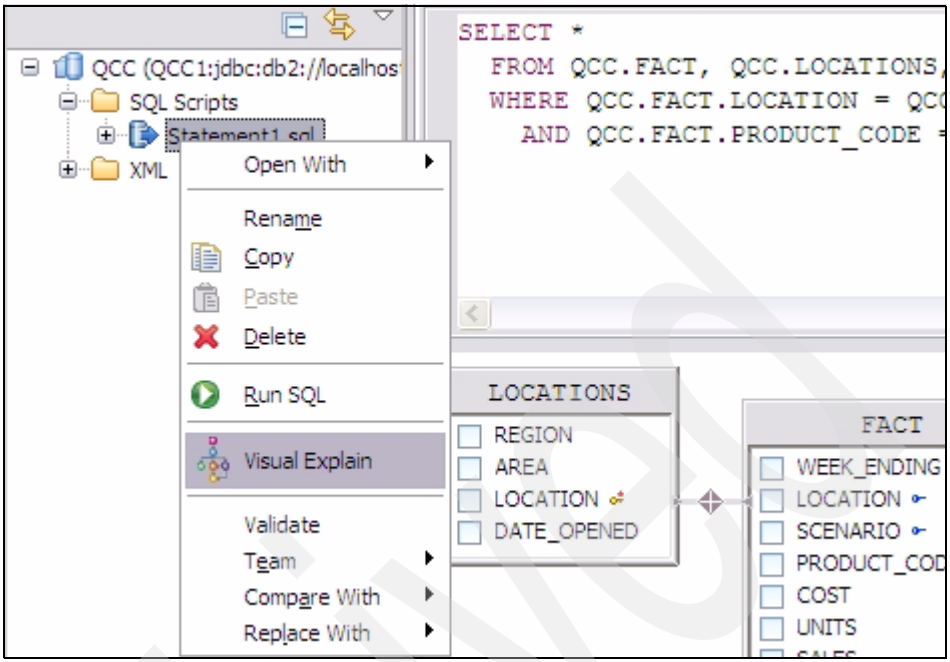


Figure 6-14 Data Studio- DB2 Visual Explain

The access plan opens, as shown in Figure 6-15.

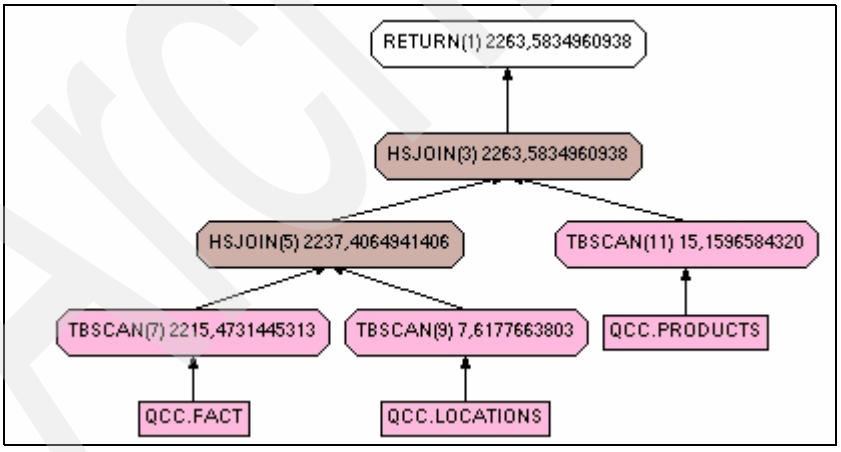


Figure 6-15 Data Studio- DB2 Visual Explain - access plan

6.4.6 DB2BATCH

DB2BATCH is a benchmark tool that you can use at the command line. With this tool, SQL queries are read, prepared, and finally described with different options. In section 6.2.1, “Relational database tuning” on page 165, we show a number of facts to consider for adjusting DB2, such as query optimization level and concurrency. With the DB2BATCH Facility, you can determine the best set up for your environment, for example, when you execute the following db2batch command:

```
db2batch -d <db-name> -f <filename> -iso <isolation level>

-o p <perf_detail> o <optlevel>
```

- -f specifies the filename, which includes the SQL statements to be analyzed
- -iso specifies the isolation level, which is used for execution of the SQL statements
- -o :
  - p specifies the details of performance analysis (0: no information, 5: all information)
  - o specifies the optimization level, which is used for the executed queries

6.4.7 Performance Expert

With the DB2 Performance Expert you can get a detailed analysis of a systems performance. It provides you with a monitor and analysis facility from your DB2 data and from the operating system too. So it is possible to see the impact on the database and on the operating system during the population process of the data into cache of the cube. Furthermore, you can identify the SQL statements that are used and that need significant time for execution during that process, for example, you can monitor the following performance relevant elements:

- Locking conflicts and deadlocks
- Applications and SQL statements causing high workloads
- Buffer pool, cache and heap sizes
- Sufficient table space
- CPU, memory, file system shortages
- Full information about each data partition

Tip: More information about the DB2 Performance Expert is at the following Web site:

http://www-01.ibm.com/software/data/db2imstools/db2tools/db2pe/db2pe-mp.html

6.5 Cube update

In a cube the data is cached if the cube is running or you explicitly flush the cache. But the data of the underlying database, which delivers data to the cube, can change and the cube does not notice the modification of the base data; therefore, the results that the cube provides are no longer current. Therefore you must develop a strategy for how and how often the data of the cube is refreshed. In this section, we present considerations for you as you develop an update strategy, and we show examples of how you can update the cubes.

6.5.1 Frequency of cube updates

The frequency with which you update your cubes depends on a number of considerations. You must understand which view of the data the cube provides in the reports. If you give a summary of the last day, last week, or last month, simply refresh the cube as often as the data is needed in your reports. The second consideration is the volume of data that is stored in the cache of the cubes. If a large volume of data must be populated, then you must plan for it to be there sufficiently long, and used, to justify the population time. Other considerations are, as examples, how many users analyze the cube data, on which hardware does the Cube Server reside, and how many cubes are stored within one Cube Server.

Consider how often the data in a Business Intelligence environment changes. It is common to have ETL processes that load the data into the data warehouse from the operational data sources. Typically there is a window of time periodically (once a night or once a week) for running those ETL processes, which might be the best time for also refreshing the cube. So, you can extend your ETL flow to include the cube refresh processes, as shown in Figure 6-16.

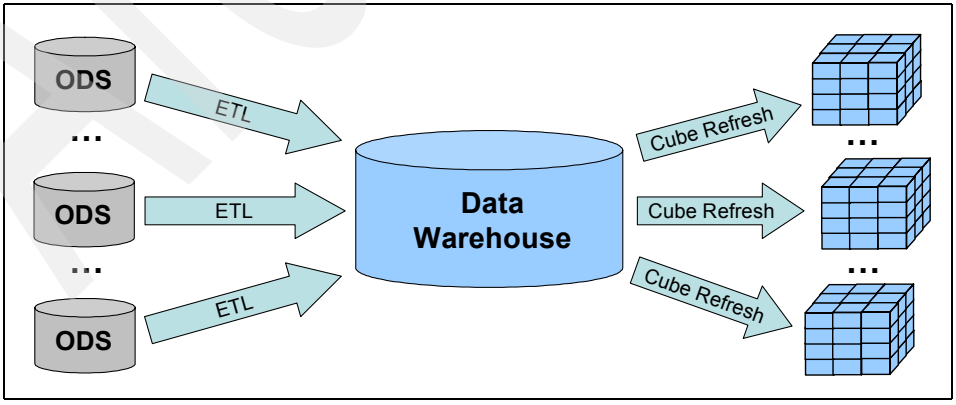


Figure 6-16 Cube - Daily Refresh

Another scenario might be that you want to compare the month-end closing data. So the data does not change during a month, and you do not have to flush the cache except on a monthly basis. Also be aware of the refresh cycles for other cubes that reside on the same Cube Server, which requires coordination or re-distribution of cubes across your Cube Servers.

In addition, you also must consider that not only must you flush the cache of the cube, but you must refresh the MQTs before you flush the cache! We provide a discussion on this topic in section 6.5.2, “Update scenarios” on page 205.

Moving towards real-time updating

In the business environment today, the demand is for more, and more current, data for decision making. Business is moving fast and change seems to be nearly constant. Management wants to have the data available nearly as soon as it is collected. The focus today is on Business Intelligence, which is the driver that is moving us towards that real-time environment.

Moving towards real-time is changing everything. The data warehouse is no longer a reasonably static source of data that is updated on a scheduled basis. Now, at least some parts of the data warehouse are updated nearly on a continuous basis. The direction seems to be towards having everything updated on a continuous basis. Certainly this poses many issues for businesses that must be resolved, but that is the direction.

Because the cubes are derived from the data warehouse, they must be able to support real-time updating as well. How will you do that?

As previously mentioned, in a Cubing Service environment, the data is stored in the cache of cube until it is flushed. So it does not automatically change when the underlying data changes. It is difficult to combine the good performance of the cube cache and a refresh process for the cube. There are a number of reasons for this, one of which is that the cache of the cube must be initialized after it was flushed. So we must design the architecture, and have a good strategy, that enables updating of the cube cache.

One part of a solution is to use smaller cubes and assign lots of resources to them so they start fast. Another might be to have more than one cube for the same report. So although your reports receive the data from one cube, the other cube can be flushed and afterwards the new data can be populated. Then, at a point-in-time, you can switch internally to the cube with the new data, which can be implemented as an ongoing process that is repeated as often, and as fast, as you need the new data.

When considering these types of solutions, be aware of other impacts that must be addressed, for example, you must be sure that no one is accessing the old cube before you flush the cache.

6.5.2 Update scenarios

Now that we opened the subject and presented some of the requirements of cube updating, we look at a few scenarios.

MQT update

Before updating the data of a cube, refresh the underlining MQTs. In section 6.3, “Impact of using MQTs and MDCs” on page 186, we describe the processes for creating an MQT, which is good information to understand and use as we discuss scenarios for updating those MQTs:

- If you chose the *immediate* option, you do not have to do anything when the underlying data changes. The MQT is refreshed automatically when the data is modified. However, there are still other related issues that you must address.
- If you chose the *deferred* option, you must refresh your MQT manually. There are two options available to do this:
  - The first option is that you use the **refresh table** command:

    ```
    db2 refresh table <table-name>
    ```

    When you execute this command, DB2 determines if an incremental update of data is possible. If not, the MQT is deleted, a full-select of your MQT definition is executed, and all the aggregated data is inserted into your MQT. Finally, it updates the catalog with the refreshed time stamp and the cardinality of the MQT.

    Tip: More information about the **refresh table** command is available at:

    http://publib.boulder.ibm.com/infocenter/db2luw/v9r5/index.jsp?topic=/com.ibm.db2.luw.sql.ref.doc/doc/r0000977.html

  - Sometimes the refresh process for your MQTs is faster if you do not use the **refresh table** command. It might be better to delete and recreate the MQT, which is the second option, as shown in the following example:

    ```
    db2 drop table <mqt-name>;
    db2 create table <mqt-name> as ....;
    db2 set integrity for <mqt-name> ...;
    ```

Here again, although the refresh can be done, there are still other related issues that you must address.

MQT information: Basic information about MQTs is in section 6.3.1, “Materialized Query Table basics” on page 186.

Manual cube rebuild scenario

Within the InfoSphere Warehouse Admin Console, you can refresh the cache of the cube manually. The easiest way to do this is to restart the cube or even the Cube Server. When you restart the Cube Server, all cubes that belong to that Cube Server are restarted.

To restart the Cube Server:

1. In the InfoSphere Warehouse Admin Console, select: **InfoSphere Warehouse → Cubing Services → Manage Cube Servers**.
2. Select the Cube Server, and click **Restart**.

To restart a cube:

1. Select **InfoSphere Warehouse → Cubing Services → Manage Cube Server → Cube Server Name → Cubes Tab**.
2. Select the cube you want to restart, and click **Restart**.

You also can refresh the cube without stopping it. Therefore you have two possibilities for which parts of the cube is flushed. For both, select **InfoSphere Warehouse → Cubing Services → Manage Cube Server → Cube Server Name → Cubes Tab**, as shown in Figure 6-17 on page 207.

- If you just want to empty the data cache, select the specific cube, and click **Empty data cache**.
- When you click **Rebuild member cache**, for the selected cube, all members are rebuilt and implicitly the data cache is emptied.

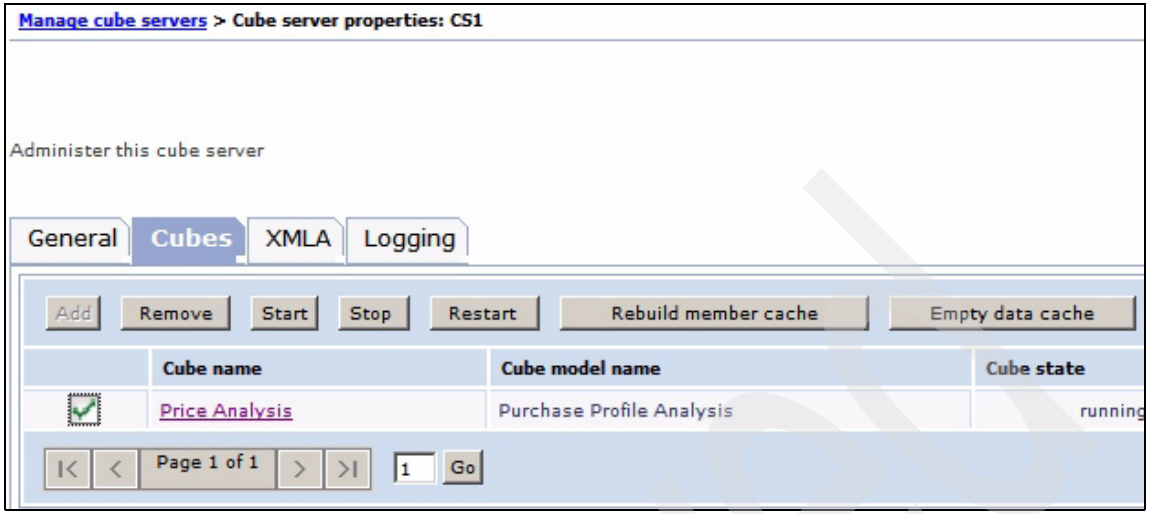


Figure 6-17 Cube Update

Automatic cube rebuild scenario

We previously described the steps necessary to rebuild a cube. Now, we describe how to combine all of them to have one process that can run automatically. We consider this scenario in an environment where the data is loaded once into a data warehouse with an ETL process. We now extend that ETL flow with the necessary steps for the cube rebuild.

When starting a cube refresh process, be sure that all data is successfully inserted into the database.

The first step is to refresh all MQTs that the cube uses. Therefore a DB2 script can be executed that first connects to the database where the MQTs reside. Execute the **`refresh table <table-name>`** command for every MQT needed. Determine whether or not it is necessary to update the statistics after every refresh of the MQT, which is the case when, after every refresh, the volume of data, or data itself, differs significantly.

Next, flush the cube so that the old data is deleted, which occurs automatically for our scenario; therefore, we cannot manually press a button as we previously described. Currently the only method that we can use to flush the cache of the cube is to restart an *operating system service* that stops and starts the Cube Server. On Windows, you must create this service, but on Linux, the Cube Server is started as a service as the default, so you do not have to create it there. For the Windows service, you find the folder *Cubing Services* in the *install directory* of the InfoSphere Warehouse. Inside resides the *bin* folder, where you must execute the following command to create the Windows service. The used

Cube Server name for this command must be equal to name of the Cube Server you defined inside the InfoSphere Warehouse console, and exactly that Cube Server is restarted when you start and stop the Cube Server service.

```
csService.bat create <Cube_Server_name>
```

In the same directory, find the start and stop commands for the operating system service of Cubing Services:

- On Windows systems execute:
  - Stopping the Cube Server: **csService.bat stop <Cube_Server_name>**
  - Starting the Cube Server: **csService.bat start <Cube_Server_name>**
- On a Linux systems execute:
  - Stopping the Cube Server: **StopCubeServer.bat <Cube_Server_name>**
  - Starting the Cube Server: **csDaemon.sh <Cube_Server_name>**

So just create two executable files, one file that stops the service for Cubing Services first and afterwards a file that starts it again.

The last step to be done is to populate the new data into the cube so that the first user of the cube has not to wait long for the specific data, which you can do in a number of ways:

1. One way is to set up a seeding query, which is executed when the cube is started. But you can use only one query. So if you want to populate a lot of data then you must write a complex query, which covers all of the data. If you have just one major report or just one large report, then you can take the initial query for that report and use it as the seeding query. To implement the seeding query:
   a. Open the InfoSphere Warehouse Admin console, and select **InfoSphere Warehouse** → **Cubing Services** → **Manage Cube Server** → **Cube Server Name** → **Cube Tab** → **Cube Name** → **Cache Tab**.
   b. Select the **Populate the cache using the following MDX query** option, as is shown in Figure 6-18 on page 209.
   c. Copy the MDX query, which should execute when the cube starts, into the text field.

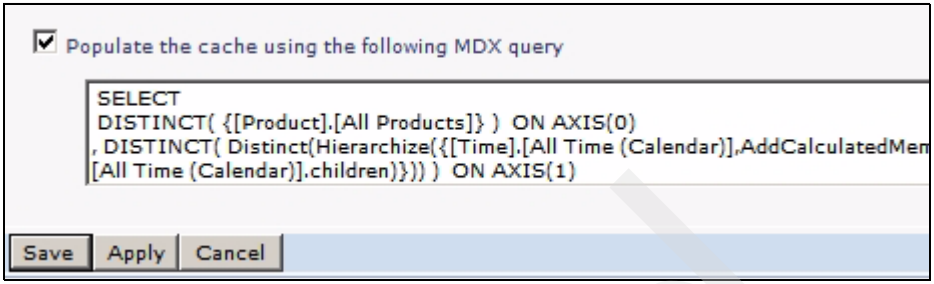


Figure 6-18 Seeding Query

2. Another possibility is to use the schedule feature from Cognos.
   - If you know exactly when the cache of the cube was flushed, you can schedule all of your Cognos reports so that the cache of the cube is initialized after the flush. In Cognos, you can schedule reports when you navigate to the folder where the report you want to schedule resides. Select the report you want to schedule and click **schedule**. As shown in Figure 6-19, the button in the box (in the lower right side of the figure, next to the words More) is the Schedule button.

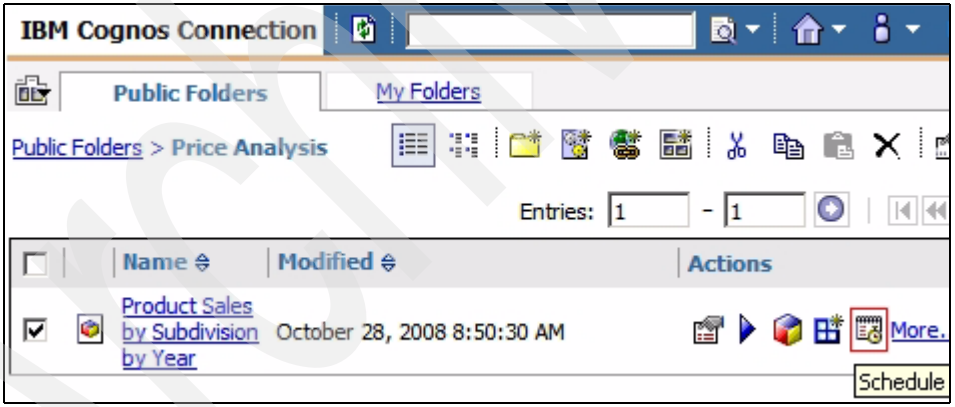


Figure 6-19 Cognos - Report Schedule 1

Now a schedule window opens, as shown in Figure 6-20 on page 210. If you want to schedule the report to run every day, click the By Day tab, and select the option Every 1 day(s). On the right sight of the panel, enter the start date and the time when the report is to start every day. On this page, you also can set the priority of the scheduled report (1 = highest and 5 = lowest) and disable the schedule, when it should not run.

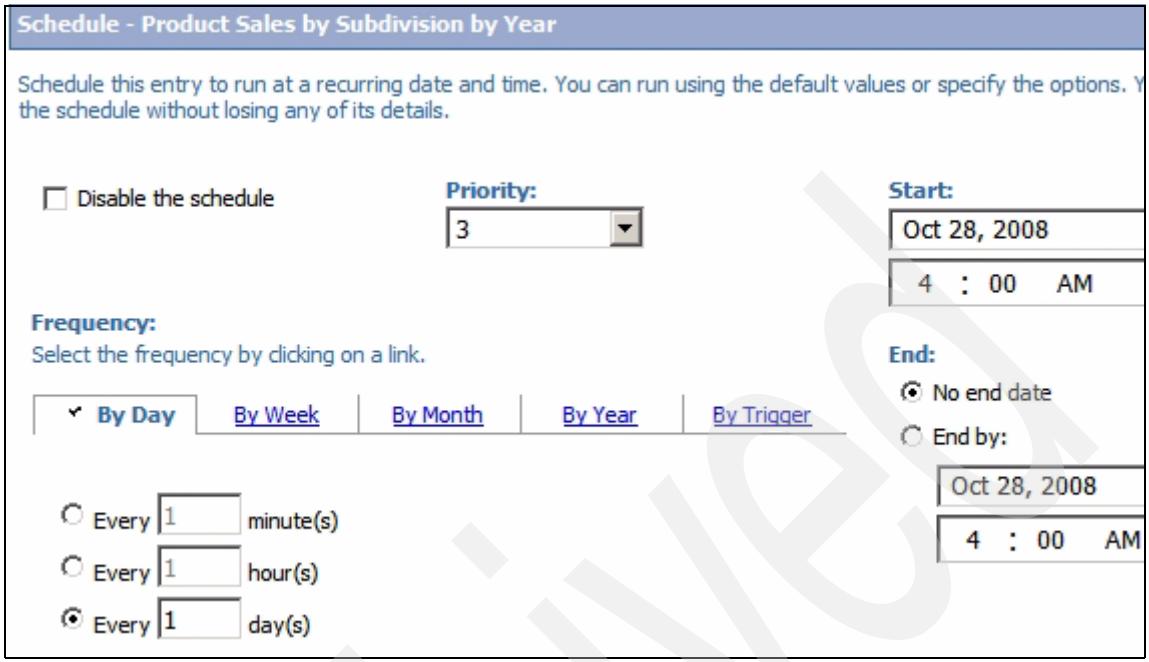


Figure 6-20 Cognos - Report Schedule 2

3. Another possibility is that you can start the reports with a Cognos trigger. These triggers can be called by an external command, such as a batch file, when you do the following:

   a. You must open the same window that you opened for the Cognos report schedule.

   b. Click the By Trigger tab, and specify a Trigger name, as shown in Figure 6-21. Click **OK** to save your changes.

Figure 6-21 Cognos - Report Trigger

c. Now you can use the trigger command, which resides in the folder \c8\webapps\utilities\trigger, inside the Cognos installation directory. On Windows, use the trigger.bat, and on Linux use the trigger.sh command. As an example for Windows, you have the following options for the trigger command:

```
trigger.bat <URL> [ <userName> <password> <nameSpace> ]
triggerList
```

Required arguments are:

- URL: IBM Cognos 8 Server URL for example. http://localhost:9300/p2pd/servlet/dispatch
- triggerList: comma separated list of trigger names, for example: triggerName1,triggerName2,triggerName3

Optional arguments for use with secured namespace are:

- userName: Username, valid within the namespace, to run the utility
- password: Password for the given user
- nameSpace: Namespace for the desired user

For the example in Figure 6-21 on page 210, you can execute the following command when no security activated and Cognos installed on a Windows operating system:

```
trigger.bat http://localhost:9300/p2pd/servlet/dispatch
Trigger_Cache
```

This command can also be included in a script file, and you can call this script within an ETL flow.

4. If you use another application that uses the Cubing Services interface, determine if there is a possibility to run initial reports against the cube. The cache is populated and you have a performance improvement because of the use of the cube cache. If there is a possibility to start these initial reports with a batch job, then include this script as it was described with the Cognos initial reports.

When you combine all of the steps to automatically refresh the cube, you gain an ETL flow with SQW, as shown in Figure 6-22 on page 212. A brief summary of all commands used, related to their operator number, is given in the following example. This example is for an InfoSphere Warehouse Cubing Services installation that resides on a Windows system:

1. Use the MQT refresh command with a DB2 Shell Script:

```
db2 connect to dwesamp

db2 refresh table <mqt-name>

db2 terminate
```

2. For updating the statistic, use the SQW runstats operator or use the runstats command:

```
db2 connect to dwesamp

RUNSTATS ON TABLE <mqt-name> ON ALL COLUMNS WITH DISTRIBUTION ON ALL
COLUMNS AND INDEXES ALL ALLOW WRITE ACCESS;

db2 terminate
```

3. Stop the Cubing Services cube with an executable file:

```
@echo off

cd <Cubing Services Inst Dir>\bin\

csService.bat stop CS1
```

4. Start the Cubing Services cube with an executable file:

```
@echo off

cd <Cubing Services Inst Dir>\bin\

csService.bat start CS1
```

5. Populate the cache of the cube with an executable file:

```
@echo off

cd \Program Files\cognos\c8\webapps\utilities\trigger\

trigger.bat http://localhost:9081/p2pd/servlet/dispatch trigger1
```

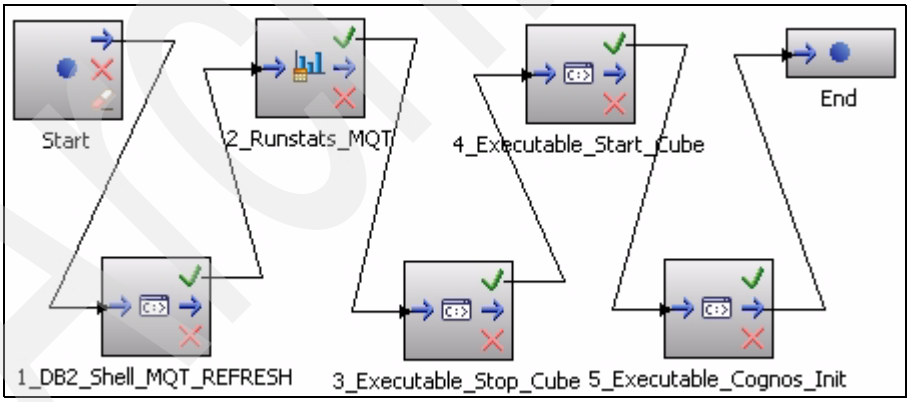


Figure 6-22 Automatic cube rebuild ETL flow

6.6 Summary of the tuning process

In the previous sections of this chapter, we gave an overview of the primary activities, all of which you must consider when you build up a Cubing Services environment. We now provide a summary of all these activities to give an end-to-end description about the entire tuning process. Figure 6-23 shows the life cycle of the Cubing Service.

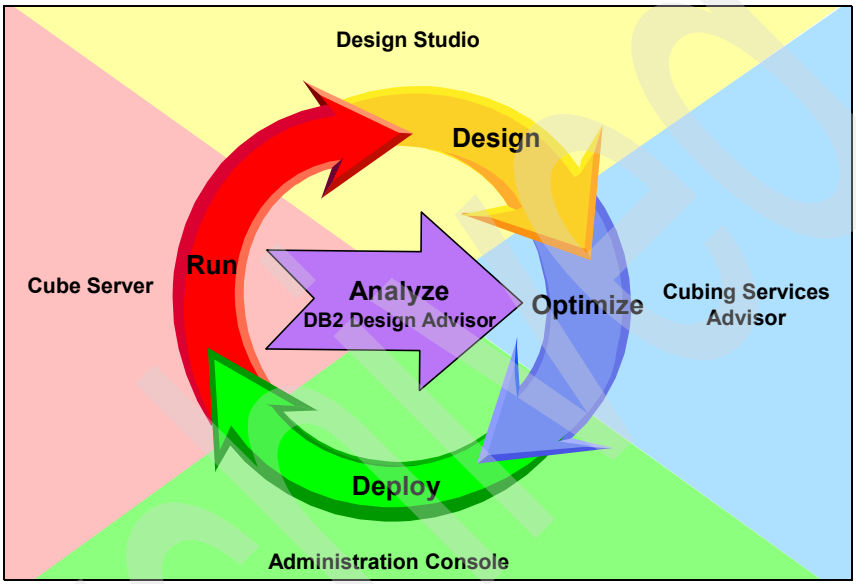


Figure 6-23 InfoSphere Cubing Service Life cycle

The cube development process consists of four steps: Design, Optimize, Deploy, and Run:

1. The first step is to design the cube, which requires the developer to understand the properties of dimensions, hierarchies, levels, measures and attributes.
2. Then gather pertinent information, such as determine how many concurrent users will access the cube, how many cubes are required, understand a typical query load, how often the cube must be refreshed, and which hardware resources are needed.These considerations must be seen in connection with your relational data model and cube model.
3. Furthermore determine the sizing of the Data Cache and Member Cache of your cube and determine whether or not there are enough resources available so that the cube starts.

4. After you deploy your cube on the Cube Server, determine if the cube starts successfully and if you get any error messages. If you have problems, think about adjusting the Data Cache and Member Cache, so that not every part of the cube must be stored in the cache of the cube. When the cube is running and you can pose queries against the database, think about how to speed up the population of the data from the relational database to cube cache. Therefore run the Optimization Advisor first to gain recommendations for creating MQTs and indexes, which are related to your cube. You can adjust the DDL script that the Optimization Advisor generates, if you know exactly how to improve it. Afterwards deploy the DDL on DB2 and on the Cube Server.
5. Examine the queries that are executed during the initialization process of the cache of the cube. To get those queries, use DB2 snapshots facilities, the DB2 Activity Monitor, or other monitoring tools. To manually analyze the queries, you can use the DB2 Explain Facility, and look at the access plan to see whether or not the MQTs and Indexes from the DB2 Optimization Advisor are used. Lastly, try to determine at which parts of the access plan the performance problems occur. To solve those problems, you can use the DB2 Design Advisor, which recommends more MQTs, MDCs, and indexes on the base of the workload from your queries, which populate the cache of the cube and the data stored in the used tables. It also tells you to delete parts that are not used.

   Check again the response time of the queries needed for the cube initialization. If there are no improvements go on with testing, which can result in further enhancements for your environment. But even if you achieve a good access plan from the workload, run the DB2 Design Advisor periodically because the data in your database can change and other MQTs and indexes improvements might be more helpful.

   The needed time to initialize the cache of the cube should now be optimized for your Cubing Service environment. Afterwards, develop a strategy for how to refresh the cache of the cube because the underlining data can change, and those changes are reflected in the cube when you restart or flush it. For designing the refresh process, think about how often the new data that is based on the cube is needed in the reports. After this decision you must think about such topics as, how to flush the cache, refresh MQTs and populate the data into the cache of the cube before the normal reports analyze the data.

   In Figure 6-24 on page 215 the entire tuning process of a Cubing Service environment is shown. But do not forget that this is an ongoing process if you use the cube, to be sure that you get the best performance for the analysis of your business data.

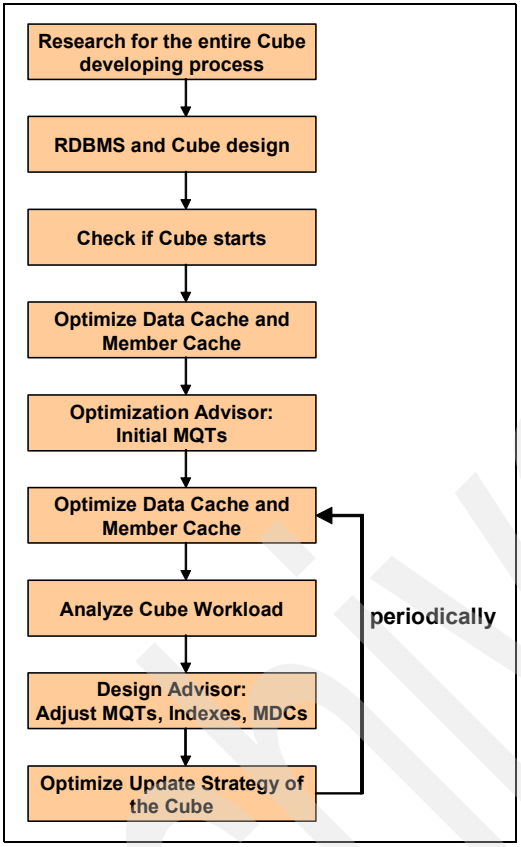


Figure 6-24 Tuning process

7

InfoSphere Warehouse on System z: Cubing Services

In recent years, there has been a building interest in bringing data warehousing and analytic tools to System z. Originally DB2 for z/OS was used for building information and reporting databases with data coming from operational systems based on existing systems, such as VSAM and IMS. However, DB2 quickly became the database of choice for business-critical operational systems, although information and reporting systems, now called data warehouses, were moved to distributed systems, which was due to a number of reasons. As examples, at the time mainframes were considered expensive, data warehouses were not deemed as mission critical, and distributed systems was where data warehouses and analytic tooling were developed. In general, the view was that it was not critical if the data warehouse was not operational because it did not affect the day-to-day running of the business. Therefore, analysts in the corporate offices could wait a few hours until the data warehouse became available.

However, over the years, the information in the data warehouse became much more important to the daily running of the business and is fast being embedded into operational applications, and many analytics are now being pushed out of the corporate offices and down into the line-of-businesses where decisions can be made much more quickly. In addition, analytical applications are even made available outside of the corporation to business partners and customers over the Web. With this increased importance, the data in the data warehouse is now

considered to be as important to the business as the more traditional operational transaction data. Therefore, if the data warehouse is not available, or response times slow, the day-to-day operations are affected.

In the meantime, the cost structure of using System z drastically changed. System z hardware cost dramatically decreased and software costs can be reduced by utilizing the System z special processors, ZIIPs, ZAAPs, and IFLs, which offer lower software licensing costs for off-loading Java, DB2, and Linux workloads. In addition, CPUs on System z are typically utilized nearly 100% with highly-mixed workloads, while workloads for distributed system are many times architected to use 20% to 40% of the CPU. The capacity on the System z can even be dynamically turned on and off to meet varying workload demands, particularly during peak periods. As was the case for a long period of time, System z is known for reliability, availability, security, and workload management, making it the choice for the most mission-critical applications in a corporation. Finally the cost of and, in some cases, limited availability of power in data centers is causing customers to consolidate hundreds of distributed systems into a System z, thereby reducing power costs or adding new capacity without increasing, and likely decreasing, power requirements.

All of these factors combined are driving customers to deploy data warehouses on System z. In response to this demand, IBM updated and expanded product capability. DB2 for z/OS V8 and V9 added SQL functionality deemed necessary for data warehousing and aggressively started porting data warehousing and analytic tooling to System z. Here are some examples:

- Rational Data Architect delivered additional data modeling capability, including logical, glossary, and domain modeling.
- IBM Industry Data Models, which are based on the experience of more than 500 clients, and more than ten years of development.
- IBM InfoSphere Information Servers for enterprise-level data integration tools such as:
  - DataStage
  - QualityStage
  - ProfileStage
  - Federation Server
- IBM InfoSphere Master Data Management Server for the management of standard corporate data such as products and customers.
- Cognos 8 BI.

Now IBM is introducing InfoSphere Warehouse on System z (ISWz). ISWz provides the core data warehousing tooling on top physical data modeling, SQL-based data movement and transformation, along with an OLAP MDX server. A basic data warehouse can be built with just the components of ISWz

installed along with DB2 for z/OS and using Microsoft Excel for the front end. As sophistication and requirements grow, the solution can grow with you and can be extended to incorporate other products as needed, as shown in Figure 7-1.

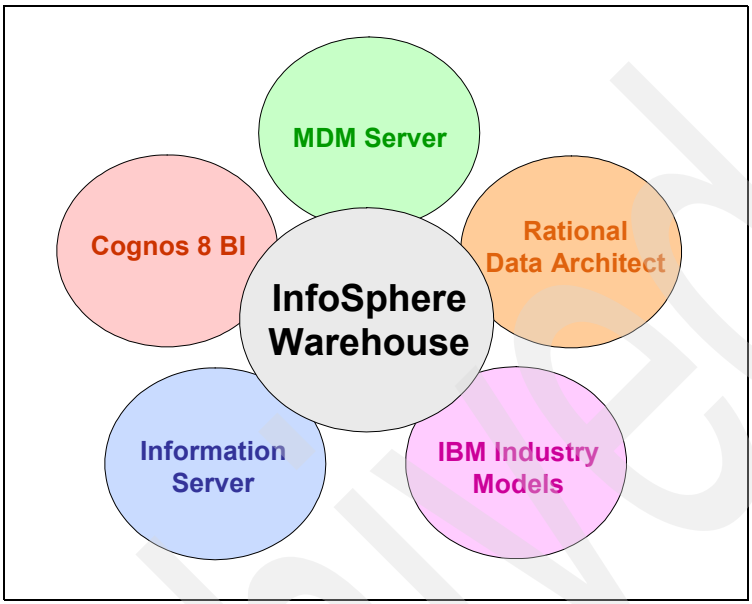


Figure 7-1 InfoSphere Warehouse provides the core data warehousing capability

In this chapter, we give a high-level overview of InfoSphere Warehouse on System z. Because this is a Cubing Services book, we focus on the Cubing Services component that is specific to the System z environment.

Important: At the time of publication, InfoSphere Warehouse on System z V9.5.2 was just entering beta. Features and functions delivered at general availability might be different from those presented in this book.

7.1 InfoSphere Warehouse on System z: an overview

InfoSphere Warehouse on System z (ISWz) adds core functionality on top of DB2 for z/OS to build a data warehouse and provide analytic data to presentation layer products. There are three primary functional components:

- Physical database modeling
- SQL-based data movement and transformation (SQL Warehousing Tool)
- OLAP cube modeling and server (Cubing Services)

An integrated development environment, called Design Studio, is used for defining relational physical database modeling, modeling the OLAP cubes, and for developing and testing data movement and transformation flows, for example, after modeling and development are completed in Design Studio, the objects are deployed to a runtime environment for system testing or production. The runtime environment consists of the OLAP engine, Cubing Services, a WebSphere Application Server-based server to execute data movement flows, and Web-based administration of the components with what is known as the Administration Console.

7.1.1 Architecture

InfoSphere Warehouse on System z consists of middleware components that reside on a System z Linux partition, a Windows or Linux-based development client, and a browser-based administration client, as seen in Figure 7-2 on page 221. These components are for the purpose of building a data warehouse in DB2 for z/OS and adding OLAP server capability on top of the DB2 data warehouse.

The client applications can access data using MDX over XMLA or ODBA connections from the OLAP engine, Cubing Services, or directly from the relational tables by using SQL. No matter which interface, MDX or SQL, a business intelligence or reporting tool uses, they access the exact same data from the data warehouse, which is done with no copying of data for OLAP usage because Cubing Services uses DB2 as its persistence layer. We call that capability *No Copy Analytics*.

The other client layer components are the integration development environment, called Design Studio, and the browser-based Administration Console. Design Studio provides the tooling that allows you to create the physical data model, define the OLAP model of cubes, and to develop and test SQL-based data movement and transformation flows.

The middleware layer in the Linux partition provides the runtime services. There is a WebSphere Application Server application that provides the back-end administration functions that a system administration sees in the browser-based Administration Console. There is also a server that manages and executes the data movement applications. The OLAP engine, Cubing Services, is a stand-alone server outside of the WebSphere Application Server that serves multidimensional data using MDX requests from client tools.

Finally, at the bottom of the stack is the DB2 for z/OS data warehouse and the source systems that feed it.

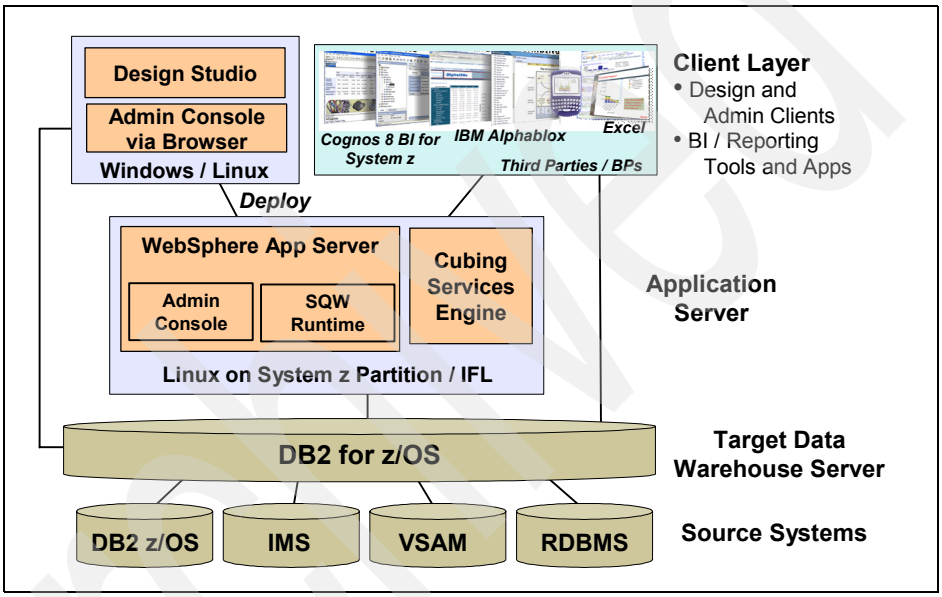


Figure 7-2 InfoSphere Warehouse on System z Architecture

7.1.2 Tooling

InfoSphere Warehouse on System z has two sets of tooling: an integrated development environment and runtime administration.

Design Studio

Design Studio is the integrated development environment that is used to develop the physical data model, the OLAP model, and data movement flows, as shown in Figure 7-3 on page 222. The Design Studio consists of a number of panes to perform tasks, such as organizing the work and housing the object-specific editors, connecting to work with live databases and various views, such as object properties, work in-progress status, and SQL results. The tasks and layout of the

Design Studio are organized into role-based perspectives. The default perspective is the Data Warehousing perspective which provides the tooling described in this chapter. Perspectives can be customized to fit the working style of the individual. The Design Studio can shell-share with other IBM Eclipse-based tooling. That means they can be installed into the same Eclipse instance and all tools appear in the workspace as different perspectives. The IBM products that Design Studio can shell share with are:

- Data Studio Developer V2.1
- Data Studio Administrator V2.1
- InfoSphere Data Architect V7.5.1
- Data Studio Optimization Exert for z/OS V2.1
- Rational Architect Developer V7.5.1
- Rational Software Architect V7.5.1

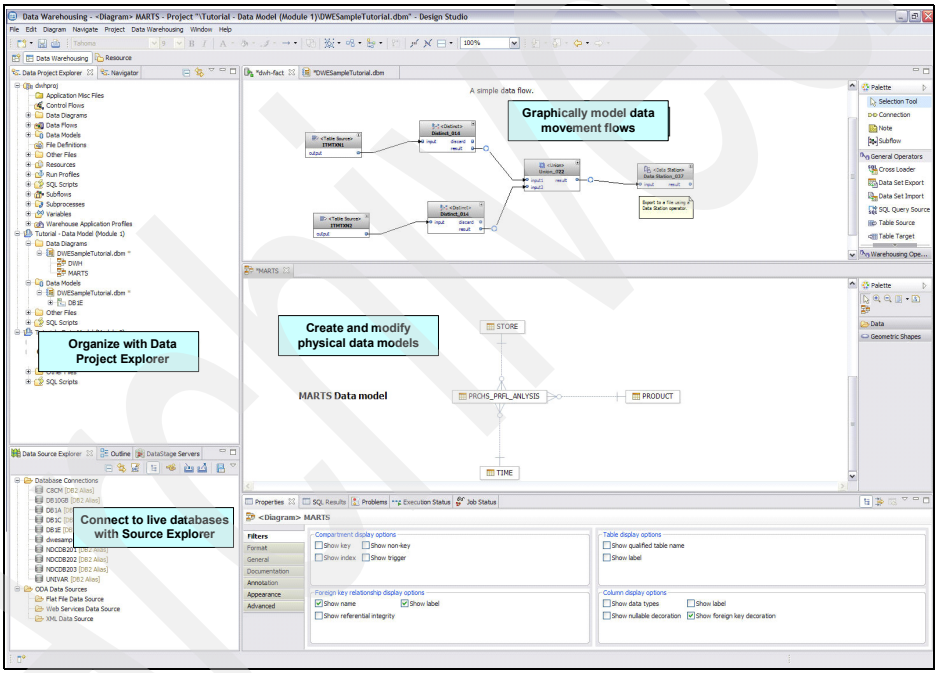


Figure 7-3 InfoSphere Warehouse Design Studio

Administration console

The Administration Console, a browser-based application for managing the runtime environment for InfoSphere Warehouse on System z V9.5.2, introduced a new Adobe Flex-based user interface. Figure 7-4 on page 223 shows the administration console.

The functions for that interface include:

- Managing database connections and other system resources
- Deploying, scheduling, and managing SQL Warehousing (SQW) data movement applications
- Defining and managing Cube Servers and Cubes

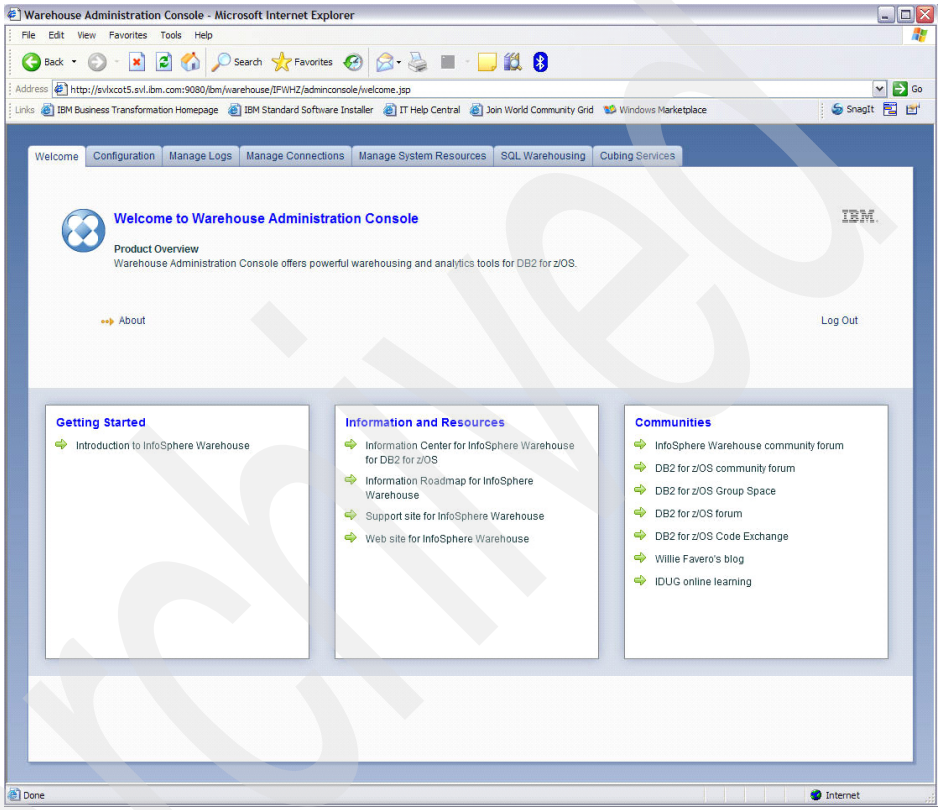


Figure 7-4 InfoSphere Warehouse Administration Console

7.1.3 Physical data modeling

A physical model of the relational database(s) is required to use the rest of the ISWz tooling. The OLAP components are built as OLAP extensions of the physical data model, and the data movement flows require metadata about the physical characteristics of the underlying database. Therefore creating a physical data model is typically the first task that you must perform.

The Design Studio data modeling component is used to create physical models, either from start to finish, or by reverse engineering the metadata from an existing database. This tooling is a subset of the enterprise level data modeling tool, the InfoSphere Data Architect (formally Rational Data Architect).

The primary purpose of the physical data model in Design Studio is to provide the metadata information about the source and target databases to the SQL Warehousing and OLAP modeling components. Table and column metadata is the most obvious, but there might be metadata representing many other object as well, such as tablespaces, storage groups, routines, constraints and so on. Physical data models can be created for the common database management systems, including DB2 on all platforms, Informix, Oracle, SQL Server and others that are supported by the Data Source Explorer and shown in Figure 7-5.

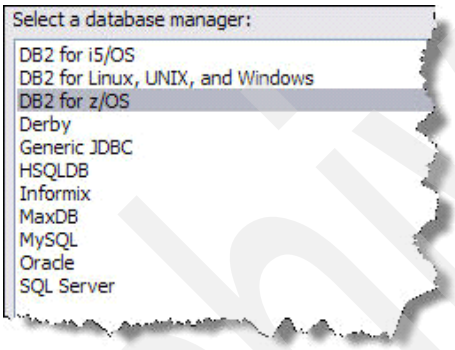


Figure 7-5 Supported Database Managers

A physical data model can be completely created from start to finish. Most of the time, however, databases already exist. If a new database is created, it is likely that the database designer uses a full-featured data modeling tool, such as InfoSphere Data Architect, to go through the entire modeling life cycle, starting with a logical model. If a physical model was created using another tool, the physical model can be imported. If a physical model does not exist then the physical data model can be reverse engineered from a live database or from the file containing the SQL data definition language (DDL).

Figure 7-6 on page 225 shows a physical data model in the Design Studio. The physical data model can be modified using either the project tree in the Data Project Explorer or graphically using a diagram. A diagram can be created over the entire data model or over subsets of the data model, which is useful to segment the data model into more manageable application oriented parts.

For DB2, you can also work with components of the underlying storage model, such as tablespaces, storage groups and VCATs. A data model can be

compared back to the original source to identify differences between the model and the database and sync the definitions, generating the appropriate delta DDL. Impact analysis can also be done for a model, and a model can be validated against dependencies and best practices rules. Finally, DDL can be generated and, given the appropriate authorizations, and can be executed from the Design Studio.

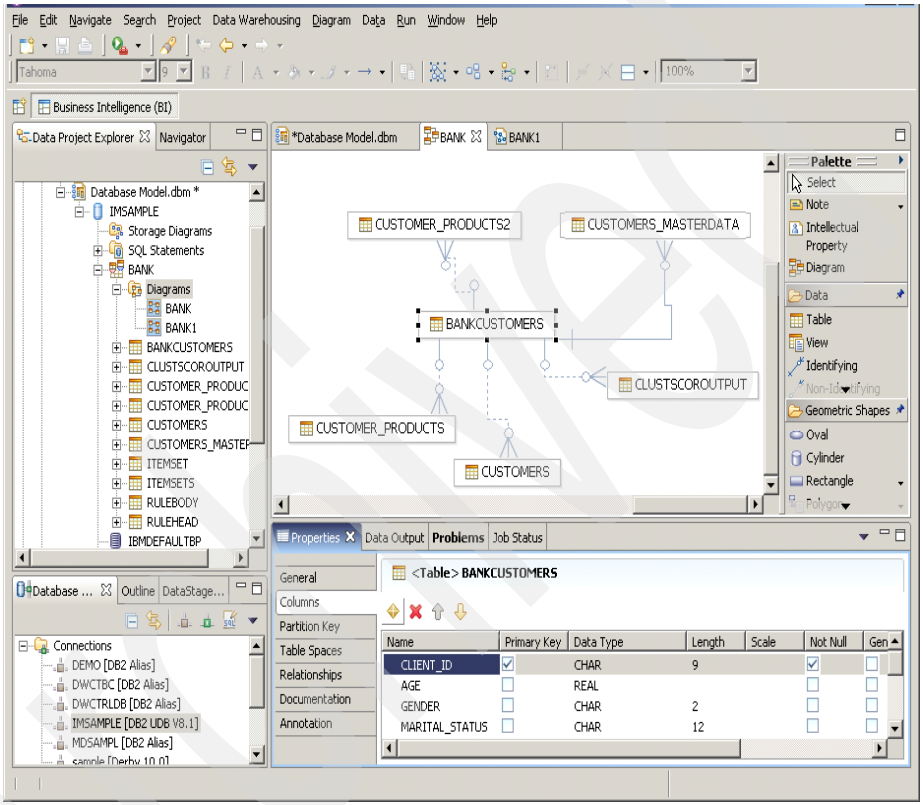


Figure 7-6 Working with a physical data model

The physical data modeling component of InfoSphere Warehouse on System z is essentially the same as the data modeling component of the LUW versions of InfoSphere Warehouse.

7.1.4 SQL Warehousing Tool

The SQL Warehousing Tool is the component of InfoSphere Warehouse on System z that is used to develop DB2-based data movement and transformation flows. DB2-based means that the vast majority of the code used to move and

transform the data is generated by DB2 SQL and utilities and is commonly classified as an *Extract, Load, and Transform* (ELT) tool.

SQW data flows can be used to load data into a data warehouse from most common relational databases that are supported by the Data Source Explorer, and flat files. Data is transformed by the provided transformation operators and utilizing DB2 column functions. Custom logic can also be added by using DB2 user-defined functions and stored procedures.

Control flows are used to sequence the execution of data flows and integrate SQL-based data flows with other types of external processing, such as running JCL jobs, performing ftps, adding control logic, sending notifications and such.

Data flows and control flows are developed using a graphical flow model editor by dropping operators onto the canvas, defining property pages to define the task parameters for the operator and connecting operators together into flows. Flows can be tested and debugged without leaving the Design Studio. After the flows are tested, they can be included in a data warehouse applications and deployment packages created. Then, using the Administration Console, packages are deployed to the runtime environment and then can be manually executed or can be executed using a schedule.

Data flows

Data flows generate SQL instructions that are optimized for DB2 for z/OS, to load, move and transform data within the data warehouse. This can be an ELT scenario that loads or updates the tables in the data warehouse, flows for building relational data marts with the data warehouse, prepares data for OLAP, extracts data for other downstream processing, and any other task that can be accomplished using SQL or DB2 utilities.

The flows are developed using a graphical flow model editor, an example of which is Figure 7-7 on page 227. Data flow operators are used to model the logic needed to accomplish a given task, such as moving data from one or more tables or files, doing some processing of the data, and putting it into a target table or file. The data flow in Figure 7-7 on page 227 takes values from two tables, removes duplicates, unions the data together and, finally, exports the data to a file.

There are data flow operators that represent a number of ways to obtain data. There is a simple local or remote table source operator, an operator to source data from a customized SQL statement and an operator to import from a data set.

There are operators that represent SQL types of transformation activity, such as sorting (Order By), aggregating (Group By), de-duplicating (Distinct), filtering

(Where Condition), joining, and unioning data. The Select List operator can subset columns, apply column transformations, and add derived columns. Any DB2 column function or customized user-defined function can be applied using the Select List operator. In addition, many of the other operators have a Select List property page that can perform all functions of the Select List operator. The Sequence operator can be used to provide new surrogate keys from a DB2 sequence. In addition to utilizing user-defined column functions in the Select List, the custom logic can be provided using the Custom SQL operator or the Table Function Operator.

There is a set of operators that perform some common tasks required when building a data warehouse, such as pivoting (Pivot) and unpivoting (Unpivot) data, splitting (Splitter) data, and looking up (Key Lookup) and replacing keys (Fact Key Replace).

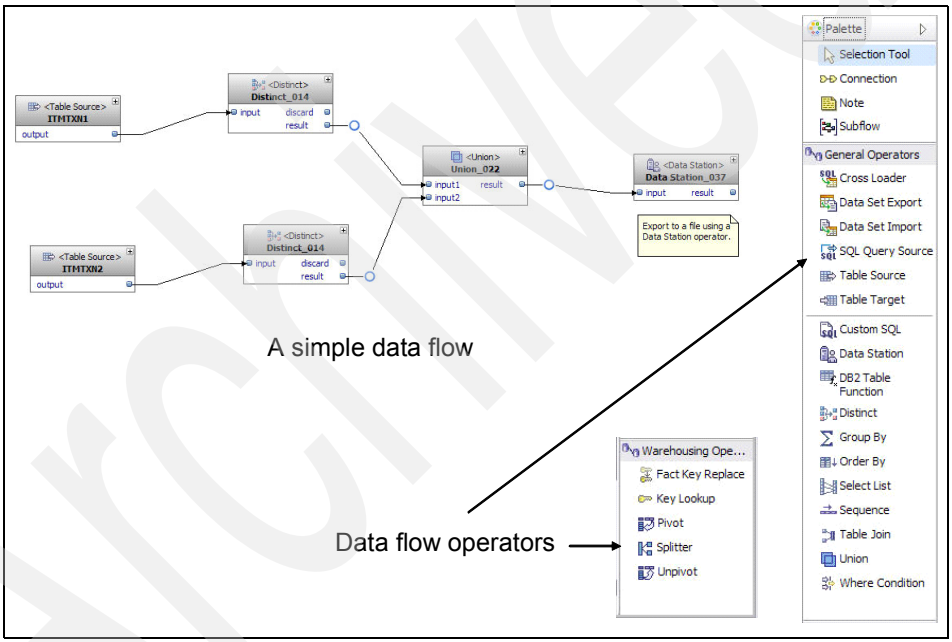


Figure 7-7 Data flow

Finally, there are the target operators. The Table Target operator can update a local or remote table using an insert, update, or delete strategy. The Cross Loader operator utilizes the DB2 load from cursor utility while data can be written to a file using the Data Set Export operator.

One important thing to remember about data flows is that you are not directly writing SQL code, despite the names of the operators. The Design Studio generates and optimizes the SQL. In some cases, one SQL statement might be

generated although in other cases two or more SQL statements are generated, and the code generator handles any temporary staging that is needed. Example 7-1 shows the generated code for the data flow in Figure 7-7 on page 227. While the graphical flow seems fairly simple and is easy to develop, the underlying SQL code to accomplish the task can be somewhat complicated.

Example 7-1 Generated code for data flow in Figure 7-7 on page 227

```
-- <ScriptOptions statementTerminator="@" />

CREATE TABLE IWTEMP3119("MKT_BSKT_TXN_ID" INTEGER, "PD_ID" INTEGER,
"ITM_TXN_TMS" TIMESTAMP, "ITM_TXN_TP_ID" SMALLINT, "ITM_TXN_SCAN_TP_ID"
SMALLINT, "NBR_ITM" SMALLINT, "ITM_STM_PRC" DECIMAL(14,2), "ITM_TXN_AMT"
DECIMAL(14,2), "VAL_DIF_RSN_TP_ID" SMALLINT, "LCL_TAX_AMT" DECIMAL(14,2),
"NAT_TAX_AMT" DECIMAL(14,2), "POS_ID" INTEGER, "SRC_STM_ID" SMALLINT)@

INSERT INTO IWTEMP3119 (MKT_BSKT_TXN_ID, PD_ID, ITM_TXN_TMS, ITM_TXN_TP_ID,
  ITM_TXN_SCAN_TP_ID, NBR_ITM, ITM_STM_PRC, ITM_TXN_AMT, VAL_DIF_RSN_TP_ID,
  LCL_TAX_AMT, NAT_TAX_AMT, POS_ID, SRC_STM_ID)
  SELECT MKT_BSKT_TXN_ID AS MKT_BSKT_TXN_ID, PD_ID AS PD_ID, ITM_TXN_TMS AS
ITM_TXN_TMS,
  ITM_TXN_TP_ID AS ITM_TXN_TP_ID, ITM_TXN_SCAN_TP_ID AS ITM_TXN_SCAN_TP_ID,
  NBR_ITM AS NBR_ITM, ITM_STM_PRC AS ITM_STM_PRC, ITM_TXN_AMT AS ITM_TXN_AMT,
  VAL_DIF_RSN_TP_ID AS VAL_DIF_RSN_TP_ID, LCL_TAX_AMT AS LCL_TAX_AMT,
  NAT_TAX_AMT AS NAT_TAX_AMT, POS_ID AS POS_ID, SRC_STM_ID AS SRC_STM_ID
    FROM
         (SELECT MKT_BSKT_TXN_ID AS MKT_BSKT_TXN_ID, PD_ID AS PD_ID,
ITM_TXN_TMS AS ITM_TXN_TMS,
  ITM_TXN_TP_ID AS ITM_TXN_TP_ID, ITM_TXN_SCAN_TP_ID AS ITM_TXN_SCAN_TP_ID,
  NBR_ITM AS NBR_ITM, ITM_STM_PRC AS ITM_STM_PRC, ITM_TXN_AMT AS ITM_TXN_AMT,
  VAL_DIF_RSN_TP_ID AS VAL_DIF_RSN_TP_ID, LCL_TAX_AMT AS LCL_TAX_AMT,
  NAT_TAX_AMT AS NAT_TAX_AMT, POS_ID AS POS_ID, SRC_STM_ID AS SRC_STM_ID
           FROM (SELECT MKT_BSKT_TXN_ID AS MKT_BSKT_TXN_ID, PD_ID AS PD_ID,
ITM_TXN_TMS AS ITM_TXN_TMS,
  ITM_TXN_TP_ID AS ITM_TXN_TP_ID, ITM_TXN_SCAN_TP_ID AS ITM_TXN_SCAN_TP_ID,
  NBR_ITM AS NBR_ITM, ITM_STM_PRC AS ITM_STM_PRC, ITM_TXN_AMT AS ITM_TXN_AMT,
  VAL_DIF_RSN_TP_ID AS VAL_DIF_RSN_TP_ID, LCL_TAX_AMT AS LCL_TAX_AMT,
  NAT_TAX_AMT AS NAT_TAX_AMT, POS_ID AS POS_ID, SRC_STM_ID AS SRC_STM_ID
                   FROM
                        (
                      SELECT MKT_BSKT_TXN_ID AS MKT_BSKT_TXN_ID,
                        PD_ID AS PD_ID, ITM_TXN_TMS AS ITM_TXN_TMS,
                       ITM_TXN_TP_ID AS ITM_TXN_TP_ID, ITM_TXN_SCAN_TP_ID AS
ITM_TXN_SCAN_TP_ID,
                        NBR_ITM AS NBR_ITM, ITM_STM_PRC AS ITM_STM_PRC,
                        ITM_TXN_AMT AS ITM_TXN_AMT, VAL_DIF_RSN_TP_ID AS
VAL_DIF_RSN_TP_ID,
                        LCL_TAX_AMT AS LCL_TAX_AMT, NAT_TAX_AMT AS
NAT_TAX_AMT,
```

```
                        POS_ID AS POS_ID, SRC_STM_ID AS SRC_STM_ID
                        FROM STAGE.ITMTXN1 AS Q357
                        WHERE NOT (EXISTS (SELECT 1
                                           FROM STAGE.ITMTXN1 AS Q1
                                           WHERE (1 = CASE
                                        WHEN (Q357.ITM_TXN_TP_ID <
ITM_TXN_TP_ID) THEN -(1)
                                        WHEN (Q357.ITM_TXN_TP_ID >
ITM_TXN_TP_ID) THEN 1
                                        WHEN (Q357.ITM_TXN_SCAN_TP_ID <
ITM_TXN_SCAN_TP_ID) THEN -(1)
                                        WHEN (Q357.ITM_TXN_SCAN_TP_ID >
ITM_TXN_SCAN_TP_ID) THEN 1
                                        WHEN (Q357.NBR_ITM < NBR_ITM) THEN
-(1)
                                        WHEN (Q357.NBR_ITM > NBR_ITM) THEN 1
                                        WHEN (Q357.ITM_STM_PRC <
ITM_STM_PRC) THEN -(1)
                                        WHEN (Q357.ITM_STM_PRC >
ITM_STM_PRC) THEN 1
                                        WHEN (Q357.ITM_TXN_AMT <
ITM_TXN_AMT) THEN -(1)
                                        WHEN (Q357.ITM_TXN_AMT >
ITM_TXN_AMT) THEN 1
                                        WHEN (Q357.VAL_DIF_RSN_TP_ID <
VAL_DIF_RSN_TP_ID) THEN -(1)
                                        WHEN (Q357.VAL_DIF_RSN_TP_ID >
VAL_DIF_RSN_TP_ID) THEN 1
                                        WHEN (Q357.LCL_TAX_AMT <
LCL_TAX_AMT) THEN -(1)
                                        WHEN (Q357.LCL_TAX_AMT >
LCL_TAX_AMT) THEN 1
                                        WHEN (Q357.NAT_TAX_AMT <
NAT_TAX_AMT) THEN -(1)
                                        WHEN (Q357.NAT_TAX_AMT >
NAT_TAX_AMT) THEN 1
                                        WHEN (Q357.POS_ID < POS_ID) THEN
-(1)
                                        WHEN (Q357.POS_ID > POS_ID) THEN 1
                                        WHEN (Q357.SRC_STM_ID < SRC_STM_ID)
THEN -(1)
                                        WHEN (Q357.SRC_STM_ID > SRC_STM_ID)
THEN 1
                                         ELSE 0
                                       END) AND (Q357.ITM_TXN_TMS =
ITM_TXN_TMS)
                                       AND (Q357.MKT_BSKT_TXN_ID =
MKT_BSKT_TXN_ID) AND (Q357.PD_ID = PD_ID)))) AS Q732
                  GROUP BY MKT_BSKT_TXN_ID, PD_ID, ITM_TXN_TMS,
```

```
                    ITM_TXN_TP_ID, ITM_TXN_SCAN_TP_ID, NBR_ITM,
                    ITM_STM_PRC, ITM_TXN_AMT, VAL_DIF_RSN_TP_ID,
                    LCL_TAX_AMT, NAT_TAX_AMT, POS_ID, SRC_STM_ID) AS Q1999
         UNION
         SELECT MKT_BSKT_TXN_ID AS MKT_BSKT_TXN_ID, PD_ID AS PD_ID,
ITM_TXN_TMS AS ITM_TXN_TMS,
  ITM_TXN_TP_ID AS ITM_TXN_TP_ID, ITM_TXN_SCAN_TP_ID AS ITM_TXN_SCAN_TP_ID,
  NBR_ITM AS NBR_ITM, ITM_STM_PRC AS ITM_STM_PRC, ITM_TXN_AMT AS ITM_TXN_AMT,
  VAL_DIF_RSN_TP_ID AS VAL_DIF_RSN_TP_ID, LCL_TAX_AMT AS LCL_TAX_AMT,
  NAT_TAX_AMT AS NAT_TAX_AMT, POS_ID AS POS_ID, SRC_STM_ID AS SRC_STM_ID
           FROM (SELECT MKT_BSKT_TXN_ID AS MKT_BSKT_TXN_ID, PD_ID AS PD_ID,
ITM_TXN_TMS AS ITM_TXN_TMS,
  ITM_TXN_TP_ID AS ITM_TXN_TP_ID, ITM_TXN_SCAN_TP_ID AS ITM_TXN_SCAN_TP_ID,
  NBR_ITM AS NBR_ITM, ITM_STM_PRC AS ITM_STM_PRC, ITM_TXN_AMT AS ITM_TXN_AMT,
  VAL_DIF_RSN_TP_ID AS VAL_DIF_RSN_TP_ID, LCL_TAX_AMT AS LCL_TAX_AMT,
  NAT_TAX_AMT AS NAT_TAX_AMT, POS_ID AS POS_ID, SRC_STM_ID AS SRC_STM_ID
                  FROM
                      (
                     SELECT MKT_BSKT_TXN_ID AS MKT_BSKT_TXN_ID,
                      PD_ID AS PD_ID, ITM_TXN_TMS AS ITM_TXN_TMS,
                      ITM_TXN_TP_ID AS ITM_TXN_TP_ID, ITM_TXN_SCAN_TP_ID AS
ITM_TXN_SCAN_TP_ID,
                      NBR_ITM AS NBR_ITM, ITM_STM_PRC AS ITM_STM_PRC,
                      ITM_TXN_AMT AS ITM_TXN_AMT, VAL_DIF_RSN_TP_ID AS
VAL_DIF_RSN_TP_ID,
                      LCL_TAX_AMT AS LCL_TAX_AMT, NAT_TAX_AMT AS
NAT_TAX_AMT,
                      POS_ID AS POS_ID, SRC_STM_ID AS SRC_STM_ID
                      FROM STAGE.ITMTXN2 AS Q1277
                      WHERE NOT (EXISTS (SELECT 1
                                         FROM STAGE.ITMTXN2 AS Q125
                                         WHERE (1 = CASE
                                    WHEN (Q1277.ITM_TXN_TP_ID <
ITM_TXN_TP_ID) THEN -(1)
                                    WHEN (Q1277.ITM_TXN_TP_ID >
ITM_TXN_TP_ID) THEN 1
                                    WHEN (Q1277.ITM_TXN_SCAN_TP_ID <
ITM_TXN_SCAN_TP_ID) THEN -(1)
                                    WHEN (Q1277.ITM_TXN_SCAN_TP_ID >
ITM_TXN_SCAN_TP_ID) THEN 1
                                    WHEN (Q1277.NBR_ITM < NBR_ITM) THEN
-(1)
                                    WHEN (Q1277.NBR_ITM > NBR_ITM) THEN
1
                                    WHEN (Q1277.ITM_STM_PRC <
ITM_STM_PRC) THEN -(1)
                                    WHEN (Q1277.ITM_STM_PRC >
ITM_STM_PRC) THEN 1
```

```
                                                    WHEN (Q1277.ITM_TXN_AMT <
ITM_TXN_AMT) THEN -(1)
                                                    WHEN (Q1277.ITM_TXN_AMT >
ITM_TXN_AMT) THEN 1
                                                    WHEN (Q1277.VAL_DIF_RSN_TP_ID <
VAL_DIF_RSN_TP_ID) THEN -(1)
                                                    WHEN (Q1277.VAL_DIF_RSN_TP_ID >
VAL_DIF_RSN_TP_ID) THEN 1
                                                    WHEN (Q1277.LCL_TAX_AMT <
LCL_TAX_AMT) THEN -(1)
                                                    WHEN (Q1277.LCL_TAX_AMT >
LCL_TAX_AMT) THEN 1
                                                    WHEN (Q1277.NAT_TAX_AMT <
NAT_TAX_AMT) THEN -(1)
                                                    WHEN (Q1277.NAT_TAX_AMT >
NAT_TAX_AMT) THEN 1
                                                    WHEN (Q1277.POS_ID < POS_ID) THEN
-(1)
                                                    WHEN (Q1277.POS_ID > POS_ID) THEN 1
                                                   WHEN (Q1277.SRC_STM_ID < SRC_STM_ID)
THEN -(1)
                                                   WHEN (Q1277.SRC_STM_ID > SRC_STM_ID)
THEN 1
                                                    ELSE 0
                                                  END) AND (Q1277.ITM_TXN_TMS =
ITM_TXN_TMS)
                                                  AND (Q1277.MKT_BSKT_TXN_ID =
MKT_BSKT_TXN_ID) AND (Q1277.PD_ID = PD_ID)))) AS Q1537
                    GROUP BY MKT_BSKT_TXN_ID, PD_ID, ITM_TXN_TMS,
                      ITM_TXN_TP_ID, ITM_TXN_SCAN_TP_ID, NBR_ITM,
                      ITM_STM_PRC, ITM_TXN_AMT, VAL_DIF_RSN_TP_ID,
                      LCL_TAX_AMT, NAT_TAX_AMT, POS_ID, SRC_STM_ID) AS Q2026)
AS Q3120@

---- BEGIN NON-SQL CODE ----

-- JAVA

-- Java Runtime Unit Class:
com.ibm.datatools.sqw.zutil.runtime.ZUtilityRuntimeUnit

-- Utility Type = UNLOAD
--
--Utility Statement = TEMPLATE UNLDDS DSN('ANDYP.DWESAMP.ITMTXN.TEMP')
DISP(NEW,CATLG,CATLG)
--UNLOAD DATA FROM TABLE IWTEMP3119 (MKT_BSKT_TXN_ID INTEGER, PD_ID INTEGER,
ITM_TXN_TMS TIMESTAMP EXTERNAL, ITM_TXN_TP_ID SMALLINT, ITM_TXN_SCAN_TP_ID
SMALLINT, NBR_ITM SMALLINT, ITM_STM_PRC DECIMAL(14,2), ITM_TXN_AMT
```

```
DECIMAL(14,2), VAL_DIF_RSN_TP_ID SMALLINT, LCL_TAX_AMT DECIMAL(14,2),
NAT_TAX_AMT DECIMAL(14,2), POS_ID INTEGER, SRC_STM_ID SMALLINT) UNLDDN UNLDDS
--
--Database Connection = dwesamp
--
--Number of Database Connection(s) = 1
--
--Database Connection 0 = dwesamp
--
--

---- END NON-SQL CODE ----

SELECT MKT_BSKT_TXN_ID AS MKT_BSKT_TXN_ID, PD_ID AS PD_ID, ITM_TXN_TMS AS
ITM_TXN_TMS,
  ITM_TXN_TP_ID AS ITM_TXN_TP_ID, ITM_TXN_SCAN_TP_ID AS ITM_TXN_SCAN_TP_ID,
  NBR_ITM AS NBR_ITM, ITM_STM_PRC AS ITM_STM_PRC, ITM_TXN_AMT AS ITM_TXN_AMT,
  VAL_DIF_RSN_TP_ID AS VAL_DIF_RSN_TP_ID, LCL_TAX_AMT AS LCL_TAX_AMT,
  NAT_TAX_AMT AS NAT_TAX_AMT, POS_ID AS POS_ID, SRC_STM_ID AS SRC_STM_ID
  FROM (SELECT MKT_BSKT_TXN_ID AS MKT_BSKT_TXN_ID, PD_ID AS PD_ID, ITM_TXN_TMS
AS ITM_TXN_TMS,
  ITM_TXN_TP_ID AS ITM_TXN_TP_ID, ITM_TXN_SCAN_TP_ID AS ITM_TXN_SCAN_TP_ID,
  NBR_ITM AS NBR_ITM, ITM_STM_PRC AS ITM_STM_PRC, ITM_TXN_AMT AS ITM_TXN_AMT,
  VAL_DIF_RSN_TP_ID AS VAL_DIF_RSN_TP_ID, LCL_TAX_AMT AS LCL_TAX_AMT,
  NAT_TAX_AMT AS NAT_TAX_AMT, POS_ID AS POS_ID, SRC_STM_ID AS SRC_STM_ID
         FROM IWTEMP3119 AS Q2531) AS Q2573@

DROP TABLE IWTEMP3119@
```

Control flows

While data flows are about moving data, control flows are about sequencing and integration. Each operator in a control flow represents a flow control point or a distinct task to be accomplished. Control flows are the units of deployment and execution in the runtime environment. When defining a data warehouse data movement application, it is one or more control flows that make up the application. In the runtime environment, control flows are what is scheduled for execution.

There are over twenty different kinds of tasks that can be used in a control flow, as shown in Figure 7-8 on page 233. There are operators for data flows, for integrating DataStage jobs and sequences, performing ftp jobs, running JCL jobs, Linux commands, calling DB2 stored procedures and scripts, and running a variety of DB2 utilities.

There are a number of operators to control the sequence of execution of tasks. Each operator has On Success, On Failure, and Always ports that allow different

execution paths to be taken. In addition, different paths can be taken depending on the value of a variable. There are also operators that allow iterative loops to be developed. The Parallel Container executes the contained tasks in parallel, taking the On Success path if all tasks complete successfully.

Figure 7-8 shows a control flow that submits a DataStage job sequence to a DataStage server and waits for completion of that job. After successful completion, a parallel container executes three data flows to load dimension tables. After all three data flows complete successfully, then another data flow loads the fact table. Then a DB2 Stored Procedure is called to perform some custom validation and then returns a value in a variable that is checked using a variable comparison operator to determine if the validation succeeded. If the validation succeeds, the DB2 runstats utility is called to update the catalog statistics. At two critical points in the flow are operators connected to the On Failure port. If the operator does not complete successfully, the On Failure path is taken, and, in this example, sends an e-mail to notify the appropriate staff that the flow failed. If any of the other operators fail, the control flow just fails and at the end, the On Failure port of the Start operator is taken, which logs the failure to an application log.

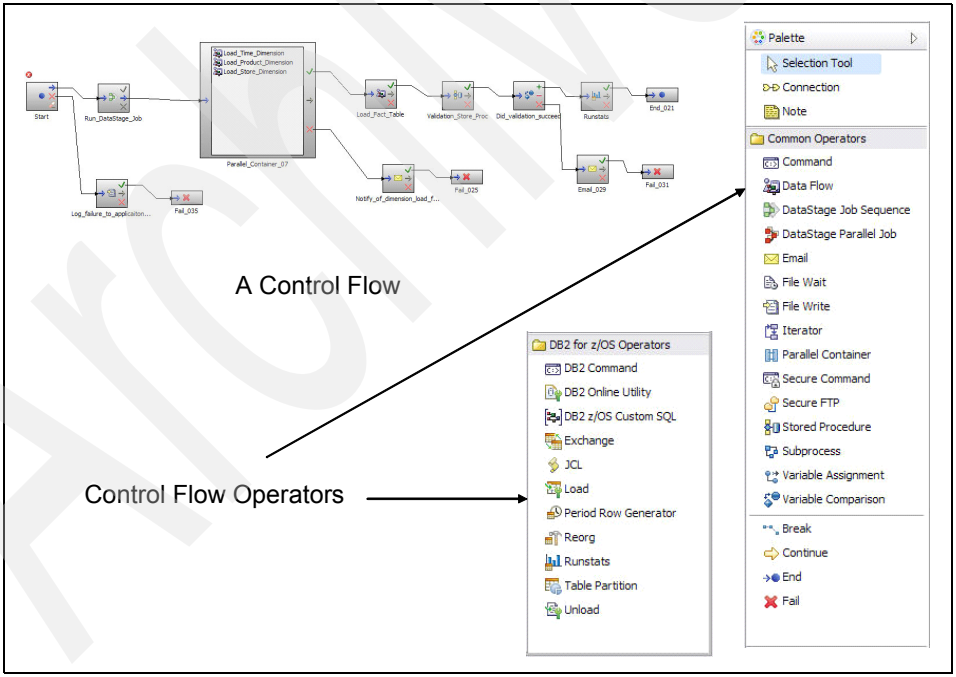


Figure 7-8 Control flow

7.1.5 Cubing Services

Cubing Services is a server for providing mutidimensional data to client tools. It is essentially a cube cache that sits between DB2 for z/OS and the client tools, as shown in Figure 7-9 on page 235. It dynamically builds one or more in-memory cubes as user requests are serviced. The cube is never persisted on disk and therefore does not require data to be copied outside of the data warehouse into another type of storage, hence the term *No Copy Analytics*.

When an MDX query is received from a client tool, the cube server first tries to fulfill the query from the in-memory cube cache. If the data is in the cache the data is obtained and the result returned. If some or all of the data is not in the cube, then the Cube Server formulates SQL to retrieve the needed data from DB2 and extends the in-memory cube before returning the result. After the data is in the cache, the next request is satisfied directly from the in-memory cube.

When the cube server must get the data from the database, a large majority of the requested data is at a higher aggregation level than the tables in the database. That means that the DB2 optimizer has an opportunity to satisfy the query from summarized data that might be in materialized query tables (MQTs), in which case the result set can be obtained more quickly than going against the detail data in the base tables. This query routing is handled by the DB2 optimizer and is transparent to the cube server.

Well how do these MQTs get created? Certainly they can be created manually. However, the ad-hoc nature of OLAP makes it difficult to predict the query workload. However, Cubing Services provided help in the form of an Optimization Advisor. The Optimization Advisor analyzes the OLAP model, looking at all of the possible slices of data and optionally samples the base tables to generate a recommendation for MQTs and output it in the form of an SQL script which can then be used to build the MQTs. Then, as the cube server needs data from the DB2 tables to populate the cube cache, the DB2 optimizer has a great chance to satisfy the request from the MQTs and thereby dramatically increase the overall performance. Then over time, if there are any gaps in performance, additional MQTs can be created to help further improve the performance.

Cubing Service on System z is essentially the same as in the LUW version, and so the techniques presented in the rest of this book apply. The System z version differences are primarily related to the differences in the underlying DB2 platforms, which we discuss in the next section.

7.2 Cubing Services on System z

As on LUW platforms, Cubing Services on System z is a multidimensional database server. It takes requests in the form of multidimensional expressions (MDX), obtains the data from a cube, and returns the results. The architecture of Cubing Services on System z is similar to Cubing Services on LUW, as shown in Figure 7-9. On System z, the cube server runs in the Linux partition and the back-end database is DB2 for z/OS V8 or V9.

Cubing Services on System z is primarily the same as Cubing Services on LUW. The cube modeling techniques and best practices that are documented in this book, and in the existing publication, *InfoSphere Warehouse: Cubing Services and Client Access Interfaces*, SG24-7582, generally apply to Cubing Services on System z. However, there are some differences, which we discuss in subsequent sections of this chapter.

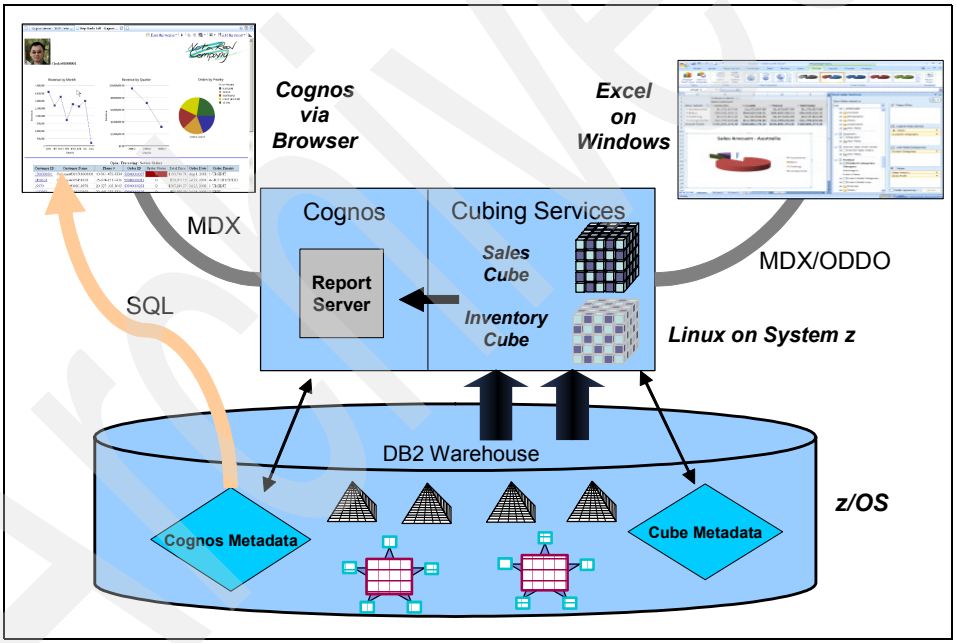


Figure 7-9 Architecture of Cubing Service on System z

7.2.1 Differences in Cubing Services on System z

There are two primary differences between Cubing Services on LUW and on the System z platforms due to the differences in the DB2 for z/OS optimizer and the MQT support of DB2 for z/OS.

The DB2 for z/OS optimizer does not eliminate lossless joins; therefore, the Cube Server SQL generator must generate different SQL for DB2 for z/OS, as shown in Figure 7-10. This is handled by the SQL generation routines of the Cube Server and the Optimization Advisor and is transparent to the users of Cubing Services.

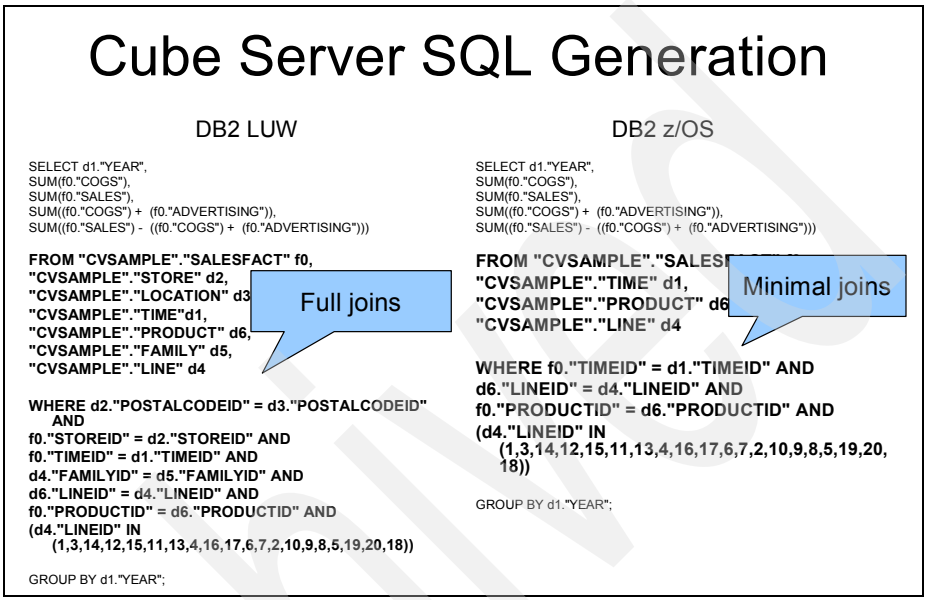


Figure 7-10 Differences in SQL generation

The DB2 optimizer can use an MQT for a distributive measure if the query matches the MQT slice or the one above it, while a non-distributive measure must match the MQT slice exactly. As documented in Figure 7-11 on page 237, all but Distinct measures are distributive in DB2 for z/OS, which allows more queries to be routed to MQTs.

Incremental refresh for MQTs is not supported on DB2 for z/OS, and therefore as the third column of Figure 7-11 on page 237 indicates, measures on z/OS can not be incrementally maintained. MQTs must be maintained with a full refresh for system maintained MQTs or by the user for user maintained MQTs.

Measures – DB2 LUW Support

Measure	Distributive	Non-distributive	Inc-maintainable
Count	x		x
Sum	x		x
Max	x		
Avg		x	
Stddev		x	
Distinct		x	

Measures – DB2 Z Support

Measure	Distributive	Non-distributive	Inc-maintainable
Count	x		N/A
Sum	x		N/A
Max	x		N/A
Avg	x		N/A
Stddev	x		N/A
Distinct		x	N/A

Figure 7-11 Distributive versus non-distributive measures

The Administration Console for InfoSphere Warehouser for System z was re-written using Adobe Flex and is shown in Figure 7-12 on page 238. For Cubing Services, the Administration Console is used to:

- Manage Cube Servers: Create, start, stop cube servers and cubes, and assign cubes to cube servers.
- Manage OLAP Metadata: Import OLAP metadata from XML files, map cubes to data sources, and assign cubes to cube servers.
- Manage Cubing Services Roles: Create user-defined roles and assign users to those roles.

While the user interface is new, the functions for Cubing Services is basically the same as in the LUW version, with the exception of two functions that did not make it into this release. The Cubing Services Optimization Advisor cannot be invoked from the Administration Console nor can SQL scripts be executed from the Administration Console. At this time, the Optimization Advisor must be invoked from the Design Studio and might require the runtime administrator to install the Design Studio client. The function to run SQL scripts from the Administration Console was primarily there to execute the scripts generated by the Optimization Advisor. These scripts can be run from other tools, such as the DB2 command line, QMF, or SPUFII.

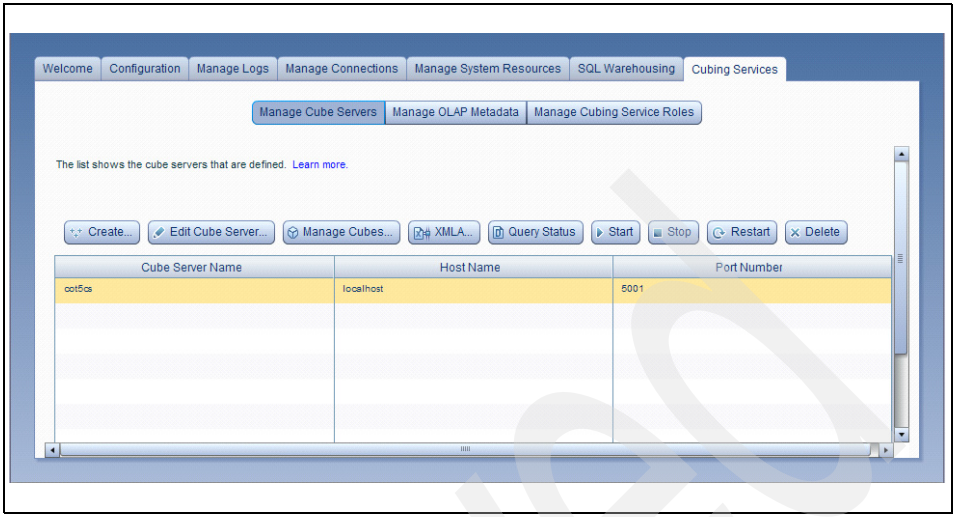


Figure 7-12 Cubing Services administrative services

7.2.2 Best Practices

The best practices and design techniques documented in the other parts of this book, and in *InfoSphere Warehouse: Cubing Services and Client Access Interfaces*, SG24-7582, generally apply to Cubing Services on System z. However, there are some settings, database design, and troubleshooting recommendations that are specific to System z.

In general, the best performance experience is when the MDX query can be serviced from the cube server cache. If the query cannot be satisfied from the cache, then the cube server generates SQL to retrieve the data from DB2 to add to the cache. At cube start, and after specifically clearing the data cache, most of the MDX requests result in SQL queries to DB2 until such time that the cache is adequately populated such that it satisfies most requests from the cache. Therefore the performance of queries to DB2 affect the overall performance of the cube server.

The primary tuning knob for cube server is to ensure that the appropriate MQTs exist to support the cube. The recommendations of the Cubing Services Optimization Advisor is the starting point for overall cube performance. In addition, consider the recommendations in the next sections.

Recommended DB2 ZPARM settings

CDSSRDEF enables query parallelism. Partitioning any large fact tables and setting the degrees of parallelism helps to reduce the elapsed time when the base tables must be accessed.

Enable star joins by setting STARJOIN and, depending on workload, adjust the star join enablement threshold from the default of 10 by using SJTABLES.

The routing of queries to MQTs is key to overall Cubing Services performance. The parms REFSHAGE and MAINTYPE must be set to allow automatic query rewrite to be considered by the DB2 optimizer.

Data warehouse database design recommendations

In this section, we provide a number of design recommendations for the data warehouse database.

Adjusting the MQT definitions

The MQT recommendations suggested by the Cubing Services Optimization Advisor might not cover all possible slices, with a result that certain queries might not be routed to an MQT. If this happens to be a path that is common for the users, additional MQTs might need to be created to improve the performance of the cube.

Using separate buffer pools for tables and indexes

In a star schema database where the fact table size is usually large, it is recommended to place the data and indexes for the fact table into separate buffer pools, which improves the query elapsed time by increasing the buffer pool hit ratio, which also has the advantage of making it easier to use the DB2 statistics and accounting traces to identify performance bottlenecks and to perform diagnostics.

Adjusting work file buffer pools and data sets

If the query issued by the Cube server must access the base fact table, it can result in intensive sort activity. To improve sort performance, put the work files database, DSNDB07, into separate buffer pools, and increase the size of the buffer pools as system resources allow. In addition, to reduce file contention for parallel sorting, additional work file data sets might be required.

Compressing data and indexes

While the usual impetus for compression is to decrease the cost of storage, an increase in performance might also be obtained. This is due to the significant reduction of I/O suspension time with minimal CPU overhead. Decoding of compressed data for reading is much less costly than compressing the data when writing data. In a data warehouse, most access to the databases is for

reading with a relatively smaller amount of updates that reduce the comparative of data compression.

Partitioning data

Using classic partitioned table spaces for large tables and using data partitioned secondary indexes (DPSI) improve SQL query response time.

Troubleshooting cube performance

Despite best efforts, there might be occasions that certain paths through the data consistently perform poorly. In these cases, it might be useful to follow this methodology to troubleshoot the SQL performance issues:

1. Use the Administration Console to turn on the SQL trace and, optionally, the MDX trace for this cube server.
2. Execute the workload that is having performance issues to capture the MDX SQL statements in the SQL log file, SQLLog.txt, which is found in <installation directory>/CubingServices/<cube server name>/Logs.
3. Extract the SQL statements from the SQL trace log.
4. Run explain on the extracted SQL statements. Check to see if any queries are not matching MQTs and determine which MQTs the queries are using.

   Note: See the additional materials link on the ITSO Web page for this book for a sample Java program that can be used for step 4.

5. Assess whether any of the queries not matching MQTs might benefit by manually creating one or more MQTs.
6. If the extracted workload is enough to represent a typical workload, it is helpful to drop MQTs that are not used, which saves storage and CPU costs for refreshing.
7. For any new MQTs and SQL queries that cannot be helped with an MQT, use Optimization Expert to get recommendations for indexes. Finally, use RUNSTATS to update catalog statistics on any new MQTs. For more information, refer to the IBM Redbooks publications at the following locations:

   http://www.redbooks.ibm.com/abstracts/sg247421.html?Open

   http://www-01.ibm.com/software//data/db2imstools/db2tools/opti-expert-zos/

Glossary

Access control list (ACL). The list of principals that have explicit permission (to publish, to subscribe to, and to request persistent delivery of a publication message) against a topic in the topic tree. The ACLs define the implementation of topic-based security.

Aggregate. Pre-calculated and pre-stored summaries that are kept in the data warehouse to improve query performance.

Aggregation. An attribute-level transformation that reduces the level of detail of available data, for example, having a Total Quantity by Category of Items rather than the individual quantity of each item in the category.

Application programming interface. An interface provided by a software product that enables programs to request services.

Asynchronous messaging. A method of communication between programs in which a program places a message on a message queue and then proceeds with its own processing without waiting for a reply to its message.

Attribute. A field in a dimension table.

BLOB. Binary large object: A block of bytes of data (for example, the body of a message) that has no discernible meaning but is treated as one solid entity that cannot be interpreted.

Commit. An operation that applies all to the changes made during the current unit of recovery or unit of work. After the operation is complete, a new unit of recovery or unit of work begins.

Composite key. A key in a fact table that is the concatenation of the foreign keys in the dimension tables.

Computer. A device that accepts information (in the form of digitalized data) and manipulates it for some result based on a program or sequence of instructions about how the data is to be processed.

Configuration. The collection of brokers, their execution groups, the message flows and sets that are assigned to them, and the topics and associated access control specifications.

Continuous Data Replication. Refer to Enterprise Replication.

DDL (data definition language). An SQL statement that creates or modifies the structure of a table or database, for example, CREATE TABLE, DROP TABLE, ALTER TABLE, or CREATE DATABASE.

DML (data manipulation language). An INSERT, UPDATE, DELETE, or SELECT SQL statement.

Data append. A data loading technique where new data is added to the database, leaving the existing data unaltered.

Data cleansing. A process of data manipulation and transformation to eliminate variations and inconsistencies in data content, which is typically to improve the quality, consistency, and usability of the data.

Data federation. The process of enabling data from multiple heterogeneous data sources to appear as though it is contained in a single relational database. Can also be referred to “distributed access.”

Data mart. An implementation of a data warehouse, typically with a smaller and more tightly restricted scope, such as for a department or workgroup. It can be independent or derived from another data warehouse environment.

Data mining. A mode of data analysis that has a focus on the discovery of new information, such as unknown facts, data relationships, or data patterns.

Data partition. A segment of a database that can be accessed and operated on independently, even though it is part of a larger data structure.

Data refresh. A data loading technique where all the data in a database is completely replaced with a new set of data.

Data warehouse. A specialized data environment developed, structured, and used specifically for decision support and informational applications. It is subject oriented rather than application oriented. Data is integrated, non-volatile, and time variant.

Database partition. Part of a database that consists of its own data, indexes, configuration files, and transaction logs.

DataBlades. These are program modules that provide extended capabilities for Informix® databases and are tightly integrated with the DBMS.

DB Connect. Enables connection to several relational database systems and the transfer of data from these database systems into the SAP Business Information Warehouse.

Debugger. A facility on the Message Flows view in the Control Center that enables message flows to be visually debugged.

Deploy. Make operational the configuration and topology of the broker domain.

Dimension. Data that further qualifies or describes a measure, or both, such as amounts or durations.

Distributed application In message queuing, a set of application programs that can each be connected to a different queue manager, but that collectively constitute a single application.

Drill-down. Iterative analysis, exploring facts at more detailed levels of the dimension hierarchies.

Dynamic SQL. SQL that is interpreted during execution of the statement.

Embedded Database. A database that works exclusively with a single application or appliance.

Engine. A program that performs a core or essential function for other programs. A database engine performs database functions on behalf of the database user programs.

Enrichment. The creation of derived data. An attribute-level transformation performed by some type of algorithm to create one or more new (derived) attributes.

Enterprise Replication. An asynchronous, log-based tool for replicating data between IBM Informix Dynamic Server database servers.

Extenders. These are program modules that provide extended capabilities for DB2 and are tightly integrated with DB2.

FACTS. A collection of measures and the information to interpret those measures in a given context.

Federation. Providing a unified interface to diverse data.

Gateway. A means to access a heterogeneous data source. It can use native access or ODBC technology.

Grain. The fundamental lowest level of data represented in a dimensional fact table.

Instance. A particular realization of a computer process. Relative to the database, the realization of a complete database environment.

Java Database Connectivity. An application programming interface that has the same characteristics as ODBC, but is specifically designed for use by Java database applications.

Java Development Kit. A Software package used to write, compile, debug, and run Java applets and applications.

Java Message Service. An application programming interface that provides Java language functions for handling messages.

Java Runtime Environment. A subset of the Java Development Kit that enables you to run Java applets and applications.

Materialized query table. A table where the results of a query are stored for later reuse.

Measure. A data item that measures the performance or behavior of business processes.

Message domain. The value that determines how the message is interpreted (parsed).

Message flow. A directed graph that represents the set of activities performed on a message or event as it passes through a broker. A message flow consists of a set of message processing nodes and message processing connectors.

Message parser. A program that interprets the bit stream of an incoming message and creates an internal representation of the message in a tree structure. A parser is also responsible for generating a bit stream for an outgoing message from the internal representation.

Metadata. Typically called data (or information) about data. It describes or defines data elements.

MOLAP. Multidimensional OLAP. Can be called MD-OLAP. It is OLAP that uses a multidimensional database as the underlying data structure.

Multidimensional analysis. Analysis of data along several dimensions, for example, analyzing revenue by product, store, and date.

Multidimensional Clustering. A technique that allows for rows of data with similar values across multiple dimensions to be physically clustered together on disk**.**

Multitasking. Operating system capability that allows multiple tasks to run concurrently, taking turns using the resources of the computer.

Multithreading. Operating system capability that enables multiple concurrent users to use the same program, which saves the overhead of initiating the program multiple times.

Nickname. An identifier that is used to reference the object located at the data source that you want to access.

Node group. Group of one or more database partitions.

Node. An instance of a database or database partition.

ODS. (1) Operational data store: A relational table for holding clean data to load into InfoCubes, and can support some query activity. (2) Online Dynamic Server, an older name for IDS.

OLAP. Online analytical processing. Multidimensional data analysis that is performed in real time. Not dependent on an underlying data schema.

Open Database Connectivity. A standard application programming interface for accessing data in both relational and non-relational database management systems. Using this API, database applications can access data stored in database management systems on a variety of computers even if each database management system uses a different data storage format and programming interface. ODBC is based on the call-level interface (CLI) specification of the X/Open SQL Access Group.

Optimization. The capability to enable a process to execute and perform in such a way as to maximize performance, minimize resource utilization, and minimize the process execution response time delivered to the user.

Partition. Part of a database that consists of its own data, indexes, configuration files, and transaction logs.

Pass-through. The act of passing the SQL for an operation directly to the data source without being changed by the federation server.

Pivoting. Analysis operation where a user takes a different viewpoint of the results, for example, by changing the way the dimensions are arranged.

Primary key. Field in a table that is uniquely different for each record in the table.

Process. An instance of a program running in a computer.

Program. A specific set of ordered operations for a computer to perform.

Pushdown. The act of optimizing a data operation by pushing the SQL down to the lowest point in the federated architecture where that operation can be executed. More simply, a pushdown operation is one that is executed at a remote server.

RSAM. Relational Sequential Access Method is the disk access method and storage manager for the Informix DBMS.

ROLAP. Relational OLAP. Multidimensional analysis using a multidimensional view of relational data. A relational database is used as the underlying data structure.

Roll-up. Iterative analysis, exploring facts at a higher level of summarization.

Server. A computer program that provides services to other computer programs (and their users) in the same or other computers. However, the computer that a server program runs in is also frequently referred to as a server.

Shared nothing. A data management architecture where nothing is shared between processes. Each process has its own processor, memory, and disk space.

Static SQL. SQL that was compiled prior to execution. Typically provides best performance.

Subject area. A logical grouping of data by categories, such as customers or items.

Synchronous messaging. A method of communication between programs in which a program places a message on a message queue and then waits for a reply before resuming its own processing.

Task. The basic unit of programming that an operating system controls. Also see Multitasking.

Thread. The placeholder information associated with a single use of a program that can handle multiple concurrent users. Also see Multithreading.

Unit of work. A recoverable sequence of operations performed by an application between two points of consistency.

User mapping. An association made between the federated server user ID and password and the data source (to be accessed) user ID and password.

Virtual database. A federation of multiple heterogeneous relational databases.

Warehouse catalog. A subsystem that stores and manages all the system metadata.

xtree. A query-tree tool that enables you to monitor the query plan execution of individual queries in a graphical environment.

Abbreviations and acronyms

ACS	Access control system
ADK	Archive Development Kit
API	Application programming interface
AQR	Automatic query rewrite
AR	Access register
ARM	Automatic restart manager
ART	Access register translation
ASCII	American Standard Code for Information Interchange
AST	Application summary table
BLOB	Binary large object
BW	Business Information Warehouse (SAP)
CCMS	Computing Center Management System
CDR	Continuous Data Replication
CFG	Configuration
CLI	Call-level interface
CLOB	Character large object
CLP	Command-line processor
CLR	Continuous Log Recovery
CORBA	Common Object Request Broker Architecture
CPU	Central processing unit
CS	Cursor Stability
DAS	DB2 Administration Server
DB	Database
DB2 II	DB2 Information Integrator
DB2 UDB	DB2 Universal Database™
DBA	Database administrator
DBM	Database manager
DBMS	Database management system
DCE	Distributed computing environment
DCM	Dynamic Coserver Management
DCOM	Distributed Component Object Model
DDL	Data definition language
DES	Data Encryption Standard
DIMID	Dimension Identifier
DLL	Dynamic link library
DML	Data manipulation language
DMS	Database managed space
DPF	Data partitioning facility
DRDA®	Distributed Relational Database Architecture™
DSA	Dynamic Scalable Architecture
DSN	Data source name
DSS	Decision support system
EAI	Enterprise Application Integration
EBCDIC	Extended Binary Coded Decimal Interchange Code
EDA	Enterprise data architecture
EDU	Engine dispatchable unit
EGL	Enterprise Generation Language
EGM	Enterprise Gateway Manager
EJB™	Enterprise Java Beans
ER	Enterprise Replication
ERP	Enterprise Resource Planning
ESE	Enterprise Server Edition

ETL	Extract, Transform, and Load
FP	Fix Pack
FTP	File Transfer Protocol
Gb	Gigabits
GB	Gigabytes
GUI	Graphical user interface
HADR	High Availability Disaster Recovery
HDR	High Availability Data Replication
HPL	High Performance Loader
I/O	Input/output
IBM	International Business Machines Corporation
ID	Identifier
IDE	Integrated Development Environment
IDS	Informix Dynamic Server
II	Information Integrator
IMS™	Information Management System
ISA	Informix Server Administrator
ISAM	Indexed Sequential Access Method
ISM	Informix Storage Manager
ISV	Independent software vendor
IT	Information technology
ITR	Internal throughput rate
ITSO	International Technical Support Organization
IX	Index
J2EE™	Java 2 Platform Enterprise Edition
JAR	Java Archive
JDBC	Java Database Connectivity
JDK™	Java Development Kit
JE	Java Edition
JMS	Java Message Service
JRE™	Java Runtime Environment
JVM	Java Virtual Machine
KB	Kilobyte (1024 bytes)
LBAC	Label Based Access Control
LDAP	Lightweight Directory Access Protocol
LPAR	Logical partition
LRU	Least Recently Used
LUN	Logical unit number
LV	Logical volume
Mb	Megabits
MB	Megabytes
MDC	Multidimensional clustering
MPP	Massively parallel processing
MQI	Message queuing interface

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

For information about ordering these publications, see “How to get Redbooks” on page 248. Note that some of the documents referenced here might be available in softcopy only.

- *DB2 Cube Views: A Primer,* SG24-7002.
- *Leveraging DB2 Data Warehouse Edition for Business Intelligence*, SG24-7274.
- *Dimensional Modeling in a Business Intelligence Environment*, SG24-7138.
- *InfoSphere Warehouse: Cubing Services and Client Access Interfaces*, SG24-7582.

Online resources

These Web sites are also relevant as further information sources:

- The strategic importance of OLAP and multidimensional analysis - A Cognos White Paper:

  http://www.cognos.com/pdfs/whitepapers/wp_strategic_importance_of_olap_and_multidimensional_analysis.pdf

- OLAP and Cubing - IBM DB2 9.5 Information Center:

  http://publib.boulder.ibm.com/infocenter/db2luw/v9r5/index.jsp?topic=/com.ibm.dwe.cubeserv.doc/olap_cs_overview.html

- InfoSphere Warehouse - IBM product site:

  http://www-306.ibm.com/software/data/infosphere/warehouse/

How to get Redbooks

You can search for, view, or download Redbooks, Redpapers, Technotes, draft publications and Additional materials, and order hardcopy Redbooks, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Index

D

E

F

G

H

I

S

T

U

V

W

X

Z

Multidimensional Analytics: Delivered with InfoSphere Warehouse Cubing Services

(0.5” spine)
0.475”<->0.875”
250 <-> 459 pages

Multidimensional Analytics: Delivered with InfoSphere Warehouse Cubing Services

Getting more information from your data warehousing environment

Multidimensional analytics for improved decision making

Efficient decisions with no copy analytics

In this IBM Redbooks publication, we discuss and describe a multidimensional data warehousing infrastructure that can enable solutions for complex problems in an efficient and effective manner. The focus of this infrastructure is the InfoSphere Warehouse Cubing Services Feature. With this feature, DB2 becomes the data store for large volumes of data that you can use to perform multidimensional analysis, which enables viewing complex problems from multiple perspectives, which provides more information for management business decision making. This feature supports analytic tool interfaces from powerful data analysis tools, such as Cognos 8 BI, Microsoft Excel, and Alphablox. This is a significant capability that supports and enhances the analytics that clients use as they work to resolve problems with an ever growing scope, dimension, and complexity. Analyzing problems by performing more detailed queries on the data and viewing the results from multiple perspectives yields significantly more information and insight. Building multidimensional cubes based on underlying DB2 relational tables, without having to move or replicate the data, enables significantly more powerful data analysis with less work and leads to faster problem resolution with the capability for more informed management decision making. This capability is known as *No Copy Analytics* and is made possible with InfoSphere Warehouse Cubing Services.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks

SG24-7679-00 ISBN 0738432628