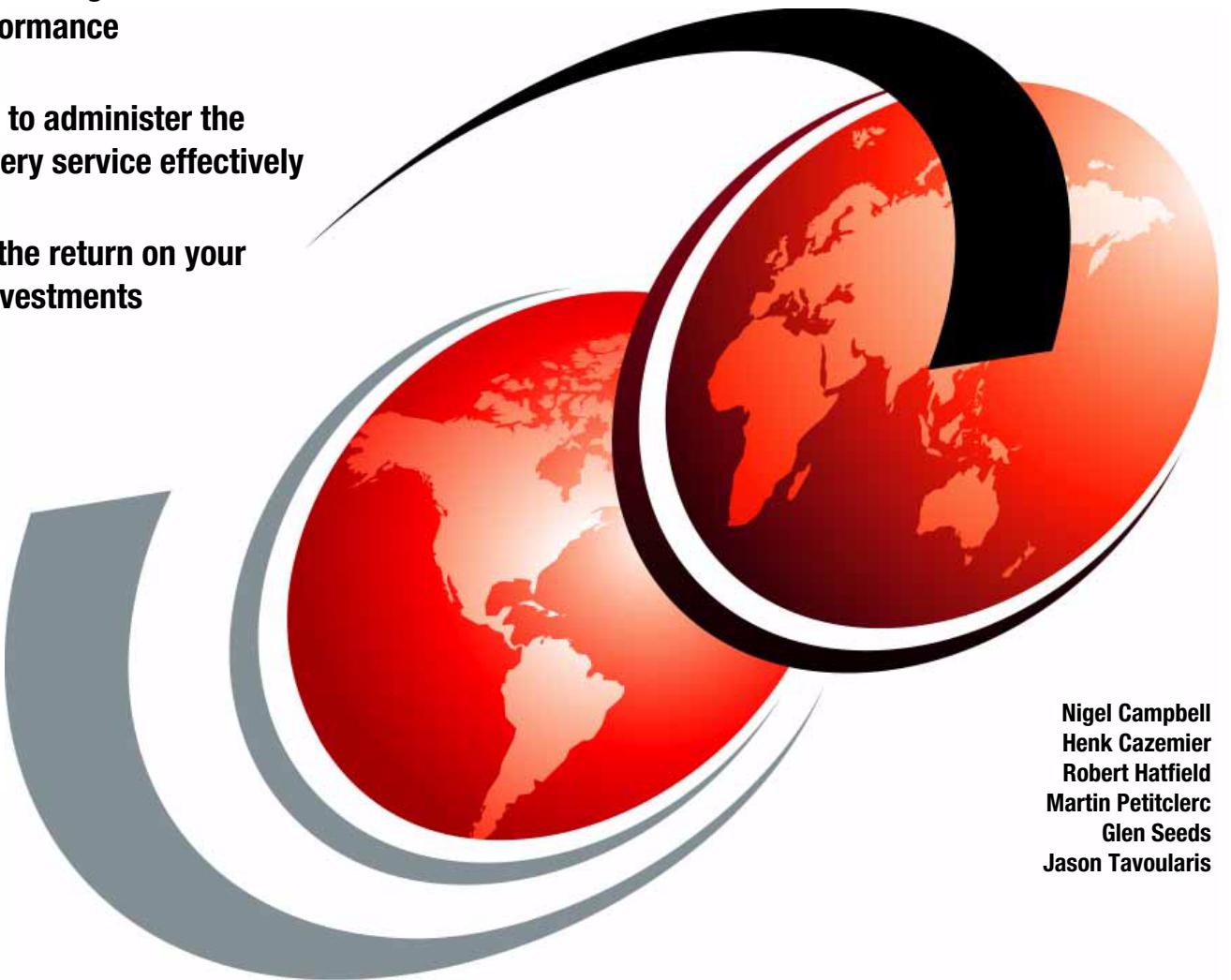


IBM Cognos Dynamic Query

Discover how Cognos accelerates query performance

Learn how to administer the Cognos query service effectively

Maximize the return on your analytic investments



Nigel Campbell
Henk Cazemier
Robert Hatfield
Martin Petitclerc
Glen Seeds
Jason Tavoularis

Redbooks



International Technical Support Organization

IBM Cognos Dynamic Query

September 2013

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

First Edition (September 2013)

This edition applies to Version 10, Release 2, Modification 1 of IBM Cognos Business Intelligence (product number 5724-W12)

© Copyright International Business Machines Corporation 2013. All rights reserved.

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Notices	vii
Trademarks	viii
Preface	ix
Authors	x
Now you can become a published author, too!	xii
Comments welcome	xii
Stay connected to IBM Redbooks	xii
Chapter 1. Overview of Cognos Dynamic Query	1
1.1 Introduction	2
1.2 Background	3
1.3 Architecture	3
1.3.1 Planning and executing the query	5
1.4 Technology selection guidance	6
1.4.1 Pure relational analytics	7
1.4.2 OLAP analytics	7
Chapter 2. Administration	11
2.1 Configuring the query service	12
2.1.1 Memory sizing	12
2.1.2 Throughput sizing	15
2.1.3 Multi-server environments	16
2.2 Data source administration	17
2.2.1 Connection command blocks	17
2.2.2 JDBC drivers	18
2.2.3 OLAP connections	19
2.2.4 ERP and CRM data sources	19
2.3 Cache management	20
2.3.1 Priming the cache	20
2.3.2 Clearing the cache	20
2.3.3 Automating cache operations	21
Chapter 3. Metadata modeling	25
3.1 Cognos Framework Manager	26
3.2 Goals of metadata modeling relational data sources	26
3.2.1 Modeling for self-service analysis	26
3.3 Framework Manager architecture	27
3.4 Key objects of a relational model	28
3.4.1 Query subjects	28
3.4.2 Dimensions	30
3.4.3 Determinants	31
3.4.4 Relationships	33
3.5 Organizing relational models	35
3.5.1 Data view	35
3.5.2 Business logic view	36
3.5.3 Presentation view	36

3.6 Relational modeling for performance	36
3.6.1 As view versus minimized SQL generation.....	36
3.6.2 Security-aware caching.....	39
Chapter 4. Macros	43
4.1 Macros explained	44
4.2 Macro language	45
4.2.1 Operator	45
4.2.2 List separator character	45
4.2.3 Functions	46
4.2.4 Comments.....	46
4.2.5 Simple case construct.....	46
4.3 Parameter maps	47
4.4 Session parameters	49
4.5 Advanced examples	51
4.5.1 Member unique name for next year	51
4.5.2 Turning promptmany result into a rowset	52
4.5.3 Dynamic column drill.....	53
4.5.4 Filtering for internal and external customers.....	55
Chapter 5. Report authoring	57
5.1 Authoring interfaces	58
5.1.1 Cognos Workspace Advanced	58
5.1.2 Cognos Report Studio.....	58
5.2 Processing report executions	59
5.2.1 Local and database processing	59
5.3 Database functions	60
5.4 Dimensional and relational reporting styles	61
5.5 Suppression	62
5.6 Dimensional summaries	63
5.7 Advanced features in Report Studio's Query Explorer	64
5.7.1 Reference queries	64
5.7.2 Union, intersect, and except queries.....	65
5.7.3 Join relationships	66
5.7.4 Master detail relationships	66
Chapter 6. Optimizing SQL for performance	67
6.1 Remember that less is faster	68
6.2 Make use of enforced and non-enforced constraints	68
6.3 Use indexes and table organization features	69
6.4 Review column group statistics.....	69
6.5 Avoid complex join and filter expressions	70
6.5.1 Temporal expressions.....	70
6.5.2 Expressions on table columns in predicates.....	70
6.6 Reduce explicit or implicit conversions	71
6.7 Minimize complexity of conditional query items	71
6.8 Review the order of conjunctions and disjunctions	80
6.9 Avoid performance pitfalls in sub-queries	81
6.10 Avoid unnecessary outer joins	84
6.11 Avoid using SQL expression to transpose values	84
6.12 Apply predicates before groupings	86
6.13 Trace SQL statements back to reports	87

Chapter 7. Troubleshooting	89
7.1 Problem solving strategy	90
7.1.1 The half-split method	90
7.2 Error messages	92
7.3 Log files and tracing	93
7.3.1 Query planning trace	94
7.3.2 Query execution trace	94
7.4 Dynamic Query Analyzer	95
7.4.1 Graph nodes	96
7.4.2 Views	97
7.5 IBM technical support	102
7.5.1 IBM Support Portal	102
7.5.2 Service requests and PMRs	103
7.5.3 IBM Fix Central	103
Related publications	105
IBM Redbooks	105
Online resources	105
Help from IBM	105

Notices

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

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

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

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

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

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

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

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

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

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

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

COPYRIGHT LICENSE:

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

Trademarks

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

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

BigInsights™
Cognos®
DB2®
developerWorks®
IBM®
IBM PureData™
InfoSphere®
PureData™
Redbooks®
Redbooks (logo)  ®
TM1®

The following terms are trademarks of other companies:

Adobe, the Adobe logo, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Netezza, and N logo are trademarks or registered trademarks of IBM International Group B.V., an IBM Company.

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

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

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

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

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

Preface

IBM® Cognos® Business Intelligence (BI) helps you make better and smarter business decisions faster. Advanced visualization capabilities bring your data to life, and you can consume your Cognos BI reports, scorecards, and dashboards through Internet browsers and mobile devices or have them sent to your email inbox. In addition, with intuitive self-service interfaces, you can explore your data and collaborate over the insights you uncover.

The Cognos BI server interprets user gestures and report specifications and translates them into data-retrieval queries that are tailored to the most popular relational database management system (RDBMS), online analytical processing (OLAP), customer relationship management (CRM), and enterprise resource planning (ERP) sources. The term *dynamic query* refers to the planning and execution of queries using the Java-based extensible query engine in the Cognos platform. Leading practices, learned over decades of BI software development, were applied to its design.

The dynamic query layer was developed to meet requirements for interactive reporting and ad hoc analysis. It employs sophisticated, multiphase query optimization techniques and can dynamically alternate between SQL and MDX processing, depending on what best suits the scenario.

Dynamic query has several advantages. Advanced in-memory caching and aggregation can reduce data warehouse workload. Users are provided with a consistent experience, no matter what the data source is. In addition, simple cache administration and query visualization tools help reduce total cost of ownership. IT organizations can also take advantage of improved query performance thanks to the reduction in query planning and execution, along with lighter database server workloads.

This IBM Redbooks® publication explains how Cognos BI administrators, authors, modelers, and power users can use the dynamic query layer effectively. It provides guidance on determining which technology within the dynamic query layer can best satisfy your business requirements. Administrators can learn how to tune the query service effectively and preferred practices for managing their business intelligence content. This book includes information about metadata modeling of relational data sources with IBM Cognos Framework Manager. It includes considerations that can help you author high-performing applications that satisfy analytical requirements of users. This book provides guidance about troubleshooting issues that are related to the dynamic query layer of Cognos BI.

Authors

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



Nigel Campbell is a Senior Developer with IBM Cognos specializing in data access and query engine technologies used by IBM Cognos Business Intelligence products. He has more than 25 years in the industry, building products and applications that span many relational and non-relational platforms.



Henk Cazemier is Senior Development Manager for IBM Relational Planning and Execution and performs additional work as an Architect in this area. Henk has worked at IBM for 27 years and continues to enjoy providing customers with the high-performing query and reporting software for business analytics.



Robert Hatfield is a Development Manager in the IBM Cognos Business Intelligence Data Access group with more than 20 years of experience developing software. He has been involved in dynamic query and Cognos BI product performance optimization since before the introduction of dynamic query. He holds a bachelor's degree in software systems from the University of New Brunswick.



Martin Petitclerc is a Business Analytics Software Architect who has worked in business intelligence software development for more than 19 years. He has led development projects for reporting applications, OLAP and database tools, query planning and execution engines (SQL and MDX), and data mining technology.



Glen Seeds is an Architect in the Cognos BI Platform team, covering all aspects of dimensional queries with a focus on aligning query engine semantics to the needs of BI authors. He has been with the company for 14 years and has made significant contributions to product documentation and training material to guide authors in creating reports that deliver surprise-free results.



Jason Tavoularis is a Product Manager, focusing on the data access layer of IBM Cognos software. He has spent the past several years engaging with IBM Cognos customers through roles in customer support, demonstrations and enablement, and product management. He has a bachelor's degree in computer engineering and an MBA from the University of Ottawa.

The project that produced this publication was managed by:

Marcela Adan, IBM Redbooks Project Leader

Additional support was provided by:

Deana Coble, IBM Redbooks Technical Writer
Shawn Tooley, IBM Redbooks Technical Writer

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

Stanley Chauvin
Tod Creasey
David Cummings
Troy Dodsworth
Rick Kenny
Roch Lefebvre
IBM Cognos Development

Armin Kamal
Pierre Valiquette
Daniel Wagemann
IBM Cognos Customer Support, Proven Practices Team

Sean MacPhee
IBM Business Analytics Information Development

Chris McPherson
IBM Business Intelligence Product Management

Now you can become a published author, too!

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

Find out more about the residency program, browse the residency index, and apply online at:
ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

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

- ▶ Use the online **Contact us** review Redbooks form found at:
ibm.com/redbooks
- ▶ Send your comments in an email to:
redbooks@us.ibm.com
- ▶ Mail your comments to:
IBM Corporation, International Technical Support Organization
Dept. HYTD Mail Station P099
2455 South Road
Poughkeepsie, NY 12601-5400

Stay connected to IBM Redbooks

- ▶ Find us on Facebook:
<http://www.facebook.com/IBMRedbooks>
- ▶ Follow us on Twitter:
<http://twitter.com/ibmredbooks>
- ▶ Look for us on LinkedIn:
<http://www.linkedin.com/groups?home=&gid=2130806>
- ▶ Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:
<https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm>
- ▶ Stay current on recent Redbooks publications with RSS Feeds:
<http://www.redbooks.ibm.com/rss.html>



Overview of Cognos Dynamic Query

This chapter provides an overview of the dynamic query layer of IBM Cognos Business Intelligence (BI) software. The chapter includes an introduction to and history of the dynamic query layer, descriptions of the architecture, and high-level guidance for determining which technology within the dynamic query layer can best satisfy your business requirements.

The chapter contains the following sections:

- ▶ 1.1, “Introduction” on page 2
- ▶ 1.2, “Background” on page 3
- ▶ 1.3, “Architecture” on page 3
- ▶ 1.4, “Technology selection guidance” on page 6

1.1 Introduction

Cognos Business Intelligence (BI) helps you make better and smarter business decisions faster. Advanced visualization capabilities bring your data to life, and you can consume your Cognos BI reports, scorecards, and dashboards through Internet browsers and mobile devices or have them sent to your email inbox. In addition, intuitive self-service interfaces allow you to explore your data and collaborate over the insights you uncover.

The Cognos BI server interprets user gestures and report specifications and translates them into data-retrieval queries that are tailored to the most popular RDBMS, OLAP, CRM, and ERP sources. The term *dynamic query* refers to the planning and execution of queries using the Java-based extensible query engine in the Cognos platform. Leading practices, learned over decades of BI software development, were applied to its design. Dynamic query retains result sets and metadata captured from optimized queries to data sources in a 64-bit, in-memory cache. It can reuse these result sets and metadata to minimize the wait times for future requests.

The dynamic query layer was developed to meet requirements for interactive reporting and ad hoc analysis. It employs sophisticated, multiphase query optimization techniques and can dynamically alternate between SQL and MDX processing, depending on what best suits the scenario.

Dynamic query has several advantages. Advanced in-memory caching and aggregation can reduce data warehouse workload. Users are provided with a consistent experience, no matter what the data source is. In addition, simple cache administration and query visualization tools help reduce total cost of ownership. IT organizations can also take advantage of improved query performance thanks to the reduction in query planning and execution, along with lighter database server workloads.

This book explains how Cognos BI administrators, authors, modelers and power users can use the dynamic query layer effectively.

The dynamic query layer refers to the *query service* of Cognos 10, which is powered by an extensible query engine written in Java. The query layer offers the following key capabilities:

- ▶ Open access to the most popular RDBMS, OLAP, CRM, and ERP data sources
- ▶ Query optimization techniques to address complex analytical requirements, large and growing data volumes, and expectations for timeliness
- ▶ Enterprise-level scalability and stability
- ▶ Intelligent combinations of local and remote processing
- ▶ Federation of multiple heterogeneous data sources
- ▶ OLAP functionality for relational data sources when using a dimensionally modeled relational (DMR) package or Cognos Dynamic Cubes
- ▶ Security-aware caching
- ▶ 64-bit processing
- ▶ JDBC connectivity to relational data sources
- ▶ Query visualizations for ease of maintenance

1.2 Background

New projects in Cognos BI version 10.2.1 are set to the *dynamic query mode*, which is powered by an eXtensible Query Engine (XQE) written in Java. XQE, the engine behind the Cognos BI query service, embraces the principles of abstraction and extensibility, allowing it to evolve into a more efficient query planner, processor, and executor with every new version of Cognos BI.

All techniques and product behaviors that are described in this book assume that the project is using the dynamic query mode. However, existing packages can be set to the *compatible query mode*, which, for reasons of compatibility with previous versions, maintains query behaviors consistent with version 8.4.1 of Cognos BI.

The C++ query engine of version 8.4.1 addressed the analytical challenges of its day, but it was limited in two fundamental ways. The first is that it is 32 bit, which is a problem because effective caching of enterprise data requires 64-bit addressable memory. The second is that its architecture cannot easily adapt to the new trends emerging in the big data era.

XQE was developed to address these limitations and accelerate the improvements to query performance that is delivered in new versions of Cognos BI.

1.3 Architecture

Some of the content of this section was previously published in IBM developerWorks®¹.

The query service accepts data and metadata requests (through the report service component) from authoring interfaces such as IBM Cognos Report Studio, IBM Cognos Report Viewer, and other clients. It returns the requested data or messages in a structured response to the report service component that formats the result for the client. Figure 1-1 on page 4 presents the workflow of requests and responses between these components.

¹ Source: *IBM Cognos Proven Practices: The IBM Cognos 10 Dynamic Query Cookbook*
http://www.ibm.com/developerworks/data/library/cognos/infrastructure/cognos_specific/page529.html

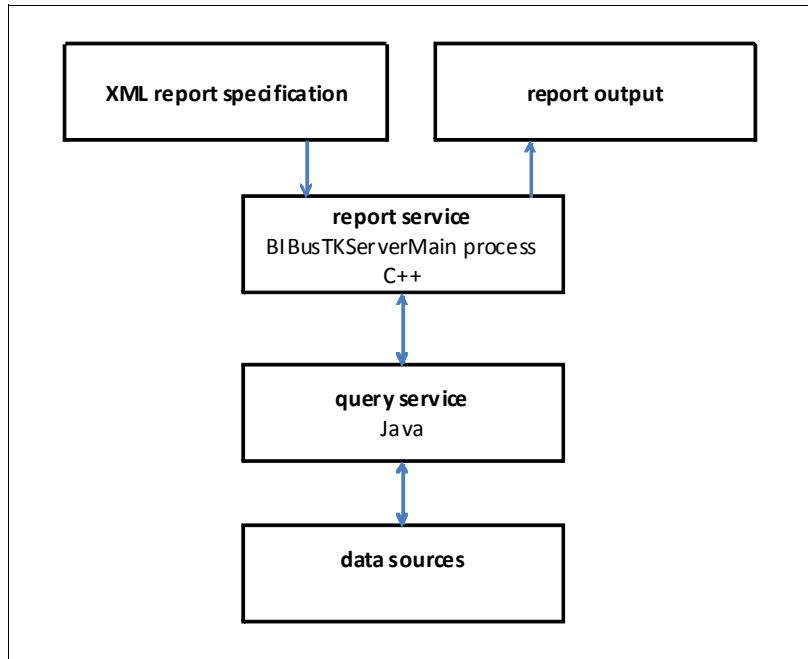


Figure 1-1 Query service request and response workflow

Figure 1-2 shows the internal architecture of the query service, which consists of the following major components:

- ▶ Transformation engine and transformation libraries
- ▶ Query execution engine
- ▶ Metadata cache
- ▶ Data cache
- ▶ RDBMS and OLAP adapters

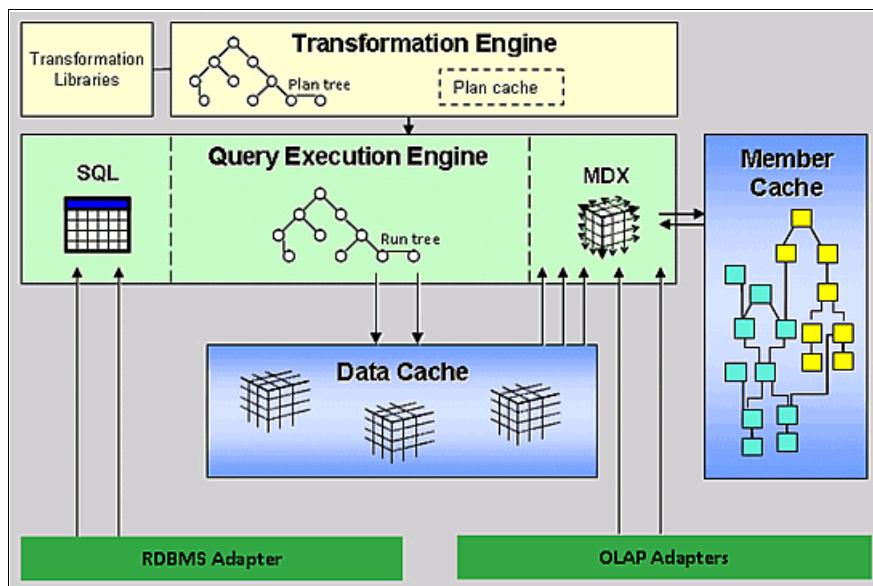


Figure 1-2 Internal architecture of the query service

The *transformation engine* does not implement any query planning logic alone. Instead, it provides an execution environment for query transformations in the *transformation libraries*, thus separating planning logic from the engine. The transformations implement query planning logic for all supported query types and functionality. When there are no more transformations to be applied, query planning is complete and the transformation engine passes the resulting run tree to the query execution engine.

The *query execution engine* can execute any query request, independent of the type of query and target data source. The engine represents all query results in memory in a single format that encompasses both dimensional style (with axes, dimensions, and cells) and relational style (with a tabular format of rows and columns). This allows the engine to combine SQL and MDX queries in a single run tree, thus enabling simplicity of representation, flexibility in post-processing, and streamlined query performance. To process the two types of queries, the query execution engine contains both SQL and MDX engines.

The SQL engine obtains data directly from the *RDBMS adapter*. The query execution engine updates the secure data cache with dimensional data for future reuse. The MDX engine obtains dimensional data either directly from the OLAP adapters or from the data cache. It also updates and reuses dimensional metadata in the secure *member cache*. The cache security features ensure that, by default, no sharing of secured data ever occurs between users with different security profiles.

The RDBMS and OLAP adapters translate Cognos SQL and MDX queries to a query dialect suitable and optimized for each data provider. The adapters send the query and fetch results through the provider's proprietary interface or a supported standard interface such as JDBC. There is only one RDBMS adapter, which uses a JDBC interface, because all supported relational providers are accessible through JDBC. The RDBMS adapter supplies data to the SQL engine in the query execution engine while the OLAP adapters supply data to the MDX engine.

1.3.1 Planning and executing the query

Two major components are involved when the query service processes requests: the transformation engine and the query execution engine. Both engines share a common environment and operate on the same query structures: the *plan tree* and the *run tree*.

An XML parser converts an incoming report request into an initial plan tree, including any embedded SQL, HiveQL or MDX queries. The tree has two main branches: the query, describing *what* the user wants to see, and the QueryResultSet, describing *how* the user wants to see the results (such as in a list or crosstab format).

With the tree in place, the planning process can begin. The transformation engine checks each node in the plan tree to see which query transformations apply to that node. The query transformations implement the logic that transforms a Cognos query into one or more SQL, HiveQL or MDX queries that the one or more target data sources can understand. The transformations also add nodes representing any data manipulation and local processing operations that might be required to produce the requested result.

The transformations occur in several passes, with potentially several iterations per pass, until all possible transformations have been applied. During this process, the transformation engine connects to the IBM Cognos 10 Content Manager to look up model information that applies to the query being processed. When all transformations have been applied, the plan tree has morphed into a run tree and is ready for execution.

The run tree is at the heart of query execution. Results flow from the leaf nodes of the run tree to the root node, where the result is represented in a format suitable for the report service to render the report output. A run tree consists of various types of nodes, each representing a different function:

- ▶ SQL execution
- ▶ MDX execution
- ▶ HiveQL execution
- ▶ Data manipulation
- ▶ Local processing

In the simplest form of a dimensional style query, MDX execution nodes cause the MDX engine to pull data from the data cache (if the cache is not available, the engine sends an MDX query to an OLAP data source). The results are stored in the data cache and go through some data manipulation nodes in the run tree, which might alter the shape of the results. After that, local processing nodes flatten the multidimensional result and sort the data before returning the requested results to the report service.

In a more complex query, such as one against a DMR package, the report request is dimensional in nature, but the data source is relational. This means the query generated for the report is MDX, but the data source only understands SQL. Thus the run tree consists of a mixture of all four types of execution nodes. In this event, the execution engine first sends SQL queries to the relational data source. Local processing nodes then reshape the results into dimensional form for storage in the data cache, from which MDX nodes query data just as they would from a dimensional data provider. Subsequent execution proceeds as it would for a dimensional query against an OLAP data source.

1.4 Technology selection guidance

Cognos offers several query technologies to address your analytical needs. This section provides guidance that can lead you to the best solution for your requirements.

Table 1-1 summarizes the guidance regarding when to employ pure relational, IBM Cognos TM1®, IBM Cognos Dynamic Cubes, or DMR analytics for different application objectives. The remainder of this section gives details about the optimal scenarios for using each of these analytic technologies.

Table 1-1 Summary of analytic technology selection guidance

Application objective	Preferred technology
▶ Reporting on leaf-level records ▶ Static reports (no user requirements for navigating through business hierarchies) ▶ Simple list reports	Pure relational
▶ Users writing back to the same data source being analyzed ▶ What-if analysis ▶ Volatile data because of planning and budgeting applications	IBM Cognos TM1
▶ Self-service interactive analysis ▶ High performance on large and growing data volumes ▶ Data warehouse structured in star or snowflake schema	IBM Cognos Dynamic Cubes
▶ Interactive analysis on operational or transactional database ▶ Tight control over caching ▶ Tight control over security	Dimensionally-modeled relational (DMR)

1.4.1 Pure relational analytics

A pure relational package is one created in IBM Cognos Framework Manager over a relational database without modeling any dimensional context. For many applications, there is no need for OLAP functionality, such as when the application is primarily for ad hoc queries or pre-authored reports with no requirement for drilling up and down. In these cases, you might choose to publish packages based on query subjects alone (with no dimensions defined). A pure relational approach is best for scheduled reports run against a transactional or operational database. It works well for simple list reports that aggregate tens of millions of records. It is common for the target report in a drill-through definition to be purely relational and contain leaf-level records in a tabular format.

1.4.2 OLAP analytics

OLAP technologies work best for interactive analysis involving frequent examination of cross sections of your business. OLAP allows users to drill up and down business hierarchies and interactively create new views of data for specific business contexts. Cognos BI supports several proprietary and third-party OLAP solutions, including Microsoft Analysis Services and Oracle Essbase. In addition, it offers intuitive, web-based user interfaces such as Cognos Workspace Advanced and Analysis Studio that are conducive to navigating your business hierarchies.

IBM Cognos Dynamic Cubes is the preferred OLAP data source for performance reasons and is recommended for use unless a specific constraint prevents you from doing so. This section contains further guidance regarding OLAP technologies in Cognos BI.

Cognos TM1

Cognos TM1 is primarily positioned for financial planning applications where users are submitting planned budgets, contributions, and resources to a centralized server; everyone is working with the same basic facts.

The powerful OLAP server of TM1 scales writers to the thousands. Its configurable model, rules, and user interface layer are used to satisfy a broad range of planning applications (financial, resource, projects, demand, merchandise, customer churn, customer and product profitability, and so on).

When TM1 is employed to satisfy write-back requirements of users, it is preferred that those same TM1 cubes be used for business intelligence reporting and analysis. However, if no such write requirements exist, an alternative OLAP solution is likely best. This is because on-demand aggregations and calculations significantly affect first execution (non-cached) query performance for TM1 cubes when they are loaded with data volumes exceeding 10 GB of disk data or 10 million records. The nature of a MOLAP² (self-contained cube) architecture, such as that of TM1, limits data volumes to what can efficiently be contained in memory.

TM1 is designed for volatility in order to support large numbers of users writing back to the system. For read-only reporting and analysis requirements, Cognos Dynamic Cubes can perform better and satisfy a greater number of users.

² multidimensional online analytical processing (MOLAP)

Cognos Dynamic Cubes

Cognos Dynamic Cubes is an extension of the dynamic query layer that uses substantial in-memory data assets and aggregate awareness to achieve high-performance, interactive analysis and reporting over terabytes of warehouse data. Cognos Dynamic Cubes requires a data warehouse that is structured in a star or snowflake schema to maximize the performance characteristics of the solution.

Cognos Dynamic Cubes is the most scalable, high-performing OLAP solution offered by Cognos BI. Cognos Dynamic Cubes is the preferred solution for customers who have a star- or snowflake-structured data warehouse and want to enable users to perform self-service analysis and reporting on large and growing data volumes.

NOTE: For more information, see *IBM Cognos Dynamic Cubes*, SG24-8064.

Dimensionally modeled relational (DMR) analytics

A DMR package enables an OLAP experience on top of a relational data source. The dimensional layer is defined in Cognos Framework Manager.

DMR analytics give you great control over latency. In Report Studio, you can specify at the query level which queries within a single report can benefit from caching versus which ones should retrieve live data for every request.

DMR analytics also give you great control over security. You can leverage the security defined in your database by associating Cognos users, groups, and roles with the corresponding users, groups, and roles in the RDBMS. In Framework Manager, you can define data-level security (user-specific filters) or object security (to prevent metadata objects from being accessible to certain users). The flexibility with respect to security requirements comes at the expense of less cache sharing. Unlike Cognos Dynamic Cubes, where security is applied on top of the same cache so that all security profiles are engaging the same cache, with DMR, there are separate caches for each security profile.

DMR analytics can have a low total cost of ownership. Aside from the administrative overhead associated with clearing and priming the cache, this approach enables you to essentially *publish and forget*, because you do not need to worry about starting or building cubes.

Consider DMR analytics when particular requirements cannot be satisfied by Cognos Dynamic Cubes. The following most common requirements necessitate DMR analytics instead of Cognos Dynamic Cubes:

- ▶ Need to reuse an existing Framework Manager model
- ▶ Strict latency requirements (up-to-the-minute data)
- ▶ Complex security requirements
- ▶ A data source that cannot be structured as a star or snowflake
- ▶ An underlining data source that is supported by DMR but not Cognos Dynamic Cubes

Using DMR instead of Cognos Dynamic Cubes is preferred in the following scenarios because of the smaller memory footprint and lower administrative overhead:

- ▶ Reporting primarily involves batch jobs scheduled outside of business hours.
- ▶ Reports primarily retrieve a small result, with minimal use of dimensional functions (as is often the case with exception and scorecard reports).
- ▶ DMR is already in use, and users are sufficiently satisfied with DMR performance.

Security differences between DMR and Dynamic Cubes

Figure 1-3 highlights the difference between a DMR package and Cognos Dynamic Cubes with respect to security applied to the corresponding in-memory cubes. A DMR package filters the data used to build a cube based on security filters that are expressed in relational terms. In contrast, Cognos Dynamic Cubes reads the data as a super user with the highest authorization available and then applies security filters expressed in dimensional terms to the same cube.

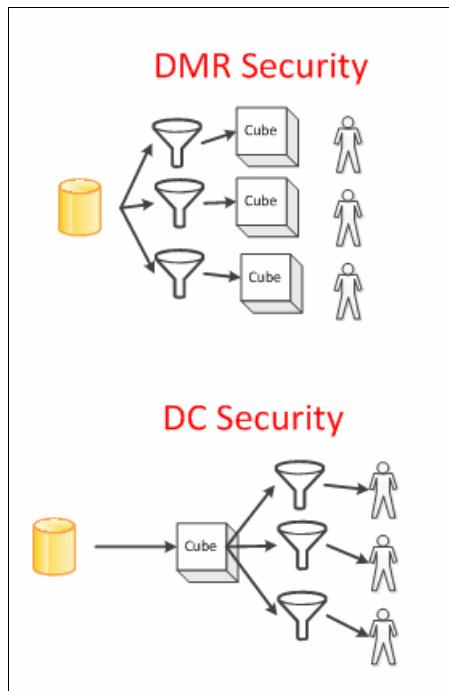


Figure 1-3 Comparison of security in a DMR approach and Cognos Dynamic Cubes (DC)



Administration

This chapter describes configuring the query service, data source administration, and cache management. Administrators can learn how to tune the query service effectively and about preferred practices for managing their BI content.

The chapter contains the following sections:

- ▶ 2.1, “Configuring the query service” on page 12
- ▶ 2.2, “Data source administration” on page 17
- ▶ 2.3, “Cache management” on page 20

2.1 Configuring the query service

When you install the application tier component of IBM Cognos Business Intelligence (BI), the query service is included and enabled by default. The query service can be disabled, and its JVM prevented from launching, by using Cognos Configuration.

Most tuning options for the query service are located in Cognos Administration. To configure settings for the query service, log in to Cognos Administration and then navigate to the Dispatchers and Services area of the Configuration tab to locate the QueryService object (highlighted in Figure 2-1).

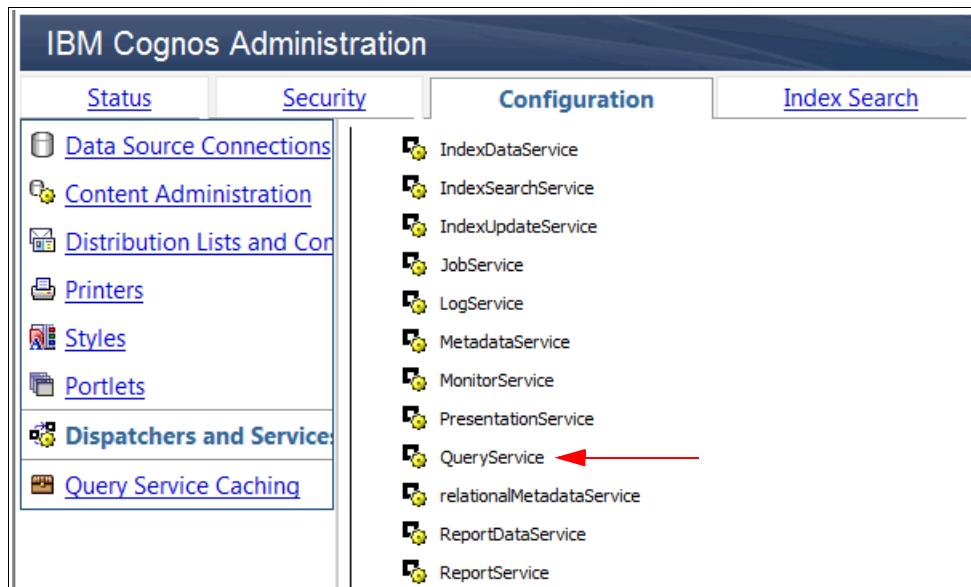


Figure 2-1 QueryService object on Configuration tab of Cognos Administration

The query service has three categories of settings that can be configured: environment, logging, and tuning. By default, each instance of the query service acquires applicable configuration settings from its parent. You can override the acquired values by setting them explicitly on the Settings tab of the Set properties window for the QueryService.

2.1.1 Memory sizing

Although the query service can operate as a 32-bit Java process in 32-bit installations of Cognos BI servers, 64-bit servers are preferred. The initial size and size limit of the JVM heap for the query service can be set in Cognos Administration on the Settings tab of the QueryService object. By default, both the size and size limit are set to 1 GB. For details about setting query properties, see the product information center:

<http://ibm.co/19eGKCd>

In production environments where IBM Cognos Dynamic Cubes is not employed, a good practice is to set the initial heap size to 4 GB and the limit to 8 GB (a 64-bit server is required for memory settings above 2 GB). Then, you can monitor system resource usage during peak periods and adjust the settings accordingly.

For a server that is operating one or more Cognos Dynamic Cubes, the size limit of the query service should be set in accordance with the Cognos Dynamic Cubes hardware sizing recommendations.

The recommendations are available in an IBM developerWorks article:

<http://www.ibm.com/developerworks/library/ba-pp-performance-cognos10-page635/>

Monitoring the JVM heap size of the query service

You also can monitor metrics related to the JVM heap size of the query service. Figure 2-2 shows the JVM metrics view of the QueryService in Cognos Administration, which you can access by navigating to the system area of the Status tab and then selecting the QueryService object for a particular server.

Metrics - QueryService		
<input checked="" type="checkbox"/> 0 <input type="checkbox"/> 2 <input checked="" type="checkbox"/> 1 <input checked="" type="checkbox"/> No metric score		
	Name	Value
<input checked="" type="checkbox"/>	JVM	
	Committed heap size (bytes)	1,048,576,000
	Committed heap size high watermark (bytes)	1,048,576,000
	Current heap size (bytes)	378,019,808
	Current heap size high watermark (bytes)	454,785,800
	Initially requested heap size (bytes)	--
	Maximum heap size (bytes)	1,572,864,000
<input checked="" type="checkbox"/>	Request	
		<input type="checkbox"/> 0 <input type="checkbox"/> 0 <input type="checkbox"/> 0

Figure 2-2 Query service JVM metrics monitor

You can assign a high water mark or threshold for the JVM heap size to cause the indicators to change color when the JVM heap grows beyond the threshold. You then may want to assign more memory to a query service when it approaches its configured JVM heap size limit during peak periods.

Using these query service metrics available in Cognos Administration, you can ensure that there is enough memory both for typical workloads and for additional, infrequent queries that require more memory.

The preferred settings for the query service JVM heap are an initial size of 4096 MB and a maximum size of 8192 MB. The intention of these settings is for the system to use 4 GB or less most of the time, with an additional buffer of 4 GB to handle temporary loads. You can monitor the high value of the committed heap size to detect if the heap usage is too small. Figure 2-3 on page 14 shows the JVM usage at the time of system startup.

Metrics - QueryService	
<input checked="" type="checkbox"/> <input type="checkbox"/> 0 <input checked="" type="checkbox"/> <input type="checkbox"/> 0 <input checked="" type="checkbox"/> 0 <input type="checkbox"/> 0 <input checked="" type="checkbox"/> No metric score	
Name	Value
JVM	
Committed heap size (bytes)	4,294,967,296
Committed heap size high watermark (bytes)	4,294,967,296
Current heap size (bytes)	1,616,279,784
Current heap size high watermark (bytes)	1,616,279,784
Initially requested heap size (bytes)	-- 4,294,967,296
Maximum heap size (bytes)	-- 8,589,934,592

Figure 2-3 Query service JVM metrics monitor at startup

If the system never needs more than 4 GB of heap, the committed heap size will never increase. It will stay at 4,294,967,296 bytes. The current heap size high value indicates the maximum amount of heap that has been used. Because the peak heap size frequently approaches the committed heap size, the JVM may increase committed heap size. If your workload causes the heap to always expand above the initially requested heap size, the initially requested heap size can be increased. If the workload always causes the heap to expand to the maximum size, the JVM heap size is too small. Figure 2-4 shows a JVM where the committed heap size has expanded.

Metrics - QueryService	
<input checked="" type="checkbox"/> <input type="checkbox"/> 0 <input checked="" type="checkbox"/> <input type="checkbox"/> 0 <input checked="" type="checkbox"/> 0 <input type="checkbox"/> 0 <input checked="" type="checkbox"/> No metric score	
Name	Value
JVM	
Committed heap size (bytes)	4,686,348,288
Committed heap size high watermark (bytes)	4,686,348,288
Current heap size (bytes)	2,265,157,888
Current heap size high watermark (bytes)	2,265,157,888
Initially requested heap size (bytes)	-- 4,294,967,296
Maximum heap size (bytes)	-- 8,589,934,592

Figure 2-4 Committed heap size expansion

In Figure 2-4, the heap expanded but the current heap size is much smaller than the committed heap size. This is not unusual because of how the JVM operates. The JVM heap is split into a nursery, or new generation, memory space and a tenured, or old generation, memory space. If one of these memory spaces is frequently low, the JVM will try to expand it. In the case of Figure 2-4, the committed heap was increased in order to grow one of these two heap segments. By increasing the initially requested heap size to 5 GB, the heap expansion will be avoided entirely.

2.1.2 Throughput sizing

With every new release of Cognos BI, optimizations are introduced to maximize the query service's throughput. The query service's cache management system continues to be optimized, too. These improvements reduce user wait times because the Cognos BI server retrieves objects from its own in-memory cache faster than it does from a query to a remote data source. The query service JVM is multi-threaded and makes use of available processors.

As of Cognos BI version 10.2, the maximum number of threads that the query service can process concurrently is determined dynamically based on the number of processes and affinity settings defined for the ReportService within Cognos Administration. By default, the Cognos BI server is configured to have two report service processes with eight low-affinity and two high affinity threads for each. With these settings, the query service is capable of sending 20 concurrent queries to data sources.

If an administrator uses Cognos Administration to adjust either the affinities or the number of available processes for the ReportService, the query service will automatically adjust to allow the increase in concurrent requests without requiring any additional query service-specific configuration changes.

The preferred starting point for tuning your Cognos BI server is to leave the affinity settings at their default values and set the number of report service processes to 1.5 times number of available processors. For example, if your Cognos BI server is the only application running on hardware that has two processors, set the number of processes for the report service during peak and non-peak periods to three as a starting point, and then monitor system resource usage and adjust the number accordingly.

Increasing the query service's throughput is intended to reduce the risk of a user waiting while their query is queued because the Cognos BI server is operating at capacity. Yet in certain high-load scenarios, additional throughput might result in longer user wait times. This additional wait time is attributed to the fact that each concurrent request must establish its own data source connection and in-memory cache instead of re-using one that was created by a previous request. For example, when using a query service configured for four report service processes, each with eight low-affinity and two high-affinity threads, a concurrent 40-user load test will send all 40 requests concurrently. This, in turn, will result in 40 simultaneous data source connections. If the settings were such that only 10 data source connections could be opened concurrently, then the remaining 30 requests would be satisfied by reusing one of the 10 initial data source connections, thus reducing the overall load on the underlying data source. In situations where the overall data source load causes the request performance to suffer significantly, you can reduce the number of processes for the report service to limit throughput and make better use of caching.

Batch reporting

The Cognos BI architecture differentiates between the processing of interactive and non-interactive requests. All requests that are initiated through user activity are processed by the report service; scheduled or event-driven activity is processed by the batch report service. The query service returns results retrieved from data sources to the requesting batch or report service.

Scheduled reporting is a critical aspect of any large-scale enterprise reporting solution. The effective management of low or non-interactive usage time periods, in combination with an organization's data refresh cycles, provides an opportunity for administrators to prepare as much information as possible during off-peak times for later retrieval by users.

Reports can be scheduled on an individual, one-by-one basis, but this can be a burden when you have many reports to schedule. Instead, you can use jobs to execute scheduled activities. A job is a container of scheduled processing activities that run in a coordinated manner. Instead of scheduling individual reports, a job allows multiple reports to execute using the same schedule. Each activity within a job is given a sequence order based on how the job was selected. Jobs can be submitted to run either all at once (all activities in the job execute simultaneously) or in sequence (activities execute one at a time based on their sequence order).

The query service throughput for batch requests is determined in the same manner used for interactive requests, except that the associated properties of the BatchReportService in Cognos Administration are retrieved rather than those for the ReportService.

2.1.3 Multi-server environments

By default, Cognos BI loads balance dynamic query report executions among all of the application tier servers that are operating active query services in the environment. If existing servers are at capacity during peak periods, you can add more servers to scale your environment up to accommodate more requests.

Cognos BI version 10.2.1 does not permit a single query execution to write to the in-memory query service cache of multiple Cognos BI servers. Only the server that executed the query can have its cache primed by that query. So to best take advantage of query service caching in a multi-server environment, all content that might use a common cache should be tied to a single server (assuming the load is not too great for the one server to handle). By configuring advanced dispatcher routing (ADR), you can route requests based on a Cognos security profile (user, group or role), or by package. Using ADR, you can minimize the number of cache misses by directing package A content to server X only, and package B content to server Y only.

For more information about advance dispatcher routing, see the product information center:

<http://ibm.co/138Qa19>

2.2 Data source administration

Figure 2-5 shows how objects are organized in the data source connection area of Cognos Administration.

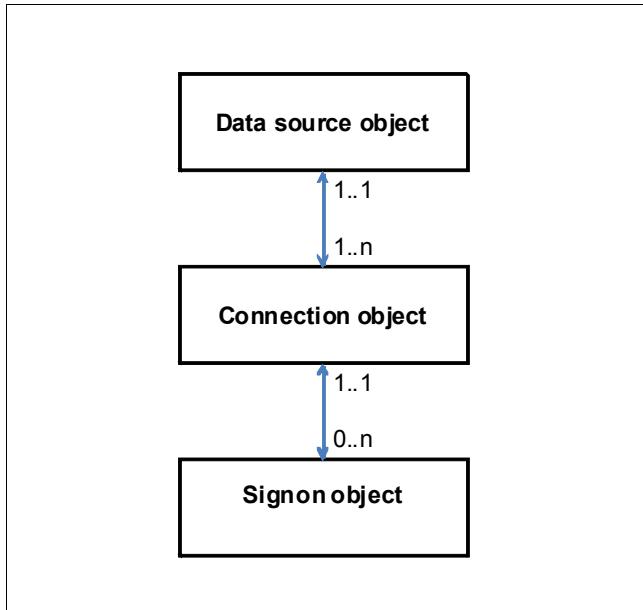


Figure 2-5 *Structure of data source administration objects*

At the highest level are data sources. A data source is really only a container for one or more connections. When running a report, one or more data sources are accessed based on how the Framework Manager model was defined. Therefore, the user must appropriate permissions on the data source object (or objects) being referenced by the report.

Every data source object must contain one or more connection objects. A connection holds information such as the type of data provider being accessed and the provider's server hostname and port number. If a user has permissions on only one connection within a data source, then that connection is automatically used. If a user has permissions on more than one connection within a data source, then the user is prompted to declare which connection they want to use when they run the report.

Within every connection object there can be one sign-on object, more than one sign-on object, or no sign-on object. A sign-on object simply holds the user name and password needed to connect to the database (as defined by the connection object). If a user has permissions on only one sign-on within a connection, then that sign-on is automatically used. If a user has permissions on more than one sign-on within a connection, then the user is prompted to declare which sign-on to use when running the report. If a user does not have permissions on any sign-ons or there are no sign-ons defined in the connection, then that user is prompted to enter a user name and password for a secured database when running the report.

2.2.1 Connection command blocks

Connection command blocks are mechanisms with which the Cognos BI server can send additional context to a data source. Data source administrators want to know details about applications that connect to their database systems and they use this information for auditing, workload management, and troubleshooting.

Open connection, open session, close session, and close connection command blocks can be defined on the last window of the New data source wizard in Cognos Administration. Using these command blocks, Cognos BI administrators can give database administrators details about reporting and analysis applications and the users who are accessing them. These details might include the default set of information about authenticated users that is retrieved from authentication providers.

This information can be extended by specifying custom namespace mappings in Cognos Configuration. Using the mechanisms built into your database, you can implement the most appropriate methods of passing Cognos context to the database. The macro functions supported by the query service can provide information in a command block about users and reporting application objects, such as packages, reports, or queries. All macro functions can return values when referenced from a command block, which allows for application context to be passed to the data source from a command block. Macro functions that reference parameter maps in a model may also be used.

More information about connection command blocks including instructions and numerous examples are in the product information center:

<http://ibm.co/138SKV7>

2.2.2 JDBC drivers

The query service employs Java Database Connectivity (JDBC) driver files to connect to non-OLAP data sources. Appropriate driver .jar files must be made available to the query service, so you must have access to the JDBC driver provided by your database vendor.

Start by selecting the driver version that matches the version of your data source. This version can vary depending on software updates, patches, revisions, and service packs. For some JDBC drivers, you might also need to make appropriate license files available to the query service.

With the driver version selected, use the following procedure to make driver and license files available to the query service.:

1. Install the appropriate database software.
2. Copy the JDBC driver file (or files) to the c10_location\webapps\p2pd\WEB-INF\lib directory.
3. Stop and restart the Cognos BI server.

You can choose to use a JDBC 4.0 driver (if your vendor offers one) or a JDBC 3.0 driver. For IBM DB2®, the drivers are named db2jcc4 and db2jcc, respectively. You can use either a JDBC 4.0 or 3.0 driver as the query service is not dependent on JDBC methods, which only exist when a JDBC 4.0 driver is present.

The JDBC standard uses terms such as type-2 and type-4 to describe key architectural implementations. A *type-2* driver is a JDBC driver which combines Java code on top of non-Java libraries that must be installed on the machine. A *type-4* driver is written entirely in Java and communicates with the server using the internal protocol the server understands. A vendor might offer a type-2 driver and in some cases deliver functionality that is not provided in their type-4 driver. If your vendor offers a type-2 or type-4 driver, assume that only the type-4 driver is tested in the IBM lab unless stated otherwise on the Software Compatibility Report (SPCR) for the version of Cognos BI you are using. The SPCRs for Cognos BI products are on the Cognos support website:

<http://www.ibm.com/support/docview.wss?uid=swg27037784>

In certain cases, a vendor may offer more than one JDBC driver that is capable of interacting with its servers. For instructions of how to configure the JDBC connection to a particular type of data source, see the following resources:

- ▶ Product information center:

<http://pic.dhe.ibm.com/infocenter/cbi/v10r2m1/index.jsp>

- ▶ Cognos support website:

<http://www.ibm.com/cognos/support>

2.2.3 OLAP connections

The query service connects to OLAP data sources such as IBM Cognos TM1, SAP BW, Oracle Essbase, and Microsoft Analysis Services through vendor-specific full, or thick, client software that must be installed on the same server as the query service. Many vendors distribute their runtime clients with numbers that match the server version with which the clients were initially released. In some cases, however, the clients can be released with their own version numbers and release cycles.

Typically, you must configure certain environment variables to allow the query service to locate the corresponding OLAP client. For instructions on how to configure the native client connection to a particular type of OLAP data source, see the product information center and the Cognos support website at the addresses provided at the end of the previous sub-section.

XMLA support

The query service can query Microsoft Analysis Services through XMLA, the industry standard for analysis. XMLA is based on web standards such as XML, SOAP, and HTTP and therefore does not require any additional client software to be made available to the Cognos BI server. XMLA allows the query service installed on Linux and UNIX platforms to support Microsoft Analysis Services (which has a native client available only on Windows operating systems).

2.2.4 ERP and CRM data sources

The query service also supports a variety of ERP and CRM providers including SAP ECC, Oracle Siebel, and Salesforce.com, each of which are treated as relational databases. For instructions of how to configure connections to these data sources, see the product information center and the Cognos support website at the addresses provided at the end of 2.2.2, “JDBC drivers” on page 18.

2.3 Cache management

This section covers administrative aspects of managing the query service's cache. See Chapter 3, “Metadata modeling” on page 25 for details about the types of objects that can be cached in the Java memory and the conditions for which those objects may be reused.

2.3.1 Priming the cache

The query service's in-memory cache can be primed (populated with reusable objects) only by means of executing reports, either in batch mode or interactively.

Reports that retrieve the most commonly accessed data are best scheduled to execute before business hours. This minimizes the wait times for users during the day. An effective way of priming the cache in this manner is to create a job consisting of simple reports that include the data most frequently consumed during interactive analysis.

Results that are cached from one user's request may be used for another user's request (under default settings) only if both users have the same data security profiles. This is explained in Chapter 3, “Metadata modeling” on page 25. So, to prime the cache for the greatest possible number of users, consider scheduling the same report to run under various security profiles.

Reports that are ideal for cache priming include those from a known set of dashboards that a majority of users will use. For example, if there is a set of dashboards or reports that most users use as a starting point, these reports are good candidates for priming because all of those users can benefit from the fast performance that cached values can provide.

Reports that contain large volumes of data and many levels of hierarchies are also good candidates for cache priming because other reports that contain a subset of the data and metadata retrieved when executing the original report might benefit from the associated cached objects.

2.3.2 Clearing the cache

Java garbage collection is a term used to describe memory management operations in which memory associated with artifacts that are not actively being used by the program are automatically reclaimed under certain conditions. In this case, Java garbage collection automatically begins clearing out memory when the configured size of the query service's JVM heap nears the limit of available space. Yet it is best not to rely on this safeguard. The problem with approaching an out-of-memory condition is that processor cycles result in being devoted to garbage collection instead of query executions.

Clear cache operations can be initiated manually or scheduled from Cognos Administration. The Query Service Caching section of the Configuration tab allows for manual cache clearing and writing the cache state to file for one or more server groups. The Write cache state feature creates the following time-stamped XML file showing the state of all OLAP caches.

```
c10\logs\XQE\SALDump_a11_a11_a11_timestamp.xml
```

In a distributed installation, each report server that has an OLAP cache writes the cache state file to its local logs directory.

Pure relational caching

Results from pure relational queries are retained in memory only while the data source connection remains open. This is different from dimensional queries, such as those for DMR packages, which, as of Cognos BI version 10.2, persist in-memory until a manual or scheduled clear cache operation occurs. Query service data source connections have a default idle timeout of 300 seconds (5 minutes), but this can be set to a different value using the Properties window of the query service in Cognos Administration. For details about modifying these properties, see the product information center:

<http://ibm.co/14Vhe5c>

DMR, SAP BW, Oracle Essbase, and Cognos Dynamic Cubes caching

The typical cache management strategy when using DMR packages, SAP BW, Oracle Essbase, and Cognos Dynamic Cubes is to schedule a clear cache operation following a load of new data in the underlying data source. These caches can be cleared using the query service administration task wizard in Cognos Administration (shown in Figure 2-6). These cache clearing tasks can be targeted at a specific package or data source. An asterisk (*) entered in either the package or data source field is considered a wild card that will clear everything.

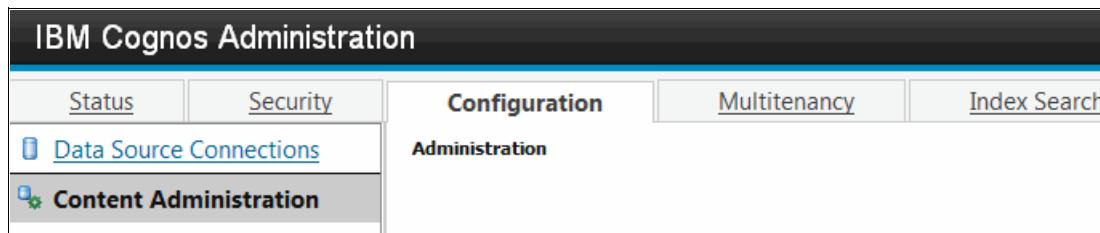


Figure 2-6 New query service administrative task wizard

TM1 caching

The query service can detect updates in Cognos TM1 cubes and automatically clear any associated stale data from its memory. Therefore taking any action to clear the cache for TM1 data sources is unnecessary.

2.3.3 Automating cache operations

The query service can be instructed to perform cache operations automatically in response to external events. The Cognos BI Software Development Kit (SDK), trigger-based scheduling, and the query service's command line API are the three primary mechanisms for which you can automate query service cache operations in conjunction with external business operations such as loading new data into your data warehouse.

The SDK and trigger-based scheduling are the most popular mechanisms to add query service cache operation automation into programming scripts and are explained in the standard product documentation. Use of the query service command-line API does not appear in the product documentation. However, because some system administrators might find the syntax of the command-line API more convenient than that of the SDK or trigger-based scheduling, the query service command-line API is explained here.

Managing cache operations with the SDK

The Cognos BI SDK provides a platform-independent automation interface for working with Cognos BI services and components. The SDK uses a collection of cross-platform web services, libraries, and programming interfaces. You can choose to automate a specific task or you can program the entire process, from modeling through to reporting, scheduling, and distribution.

The query service administration task wizard in Cognos BI was built using the Cognos BI Software Development Kit (SDK). Any task that can be defined through the wizard can also be established programmatically using the SDK. For more information, see the Cognos BI SDK documentation or the following developerWorks article:

http://www.ibm.com/developerworks/data/library/cognos/development/how_to/page565.html

Managing cache operations with trigger-based scheduling

Both priming the query service (by running reports) and clearing the service caches can be scheduled using the standard scheduling interface of Cognos Connection. There, you can use triggers to schedule certain actions based on an occurrence, such as a database refresh or receipt of an email. The occurrence acts as a trigger causing the entry to run, such as when you run a report every time a database is refreshed. Trigger-based scheduling can help prevent exposing users to stale data. Triggers provide a mechanism for which to implement a script for the routine presented in Figure 2-7. To set up a trigger occurrence, see the following documentation:

<http://ibm.co/17Jm9sz>

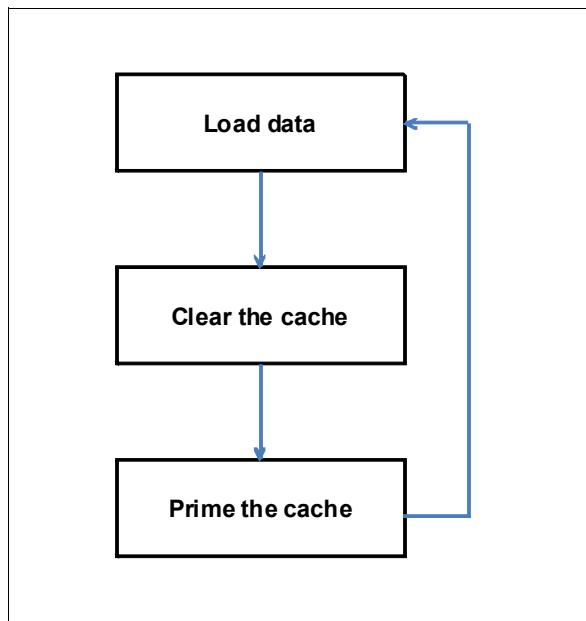


Figure 2-7 Scripting to optimize cache lifecycles

Managing cache operations with the command-line API

In addition to the Cognos Administration interface for executing and scheduling cache management tasks, a command-line API enables manual and automated cache management outside the normal Cognos BI administration environment. The command-line utility is located in the c10\bin directory and has either of the following names, depending on your operating system:

- ▶ `QueryServiceAdminTask.sh`
- ▶ `QueryServiceAdminTask.bat`

The `QueryServiceAdminTask` utility accepts up to two arguments:

- ▶ Cache operation (mandatory)

Here you specify one of the following values to select the corresponding cache operation:

- A value of 1 to clear the cache
- A value of 2 to write the cache state

- ▶ Cache subset (optional)

Use this argument to specify the portion of the cache to which the operation applies by naming a data source, catalog, and cube (separated by forward slashes). You can use the wildcard character (*) to represent all data source, catalog, or cube names. Omitting this argument causes the cache operation to apply to the entire cache.

For example, to clear the cache for all cubes in all catalogs under all data sources, enter the following command in a command shell:

```
queryServiceAdminTask 1 "*//*/*"
```

Optionally, you can enter this command:

```
queryServiceAdminTask 1
```

Entering `QueryServiceAdminTask -help` in a command shell displays detailed usage instructions for the utility.

Because this command-line utility makes an immediate task request, it does not go through the job scheduler and monitoring service. Consequently, it affects only the Cognos BI server on which the command is run.



Metadata modeling

This chapter describes metadata modeling of relational data sources with IBM Cognos Framework Manager, a metadata modeling tool that drives query generation for IBM Cognos Business Intelligence (BI). Cognos BI enables performance management on normalized and denormalized relational data sources and also a variety of OLAP data sources.

The chapter contains the following sections:

- ▶ 3.1, “Cognos Framework Manager” on page 26
- ▶ 3.2, “Goals of metadata modeling relational data sources” on page 26
- ▶ 3.3, “Framework Manager architecture” on page 27
- ▶ 3.4, “Key objects of a relational model” on page 28
- ▶ 3.5, “Organizing relational models” on page 35
- ▶ 3.6, “Relational modeling for performance” on page 36

3.1 Cognos Framework Manager

With Cognos Framework Manager you can do the following tasks:

- ▶ Create a project representing your data source (or sources)
- ▶ Import required metadata, such as tables and views, from your data source
- ▶ Model the metadata for your users
- ▶ Set and manage security
- ▶ Publish packages to make appropriate parts of the model available to report authors and self-service analysts

The chapter presents important reminders, examples, and preferred practices, but assumes you are already familiar with Framework Manager modeling concepts. It also assumes your project is using the dynamic query mode.

For additional details about Framework Manager modeling, see the following resources:

- ▶ Framework Manager User Guide:
http://pic.dhe.ibm.com/infocenter/cbi/v10r2m1/nav/5_10
- ▶ Guidelines for Modeling Metadata:
http://pic.dhe.ibm.com/infocenter/cbi/v10r2m1/nav/5_5
- ▶ IBM developerWorks articles on proven practices in business analytics:
<http://www.ibm.com/developerworks/analytics/practices.html>
- ▶ Business analytics product training from IBM Education:
<http://www.ibm.com/software/analytics/training-and-certification/>

3.2 Goals of metadata modeling relational data sources

Cognos Framework Manager is a metadata modeling tool where the building blocks for authoring reports and performing analysis are defined. A Framework Manager model is a business-focused representation of information from one or more data sources. It allows you to define reusable objects for security, translations, custom calculations, and other functions in a way that allows a single model to serve the needs of many groups of users.

Modeling with Framework Manager is an iterative process of refining different views of your metadata, starting with the data source view, then the business view, and finally the presentation view that your users consume. The end result is a metadata model that depicts all of your organization's data in a simplified way that hides the structural complexity of the underlying data sources.

3.2.1 Modeling for self-service analysis

Successful self-service applications require presentation layers that have intuitive naming conventions and data formatting specifications that align with business conventions.

Performance problems associated with long-running queries during ad hoc analysis can be avoided by adding appropriate embedded filters into the Framework Manager model. Modelers should also consider adding reusable stand-alone filters and calculations to the package to help users avoid the wait times that develop during the sequence of steps that users undertake to define the filters and calculations themselves.

3.3 Framework Manager architecture

Framework Manager is a client-side graphical user interface that performs two primary functions:

- ▶ Updates the project's `model.xml` file, which is the primary repository for models created in Framework Manager
- ▶ Calls on the query service and other components of the Cognos BI server as needed

Figure 3-1 illustrates the communication channels with components of the Cognos BI server during various Framework Manager operations.

Framework Manager uses information entered into Cognos Configuration to communicate with server components. For example, Framework Manager uses the configured dispatcher URI to locate the active content manager service that populates the list of available data sources for metadata import. When testing query items, the query service on a Cognos BI server is what establishes connectivity to the data source and returns the requested results to Framework Manager. Framework Manager does not communicate with dynamic query data sources directly, which means that the associated JDBC drivers (which Framework Manager will not use) only need to be made available to the Cognos BI server. Note that a valid gateway URI must be entered into Cognos Configuration because Framework Manager connects through the gateway to authenticate users.

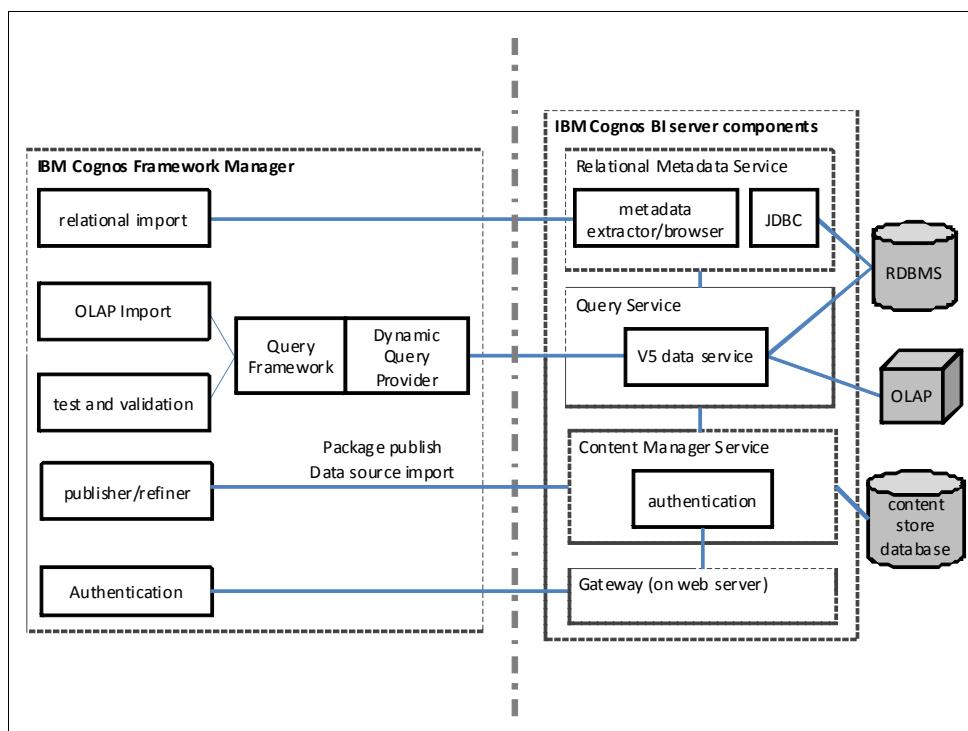


Figure 3-1 Framework Manager communication channels with Cognos BI server components

3.4 Key objects of a relational model

Query subjects, dimensions, determinants, and relationships are the primary objects used to build a metadata model. This section explains each of these objects and provides guidance of how to use them to convey the rules for generating effective queries of your business intelligence.

3.4.1 Query subjects

A query subject is a set of query items that have an inherent relationship. For example, a table and its columns in a relational database may be represented in Cognos BI as a query subject and its query items. You use Framework Manager to modify query subjects to optimize and customize the data that they retrieve, such as by adding filters or calculations. When you change the definition of a query subject, Framework Manager regenerates the associated query items to ensure that any changes to the query subject properties are reflected in all query items for that query subject.

Various types of query subjects are available in Framework Manager:

- ▶ Data source query subjects
- ▶ Model query subjects
- ▶ Stored procedure query subjects

Data source query subjects

Data source query subjects directly reference data in a single data source. Framework Manager automatically creates a data source query subject for each table and view that you import into your model.

For example, if you import the Employee Detail Fact table from the Great Outdoors Warehouse sample database (included with all Cognos BI products), Framework Manager creates a query subject using the following SQL statement:

```
Select * from [go_data_warehouse].EMPLOYEE_DETAIL_FACT
```

Framework Manager generates query subjects that represent tabular data from the data source. In this way, a query subject that references an entire table contains query items that represent each column in the table. If the SQL selects only specific columns, only those columns are represented as query items.

Unlike model query subjects (described later in this chapter), each data source query subject can reference data from only one data source at a time. Yet the advantage of data source query subjects is that you can directly edit the SQL that defines the data to be retrieved. This means that you can insert parameters to tailor your queries based on variables that are populated at run time, including attributes of the user that is initiating the query.

Changing the type of SQL entered into data source query subjects

By default, the SQL statement for a data source query subject is Cognos SQL but you have the option to define it as either native SQL or pass-through SQL. Native and pass-through SQL statements must be completely self-contained and must not reference anything outside that SQL, such as database prompts, variables, or native formatting that would normally be supplied by the calling application. In contrast, Cognos SQL statements are analyzed using metadata from either the model or the relational data source. By default, Cognos SQL is case-sensitive, so it looks up metadata using the names as they are displayed in the SQL statement.

If you change an existing query subject to native SQL or pass-through SQL, you must first ensure that the SQL reflects the rules that apply to the native data source so that your query runs properly.

Cognos SQL

By default, Cognos SQL is used to create and edit query subjects in Framework Manager. Cognos SQL adheres to SQL standards and works with all relational and tabular data sources. The Cognos BI server generates Cognos SQL that is optimized to improve query subject performance, such as by removing unused elements at query time.

Cognos SQL works with any supported database because it is transformed into the appropriate native SQL at query time. In this way, working with Cognos SQL is preferable to the native or pass-through methods.

Native SQL

Native SQL is the SQL that the data source uses, such as DB2 LUW SQL or Oracle SQL. Use native SQL to pass the SQL statement that you enter to the database. Cognos BI can add statements to what you enter to improve performance. You cannot use native SQL in a query subject that references more than one data source in the project.

Pass-through SQL

Use pass-through SQL when the SQL statement that you enter is not valid inside a derived table or subquery. Pass-through SQL lets you use native SQL without any of the restrictions that the data source imposes on subqueries. This is because pass-through SQL query subjects are not processed as subqueries. Instead, the SQL for each query subject is sent directly to the data source where the query results are generated.

If the SQL you are entering is valid inside a derived table or subquery, identify it as native instead of pass-through because doing so increases the opportunity for improved performance when more query processing is performed by the database and less data is returned from the database to the Cognos BI server. To optimize performance, the Cognos BI server will always try to submit as much of the processing to the database as possible and will employ derived tables to do it. Identifying custom SQL as pass-through SQL prevents the Cognos BI server from submitting this SQL inside of a derived table.

Model query subjects

Model query subjects are not generated directly from a data source but are based on query items in other query subjects or dimensions, including other model query subjects. By using model query subjects, you can create a more abstract, business-oriented view of a data source.

Model query subjects are based on the metadata in your model. This allows you to take the following actions:

- ▶ Rename items in your model and reorganize them into a layer that is appropriately presented for authors.
- ▶ Reuse complex SQL statements that exist elsewhere in the model.
- ▶ Reference objects from different data sources in the same query subject.

The SQL for a model query subject is generated by the query service and cannot be edited directly. If you want to edit the SQL of a model query subject, the preferred method is to copy the SQL for the model query subject from the query information tab and paste it into a new data source query subject. Otherwise, you can convert the model query subject into a data source query subject through the Actions menu.

Stored procedure query subjects

Stored procedure query subjects are generated when you import a procedure from a relational data source. Framework Manager supports only user-defined stored procedures. System-stored procedures are not supported.

The procedure must be run in Framework Manager to get a description of the result set that the procedure is expected to return. The stored procedure must return a single uniform result set. Cognos BI supports only the first result set that is returned. If the procedure can conditionally return a different result set, the format of that set must be consistent with the one used to define the metadata in Framework Manager.

Each result set must be returned in the same format, such as the same number and types of columns and column names. Overloaded signatures are supported by Cognos BI, but each procedure must be defined with a unique name and a separate query subject for each result set. Output parameters are not supported.

After you import or create a stored procedure query subject, it displays as a broken object. You must run it to validate the underlying stored procedure and specify the projection list. Static metadata often does not exist for the stored procedure in the relational data source that describes what a result set may look like. The result set may be known only at run time. When a stored procedure is updated in the data source, running the stored procedure in Framework Manager updates the query subject using the newly generated query items.

Sometimes, functions are imported as stored procedure query subjects. Review the stored procedure definition to determine what the procedure expects to be passed and what it attempts to return. Edit and test each stored procedure query subject that you think can be a function. If the test fails, the query subject is a function and must be deleted.

As of Cognos BI version 10.2.1, you can specify the type of transaction that is used by stored procedure query subjects. By default, a query subject that contains a stored procedure is run in a read-only transaction. However, the stored procedure might include operations that require a read/write transaction. The *transaction access mode* property for data sources specifies the access mode of a new transaction. The options are as follows:

- ▶ Unspecified: A new transaction is started in the default mode of the JDBC driver
- ▶ Read-Only: A new transaction is started in read-only mode
- ▶ Read-Write: A new transaction is started in read/write mode

The *transaction statement mode* property applies only to the compatible query mode and is ignored in the dynamic query mode.

3.4.2 Dimensions

Dimensions must be defined to enable the OLAP experience associated with a dimensionally modeled relational (DMR) package. A *dimension* is a broad grouping of data about a major aspect of a business, such as products, dates, or markets. The types of dimensions that you can work within Framework Manager are regular dimensions and measure dimensions.

Table 3-1 on page 31 presents an example of the dimensions in a project for sales analysis.

Table 3-1 Dimensions of a sample project for sales analysis

Name	Type	Description
Time	Regular	Dates of sales organized into years, quarters, months, weeks, and days when sales were made
Region	Regular	Locations of sales grouped into sales regions, countries, and cities
Product	Regular	Product details organized by product type, brand, model, color, and packaging
Customer	Regular	Customer information
Sales	Measure	Purchase details such as units sold, revenue, and profit

Query subjects and dimensions serve separate purposes. The query subject is used to generate relational queries and can be created using star schema rules, while the dimension is used for DMR analytics, which introduces OLAP behavior. Because query subjects are the foundation of dimensions, a key part of any dimensional model is a sound relational model. By creating a complete relational model that delivers correct results and good performance, you will have a strong foundation for developing a dimensional model.

3.4.3 Determinants

Determinants establish granularity by representing subsets or groups of data in a query subject and are used to ensure correct aggregation of repeated data. Determinants are closely related to the concept of keys and indexes in the data source and are imported from the database based on unique key and index information in the data source. It is preferred that you always review the determinants that are imported and, if necessary, modify them or create additional ones. By modifying determinants, you can override the index and key information in your data source, replacing it with information that is better aligned with your reporting and analysis needs. By adding determinants, you can represent groups of repeated data that are relevant for your application.

Determinants affect the grouping and aggregation of data, including other query subjects that have relationships with the query subject and also the query subject itself. When you define a non-unique item as a determinant, you should specify the Group by check box. This indicates to the Cognos BI server that when the keys or attributes associated with that determinant are repeated in the data, the server should apply aggregate functions and grouping to avoid double-counting. Do not specify determinants that have either or both of the following check boxes selected:

- ▶ Uniquely identified
- ▶ Group by

Determinants for query subjects are not the same as levels and hierarchies for regular dimensions but they can be closely related to a single hierarchy. If you plan to use your query subjects as the foundation for dimensions, consider the structure of the hierarchies you expect to create and ensure that you have created determinants that will support correct results when aggregating. The query subject should have a determinant specified for each level of the hierarchy in the regular dimension. Specify the determinants in the same order as the levels in the regular dimension.

If you expect to have multiple hierarchies that aggregate differently, you might consider creating an additional query subject with different determinants as the source for the other hierarchy.

Although determinants can be used to solve a variety of problems related to data granularity, also use them in the following primary cases:

- ▶ A query subject that behaves as a dimension has multiple levels of granularity and will be joined on different sets of keys to fact data. An example is a time query subject that has multiple levels granularity and it is joined to the inventory query subject on the month key and to the sales query subject on the day key.
- ▶ There is a need to count or perform other aggregate functions on a key or attribute that is repeated. For example, the time query subject has a month key and an attribute, days in the month, that is repeated for each day. If you want to use days in the month in a report, you do not want the sum of days in the month for each day in the month. Instead, you want the unique value of days in the month for the chosen month key.
- ▶ You want to uniquely identify the row of data when retrieving text BLOB data from the data source. Querying BLOBS requires additional key or index type information. If this information is not present in the data source, you can add it using determinants.
- ▶ A join is specified that uses fewer keys than a unique determinant that is specified for a query subject. There will be a conflict if your join is built on a subset of the columns that are referenced by the keys of a unique determinant on the 0..1 or 1..1 side of the relationship. Resolve this conflict by modifying the relationship to fully agree with the determinant or by modifying the determinant to support the relationship.
- ▶ You want to override the determinants imported from the data source that conflict with relationships created for reporting. For example, there are determinants on two query subjects for multiple columns but the relationship between the query subjects uses only a subset of these columns. If it is not appropriate to use the additional columns in the relationship, then you must modify the determinant information of the query subject.

Determinants example

Table 3-2 on page 33 presents sample rows from a time query subject with unique foreign keys. Table 3-3 on page 33 presents sample rows from a time query subject that has non-unique month keys. These two data sets illustrate the concept of determinants.

In both data sets, the day key is the unique key of the table, so you can associate all of the columns in the table to this key. Because it is a unique key in both data sets, in both scenarios you identify the day key as a determinant with the check boxes in the following states:

- ▶ Selected: Uniquely identified
- ▶ Deselected: Group by

In both scenarios, the year key is not unique so the Uniquely identified box should remain deselected for this determinant. However, because the year key is all that is needed to identify a year in the data, the **Group by** box is selected to ensure that both the *select distinct* and *group by* SQL clauses are used to display individual years in reports instead of repeated values.

The values of the month key are what provide the difference between the two data sets in this example. Unlike the month key data in Table 3-2 on page 33, the month key data in Table 3-3 on page 33 is not sufficient to identify a particular month in the data (because January in different years would have the same month key value). For the Table 3-2 on page 33 scenario, only the month key is required for the month determinant because each key contains enough information to identify the group within the data. For the Table 3-3 on page 33 scenario, the month determinant requires both the month key and the year key to identify months as a sub-grouping of years. Table 3-4 on page 33 summarizes the determinants of both data sets.

Table 3-2 Sample data set with unique month keys

Year key	Month key	Month name	Day key	Day name
2013	201301	January	20130101	January 1, 2013
2013	201302	January	20130102	January 2, 2013

Table 3-3 Sample data set with non-unique month keys

Year key	Month key	Month name	Day key	Day name
2013	01	January	20130101	January 1, 2013
2013	01	January	20130102	January 2, 2013

Table 3-4 Determinant settings for table 3-2 and 3-3 data sets

Data set	Name of determinant	Key	Attributes	Uniquely identified	Group by
Table 3-2	Year	Year key	None	No	Yes
Table 3-3	Year	Year key	None	No	Yes
Table 3-2	Month	Month key	Month name	No	Yes
Table 3-3	Month	Year key, month key	Month name	No	Yes
Table 3-2	Day	Day key	Year key, month name, day name	Yes	No
Table 3-3	Day	Day key	Year key, month name, day name	Yes	No

3.4.4 Relationships

A relationship describes how to create a relational query for multiple objects in the model. Without relationships, these objects are isolated sets of data. Relationships work in both directions. You often must examine what is happening in both directions to fully understand the relationship.

When importing metadata, Framework Manager can create relationships between objects in the model based on the primary and foreign keys in the data source or by matching query item names. You can create or remove relationships in the model so that the model better represents the logical structure of your business. After you import metadata, verify that the relationships you require exist in the project and that the cardinality is set correctly. The data source might have been designed without using referential integrity. Often, many primary and unique key constraints are not specified. Without these constraints, Framework Manager cannot generate the necessary relationships between fact tables and dimension tables.

The *cardinality* of a relationship is the number of related rows for each of the two query subjects. The rows are related by the expression of the relationship, which typically refers to the primary and foreign keys of the underlying tables.

The Cognos BI server uses the cardinality of a relationship in the following ways:

- ▶ To avoid double-counting fact data
- ▶ To support loop joins that are common in star schema models
- ▶ To optimize access to the underlying data source system
- ▶ To identify query subjects that behave as facts or dimensions. 1 to n cardinality implies fact data on the n side and implies dimension data on the 1 side

By default, Framework Manager uses Merise notation in relationship diagrams. Merise notation marks each end of the relationship with the minimum and maximum cardinality of that end. When you interpret cardinality, you must consider the notation that displays at both ends of the relationship. Possible end labels are as follows:

- ▶ 0..1 (zero or one match)
- ▶ 1..1 (exactly one match)
- ▶ 0..n (zero or more matches)
- ▶ 1..n (one or more matches)

The first part of the notation specifies the type of join for the relationship:

- ▶ An inner join (1) shows all matching rows from both objects.
- ▶ An outer join (0) shows everything from both objects, including the items that do not match. An outer join can be qualified as full, left, or right. Left and right outer joins take everything from the left or right side of the relationship respectively but take only what matches from the other side.

Users see a different report depending on whether you use an inner or outer join. Consider, for example, users who want a report that lists sales people and orders. If you use an outer join to connect sales people and orders, the report shows all salespeople, regardless of whether they have any orders. If you use an inner join, the report shows only salespeople who have placed orders.

Data in one object might have no match in the other object. However, if the relationship has a minimum cardinality of 1, an inner join is always used and these records are ignored.

Conversely, if all items match but the relationship in the model has a minimum cardinality of 0, an outer join is always used, although the results end up being the same as with an inner join. For example, the underlying table for one object contains a mandatory (non-NULL) foreign key for the other object. In this case, you must ensure that the data and cardinalities match.

The second part of the notation defines the relationship of query items between the objects.

Cognos BI supports both minimum-maximum cardinality and optional cardinality.

In a 0..1 relationship, 0 is the minimum cardinality, 1 is the maximum cardinality.

In a 1..n relationship, 1 is the minimum cardinality, n is the maximum cardinality.

A relationship with cardinality specified as being in a range from 1..1 to 1..n is commonly referred to as 1-to-n when focusing on the maximum cardinalities.

A minimum cardinality of 0 indicates that the relationship is optional. You specify a minimum cardinality of 0 if you want the query to retain the information on the other side of the relationship in the absence of a match. For example, a relationship between customer and actual sales can be specified as 1..1 to 0..n, in which case reports will show the requested customer information although there might not be any sales data present.

This means that a 1-to-n relationship can also be specified as any of the following items:

- ▶ 0..1 to 0..n
- ▶ 0..1 to 1..n
- ▶ 1..1 to 0..n
- ▶ 1..1 to 1..n

Use the Relationship impact statement in the Relationship definition dialog box to help you understand the cardinality of each side of a relationship.

When generating queries, Cognos software follows these basic rules to apply cardinality:

- ▶ Cardinality is applied in the context of a query
- ▶ A 1-to-n cardinality implies fact data on the n side and dimension data on the 1 side.
- ▶ A query subject may behave as a fact query subject or as a dimensional query subject, depending on the relationships that are required to answer a particular query.

3.5 Organizing relational models

When building a model, it is important to understand that there is no single workflow that will deliver a model suitable for all applications. Before beginning your model, you must understand the application's requirements for functionality, ease of use, and performance.

A well organized model helps users more easily find and understand the data in the model. It also makes the model easier for you to manage and maintain. By ensuring that a layer of model objects (either query subjects or dimensions) exists between the data source and the objects exposed to authors, you are better able to shield users from change.

A good approach is to import and work with small portions of metadata. In this way, determining how to build relationships and provide a clear view of the data for the users who will author reports using what you publish is easier.

A leading practice is to create several views, or layers, in the model. For example, depending on the complexity of your situation, you may not need a presentation view, and it might be possible to publish your business view for use by users.

Security can be defined in any of the views. The choice depends on your business requirements. For example, while security is typically applied in the business view, if you must prevent everyone from viewing an object, you add security to the object in the import view.

3.5.1 Data view

The data view, represented as a namespace, houses your data source-level query subjects and relationships. This view can be considered the import or physical layer.

Consolidating the creation of relationships in this view will take advantage of optimized queries in the query engine. If relationships are defined on model query subjects, then all associated joins and columns will be treated as a view and the Cognos BI server will not try to minimize the SQL for those items. For this performance reason, relationships should be defined on data source query subjects in the data view.

Avoid renaming data source query subjects or modifying the SQL in the data view to maximize the use of caching in the query engine. With minimal renaming the query engine

can further cache results, which improves performance by reducing the need to re-query your data source for metadata.

Create one or more optional namespaces or folders for resolving complexities that affect querying using query subjects or dimensional objects. To enable an OLAP experience over a relational database through a DMR package, there must be a namespace or folder in the model that represents the metadata with dimensional objects.

3.5.2 Business logic view

The business logic view, often referred to as the logical view, is where you can begin to simplify your data for presentation. This is done by denormalizing your data view by consolidating snowflake dimensions into one model query subject, or by hiding codes found in fact query subjects.

This view contains all of your model query subjects, and query items can be renamed to be more user-friendly. You can add further value with business logic by creating calculations and filters in this view.

Ideally, this view can provide an insulating layer from changes to the data source, so if the schema of the underlying database changes, there would be no need to modify any existing reports based on the business view or the presentation view. When table or other structure changes occur in the database, you can remap the model query subjects in the business view or the presentation view to new data source query subjects without any impact to report authors.

Security can be dynamically applied with calculations, session parameters, parameter maps and macros.

3.5.3 Presentation view

The presentation view can further simplify and organize your model. Depending on your audience, you can skip creating a presentation view as the organization and work done in the business view may suffice.

Using folders to group relevant items, filters, and shortcuts and relate them to the query items created in the business view allows you to provide a clear and simple view into the data for report builders.

3.6 Relational modeling for performance

This section provides guidance for creating models that will enable efficient, high performing reports. The impact of modeling techniques on SQL generation and caching is also described.

3.6.1 As view versus minimized SQL generation

There are two approaches to SQL generation that the Cognos BI server can employ when planning queries: *minimized SQL* and *as view*. The key difference between these two approaches is whether the query service is free to optimize SQL generation as it wants to or if there are constraints imposed by the Framework Manager modeler that must be respected.

With minimized SQL, the SQL generated by the Cognos BI server contains only the minimum set of tables and joins needed to obtain values for the selected query items. If you are modeling a normalized data source, you might choose to focus on minimized SQL because it can reduce the number of tables used in some requests and perform better.

With the *as view* SQL generation type, the SQL stays the same no matter which items in the query subject are referenced.

Minimized SQL example

Figure 3-2 shows four product tables as four data source query subjects that are joined to each other. The associated query items are combined into a model query subject, as shown in Figure 3-3.

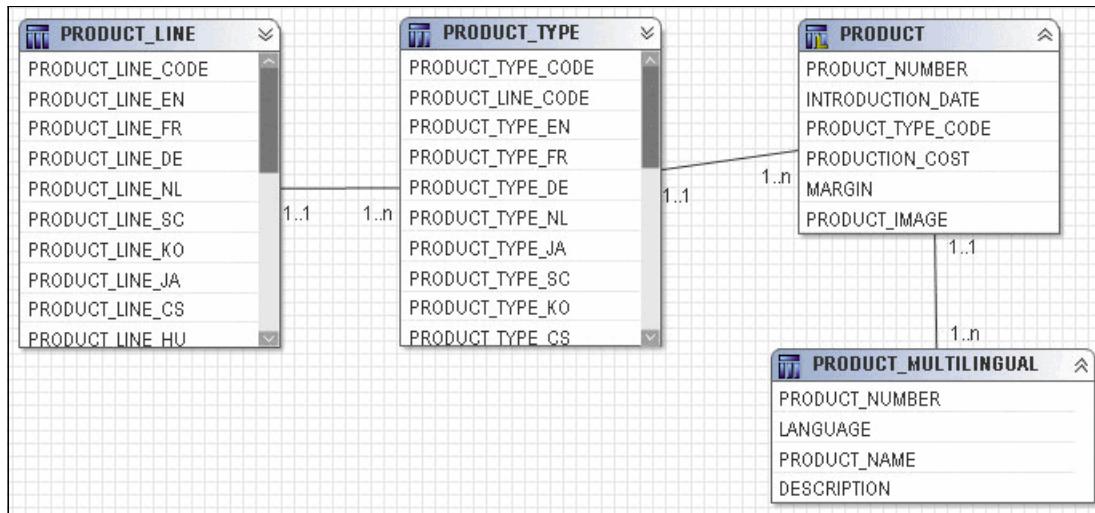


Figure 3-2 Product tables as four data source query subjects

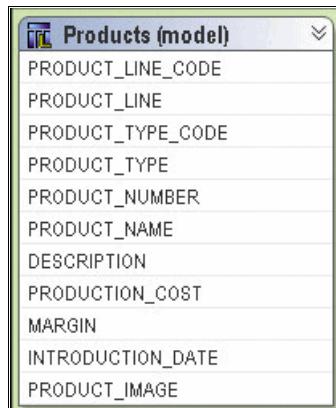


Figure 3-3 Model query subject combining all product query items shown in Figure 3-2

If you test the products model query subject depicted in Example 3-1 on page 38 as a whole, you can see that the four product tables are referenced in the *from* clause of the query, as shown in Example 3-1 on page 38.

Example 3-1 SQL generated for testing all query items

```
select
    PRODUCT_LINE.PRODUCT_LINE_CODE as Product_Line_Code,
    PRODUCT_LINE.PRODUCT_LINE_EN as Product_Line,
    PRODUCT_TYPE.PRODUCT_TYPE_CODE as Product_Type_Code,
    PRODUCT_TYPE.PRODUCT_TYPE_EN as Product_Type,
    PRODUCT.PRODUCT_NUMBER as Product_Number,
    PRODUCT_MULTILINGUAL.PRODUCT_NAME as Product_Name
    PRODUCT_MULTILINGUAL.DESCRIPTION as Product_Description,
    PRODUCT.INTRODUCTION_DATE as Introduction_Date,
    PRODUCT.PRODUCT_IMAGE as Product_Image,
    PRODUCT.PRODUCTION_COST as Production_Cost,
    PRODUCT.MARGIN as Margin
from
    gos1..gos1.PRODUCT_LINE PRODUCT_LINE,
    gos1..gos1.PRODUCT_TYPE PRODUCT_TYPE,
    gos1..gos1.PRODUCT PRODUCT,
    gos1..gos1.PRODUCT_MULTILINGUAL PRODUCT_MULTILINGUAL
where
    (PRODUCT_MULTILINGUAL."LANGUAGE" = N'EN')
and
    (PRODUCT_LINE.PRODUCT_LINE_CODE = PRODUCT_TYPE.PRODUCT_LINE_CODE)
and
    (PRODUCT_TYPE.PRODUCT_TYPE_CODE = PRODUCT.PRODUCT_TYPE_CODE)
and
    (PRODUCT.PRODUCT_NUMBER = PRODUCT_MULTILINGUAL.PRODUCT_NUMBER)
```

If you test only the product name query item, you can see that the resulting query uses only the product multilingual table, which is the only table that was required. This is the effect of minimized SQL generation shown in (Example 3-2).

Example 3-2 SQL generated that has been minimized

```
select
    PRODUCT_MULTILINGUAL.PRODUCT_NAME as Product_Name
from
    gos1..gos1.PRODUCT_MULTILINGUAL PRODUCT_MULTILINGUAL
where
    (PRODUCT_MULTILINGUAL."LANGUAGE" = N'EN')
```

Criteria that prevent minimized SQL generation

The following modeling scenarios will cause the corresponding query subject to function as a view, which means that the associated SQL will not be minimized:

- ▶ Modifying the default SQL in the expression of a data source query subject
- ▶ Adding filters or calculations to a data source query subject
- ▶ Defining relationships on a model query subject

To allow minimized SQL to be generated, relationships and determinants must be defined on data source query subjects and not model query subjects. When a relationship is defined on a model query subject, the resultant SQL changes because now it is considered a query and not only a folder of expressions.

Minimized SQL takes better advantage of database optimization than does complex SQL, so you are advised to avoid the three scenarios just described. However, there might be times when losing SQL minimization is necessary, such as when you require model query subjects with overriding relationships to control query execution paths, or you need to change the SQL on data source query subjects. This keeps the number of rows that are returned from this query subject stable, regardless of the elements that are reported from the query subject.

3.6.2 Security-aware caching

As detailed in Chapter 1, “Overview of Cognos Dynamic Query” on page 1, the Cognos BI query service employs a sophisticated cache management system to reuse objects captured from queries to data sources and use those objects to satisfy subsequent requests for that information. Caching can drastically reduce user wait times because it will always be faster for the query service to reuse an object it already has in memory than to go back to the data source for the information. The performance benefits of caching are clearly noticeable except for lighter queries where retrieving data from the database is a sub-second operation.

In cases where the underlying relational database responses appear instantaneous to users, it may be best to disable caching, either at the package level or the level of a particular query inside of a Report Studio report. Caching can be disabled either in Framework Manager (by deselecting the Allow usage of local cache check box governor and then republishing the package), or in Report Studio (by setting the Use Local Cache property of a query in the query explorer area to No). The DMR cache can consume considerable amounts of memory, which may initiate internal memory management operations that impact performance. For DMR and pure relational packages, disabling caching, either in the report or in the model, will ensure that only the portion of the query service’s Java memory that is required to execute the largest report is consumed, and nothing more. This keeps the query service’s memory footprint minimal. When the cache is enabled but is not being utilized, such as might be the case with complex security requirements, then resource consumption becomes sub-optimal and may degrade performance. For Report Studio reports, DMR caching is disabled only if one of the following conditions is present:

- ▶ The Use Local Cache property of the query object in the report is set to No
- ▶ The Allow usage of local cache governor in the Framework Manager model is deselected when the package is published

The query service’s caches for DMR and pure relational packages are security-aware in that, by default, cached metadata and data will not be shared between users with different security profiles. Users with the same security profiles are also similar in these respects:

- ▶ Sign-on information (user name and password) to the data source
- ▶ Expanded values of data source connection command blocks
- ▶ Model security defined in Framework Manager
- ▶ Expanded values of macros within in the queries used to populate the cache

Pure relational caching

Unlike dimensional queries, such as those for DMR packages (which, as of Cognos BI version 10.2, persist in memory until a manual or scheduled clear cache operation occurs), results from pure relational queries are retained in memory only while that same data source connection remains open. Query service data source connections have a default idle timeout of 300 seconds (5 minutes), but this can be configured to another value using the properties window of the query service in Cognos Administration, as explained in the product information center:

<http://ibm.co/19eGKCD>

Results that are cached from one user's request can be used for a different user's request under default settings, if both users have the same data security profiles.

DMR caching

To optimize your use of the DMR cache, it is important to understand its design, including what triggers the creation of a new cube cache or the reuse of an existing one.

The DMR cache stores members and cell values that combine to make a cube cache. The DMR cache for a certain package can have one or more cube caches in memory, as shown in Figure 3-4.

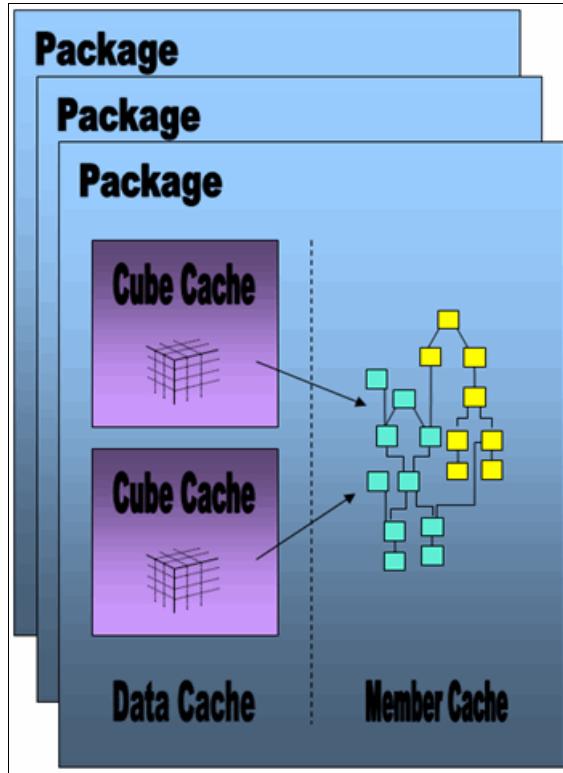


Figure 3-4 DMR cache composition

A cube cache primarily uses a member cache, which is shared by all the cube caches associated to the same package, and a cell value cache that is exclusively used by the cube cache.

Not all data requests will be cached because for some types of requests, caching provides no benefit. For example, very large batch reports that generate hundreds or thousands of pages of results consume more time writing results to the cache than are consumed by re-executing the same request multiple times. In addition, some queries perform well when executed on the underlying data source and may not benefit from using the cache. The query service automatically determines when to avoid writing results to the cache for queries that would not benefit from using it.

When using the cache, the query service caches each result in the context of all dimensions in the published package. Although many factors affect read/write performance from the cache, having a high number of dimensions negatively affect cache performance. Therefore, a good practice is to limit the choice of dimensions in a package to those that are required to satisfy the business requirements. This results in a noticeable performance gain for some situations.

Elements that compose the key to a cube cache

A cube cache is populated on-demand as metadata and cell values are retrieved during the execution of reports. Before a new cube cache is created, the query service checks to see if it can reuse an existing cube cache. Each of the cube caches associated with a package has a key that controls when that cube cache can be reused. If a new request has a key that matches that of a cache in memory, then that in-memory cache will be used. Otherwise a new cube cache is created (assuming caching has not been disabled).

A DMR cube cache is secured by a composite key that requires matches on these elements:

- ▶ Package instance
- ▶ Data source connection
- ▶ Data source sign-on
- ▶ Resolved data source command block values
- ▶ Detail filters:
 - Pre-aggregation filters
 - Filters defined in model
 - Slicers defined in report
 - Model security
 - Prompt selections
- ▶ Resolved macro values
- ▶ Runtime locale

Publishing different *packages* or versions of the same package from the same model will create different cube caches. If a version of a package results in multiple cube caches, the metadata cache will be shared by all of those cube caches through the application of security context on the metadata. Publishing the same package multiple times without any changes to the model causes two cube caches to be created, one for each package instance. A cube cache is reused only if the request originates from the same package instance.

A package references one or more data sources and every report execution is associated with a *data source connection definition*. A data source connection definition is composed of the connection string and, optionally, command blocks. By default, a cube cache is re-used only if the connection string (as sent to the database) and any resolved command block values are identical. This condition is also used when the query service is determining if it can reuse an open connection to a data source.

Pre-aggregation *detail filters*, which are typically computed by the database, include both those defined in the report and those defined in the Framework Manager. If the detail filters applied when populating a cube cache are not the same list of expressions as are in a new request, the cube cache cannot be reused because the members, cell values, or the members and cell values might be different. Any *slicers*, which are also known as context filters, that are defined in a report are converted into pre-aggregation detail filters but are applied only to the query against the fact table used to get cell values; they are not applied when loading members.

A report, the model, or the report and the model might have *prompts* defined. Not all prompts are applied to the key of a cache; only the prompts directly referenced in a query or indirectly referenced from the model (embedded in a query item) affect the cube cache's key. Prompts that are not referenced at all in the query are ignored with respect to the cache.

A cube cache is reused only if the associated *macros* whose values are resolved to create the cube cache are the same as those resolved values in a new query. If the current query has

additional macros not present in the original query, the cube cache may be reused and those new macros are added to the cube cache's key.

If *object security* is defined in the Framework Manager model, it implies that certain users do not have authorization to all objects in the model. In this case, by default, the profile associated with the object security is added to the key that controls sharing of that cube cache.

Finally, a cache is reused only if the *runtime locale* used to create the cache is the same as the current request's runtime locale.

Governors that control cache sharing sensitivity

Framework Manager offers governors so you can prevent certain elements from affecting a cache's key. To ensure confidentiality, the default settings for these governors are the most restrictive possible. You can change these governor settings to allow for greater cache reuse, if your security requirements allow you do so. For example, users might have different data source credentials specifically because the database administrator wants to audit the queries that are submitted to the database such that there are no data authorization differences between these database user accounts. In such a scenario, you might want to set the *Cache is sensitive to DB info governor* to DB + Connection instead of the default value of DB + Connection + Signon.

Note: For more details, see the dynamic query mode governor section of the product documentation at this website:

<http://ibm.co/1761dJk>



Macros

This chapter provides guidance on the usage of macro expressions in the metadata modeling and authoring interfaces of IBM Cognos Business Intelligence (BI).

Macros allow the Cognos BI applications you develop to be dynamically customized to the needs of the user immediately before the associated queries are submitted to the database. With the help of macros, you can author a single report that addresses the requirements of many different business scenarios, instead of authoring separate reports for each scenario. The chapter also describes ways that you can employ macros, combining macros with session parameters and parameter maps, and provides a variety of advanced examples.

The chapter contains the following sections:

- ▶ 4.1, “Macros explained” on page 44
- ▶ 4.2, “Macro language” on page 45
- ▶ 4.3, “Parameter maps” on page 47
- ▶ 4.4, “Session parameters” on page 49
- ▶ 4.5, “Advanced examples” on page 51

4.1 Macros explained

Macros are fragments of code that you can insert in the expression editor interfaces of Cognos BI, including within the Select statement that defines a query subject. Macros can include references to session parameters, parameter maps, and parameter map entries. For example, you can use the language session parameter to show only the data that matches the language setting for the current user.

Several facts about macros are summarized in the following list:

- ▶ Macros can give significant performance improvements in some reports.
- ▶ Macros show up in expressions as the text between two number sign (#) characters.
- ▶ An expression can contain more than one macro.
- ▶ Macros are expressions that are evaluated during query planning in such a way that the macro has been fully expanded before query execution.
- ▶ Using macros in appropriate places allows the application to be more flexible.
- ▶ There are several non-expression areas in Framework Manager where macros can be used.
- ▶ You can use macros in the data source connection command blocks defined in Cognos Administration.

The expression editors that are part of Report Studio and Framework Manager have a collection of functions that are categorized as macros. Figure 4-1 shows the macro tab within the expression editor of Report Studio.

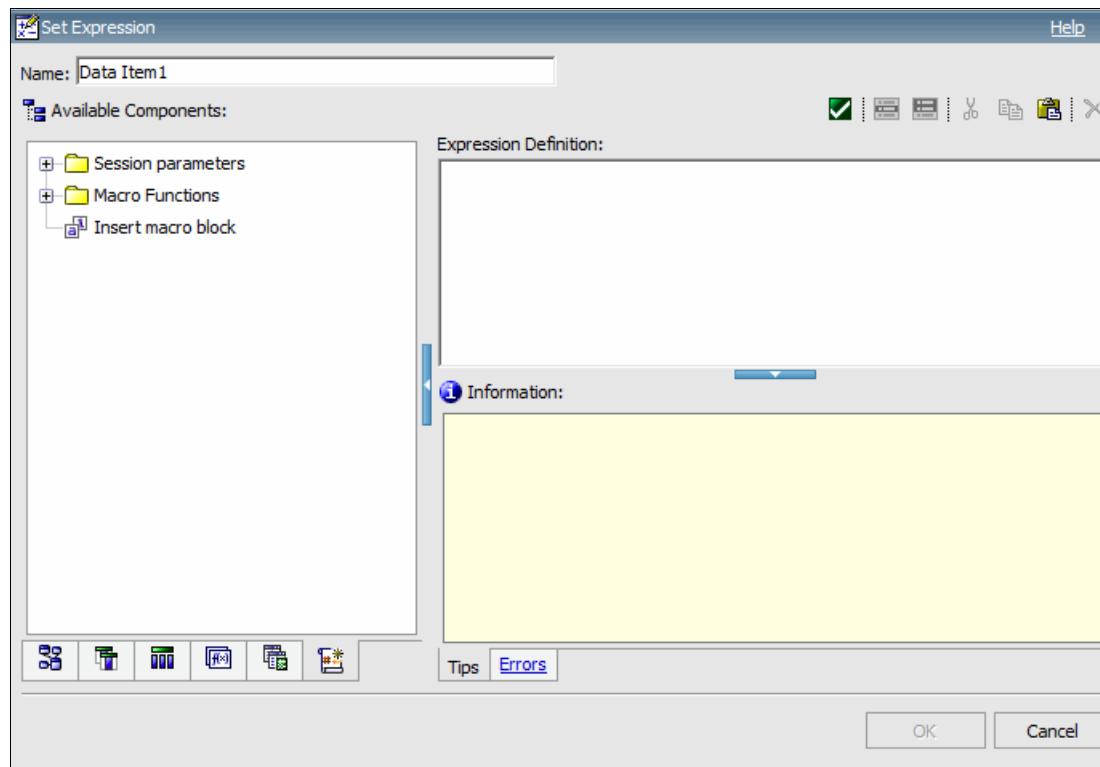


Figure 4-1 Macro tab within the expression editor of Report Studio

In working with macros, you might find that writing them is sometimes easier than reading them. Describing them accurately with comments will significantly help the next user to understand the intent of the macro.

You can use macros in a variety of ways:

- ▶ They can be inserted in SQL, as the following example shows:
`Select * from Country where Country.Name = #\$myMap{\$runLocale}#`
- ▶ They can supply an argument to a stored procedure query subject. If a value is not hard-coded for the argument, the stored procedure query subject can be used to return different data.
- ▶ They can be inserted in expressions such as calculations and filters. This filter is an example:
`[gosales].[Sales staff].[Staff name] = #\$UserLookUpMap{\$UserId}#`
- ▶ They can be used to dynamically complete the properties of a data source query subject. This enables different users to supply different connection information and thus access different data sources. The properties that can contain macros are Content Manager Datasource, Catalog, Cube, and Schema.

This is an example using the Content Manager Datasource property:

`#\$DataSourceMap{\$UserId}#`

- ▶ They can be used as a parameter wizard. When used in this context, parameters can reference other parameters, as in the following example:

`Map1, Key = en-us, Value = #\$myMap{\$UserId}#`

4.2 Macro language

This section explains the syntax to follow when writing macro expressions and presents some options you can use inside your macro expressions.

4.2.1 Operator

The macro language has only one operator, the plus sign (+) character, which is used to concatenate two strings. So the following example resolves to the value abcxyz:

`# ‘abc’ + ‘xyz’ #`

4.2.2 List separator character

The macro language recognizes both the comma (,) and the semicolon (;) characters as list separators. This is independent of any locale setting, as in the following example:

`# array (‘a’ , ‘b’ ; ‘c’) #`

4.2.3 Functions

All function names are case-insensitive. Only alpha characters are used in the names of functions. Some function names are short. Macro functions are used more by programmer-type report authors than casual report authors.

The expression editors in Report Studio and Framework Manager have a collection of functions that are categorized as macros and are displayed for drag-and-drop use. All of these macro functions have screen tips with examples.

4.2.4 Comments

Use comments to explain macros for other individuals who will use them. Adding comments is useful because they help to more easily maintain and support models. There are two rules for comments:

- ▶ Any text between the /* and */ strings, including new lines, is considered a comment.
- ▶ Any text between the // string and the end of a line is considered a comment.

The macro expression in Example 4-1 resolves to the value 2012 and demonstrates how comments help other users understand the intent of an expression.

Example 4-1 Commenting a macro expression

```
# // a macro is used to get the previous year
timestampMask(          // 3: extract the year portion
    _add_years(          // 2: subtract one year
        $current_timestamp, // 1: 2013-01-29 22:39:14.135-05:00
        -1),
    'yyyy')
#
```

4.2.5 Simple case construct

The case construct is used in programming to identify different sets of instructions corresponding to various conditions. The case macro construct allows you to specify values or functions to be returned under different conditions or cases.

Using the simple case construct in a macro in combination with the prompt function is sometimes challenging. Example 4-2, Example 4-3 on page 47, and Example 4-4 on page 47 show three examples of employing the simple case construct.

Example 4-2 Macro prompt with token data type

```
# // example 1
case prompt('option', 'token')
when 3 then '[gos1].[PRODUCT_LINE].[PRODUCT_LINE_CODE] > 3'
else '[gos1].[PRODUCT_LINE].[PRODUCT_LINE_CODE] is not null'
end #
```

Example 4-3 Macro prompt with unspecified data type

```
# // example 2
case  substitute("""", "", substitute("""", "", prompt('option')))
when 3 then '[gos1].[PRODUCT_LINE].[PRODUCT_LINE_CODE] > 3'
else '[gos1].[PRODUCT_LINE].[PRODUCT_LINE_CODE] is not null'
end #
```

Example 4-4 Macro expecting a specific input value

```
# // example 3
case  prompt('option')
when "3" then '[gos1].[PRODUCT_LINE].[PRODUCT_LINE_CODE] > 3'
else '[gos1].[PRODUCT_LINE].[PRODUCT_LINE_CODE] is not null'
end #
```

The prompt macro function returns a string by default. The entered value is surrounded by single quotation marks, which makes it useful in most expressions. So if the user enters the value abc, the default result will be 'abc' in this context.

In Example 4-2 on page 46, the data type of the prompt function is specified as *token*. The response will not be surrounded by single quotation marks. The literal 3 is used in the *when* clause and will match the user-entered value 3. There are circumstances when the token should not be used to prevent SQL injection, but that is not the case here.

In Example 4-3, the data type of the prompt function is not specified and thus defaults to *string*. The code that deals with the quoted return value of the prompt function removes the single quotes at the beginning and end of the response.

In Example 4-4, the *when* clause specifies the value 3. The easiest way to do this in the macro language is to surround the value with double quotation marks, as in the following example. This value will match the user-entered value 3:

"'3'"

4.3 Parameter maps

Parameter maps are objects that store key-value pairs. Parameter maps are similar to data source look-up tables. Each parameter map has two columns, one for the key and one for the value that the key represents.

Parameter maps can be defined in Framework Manager in various ways:

- ▶ Manually enter them as name-value pairs
- ▶ Load the name-value pairs from a file
- ▶ Base them on query items in the current model

To modify the parameter map, you can export the map values to a file, perform any additions or modifications, and then import the map values back into Framework Manager. This is especially useful for manipulating large, complex parameter maps.

Normally, parameter map keys must be unique so that the query service can consistently retrieve the correct value. Do not place quotation marks around a parameter value. You can use quotation marks in the expression in which you use the parameter.

You can create a parameter map that is based on query items with different values associated with the same key. For example, consider the parameter map definition for a parameter map named PLC2PTName, which is shown in Figure 4-2. This parameter map can be referenced in the following macro:

```
#sq(join('---', @PLC2PTName{'1'}))#
```

It returns the following result:

```
'Cooking Gear---Lanterns---Packs---Sleeping Bags---Tents'
```

Note that the at (@) symbol in @PLC2PTName indicates that an array of values is to be returned.

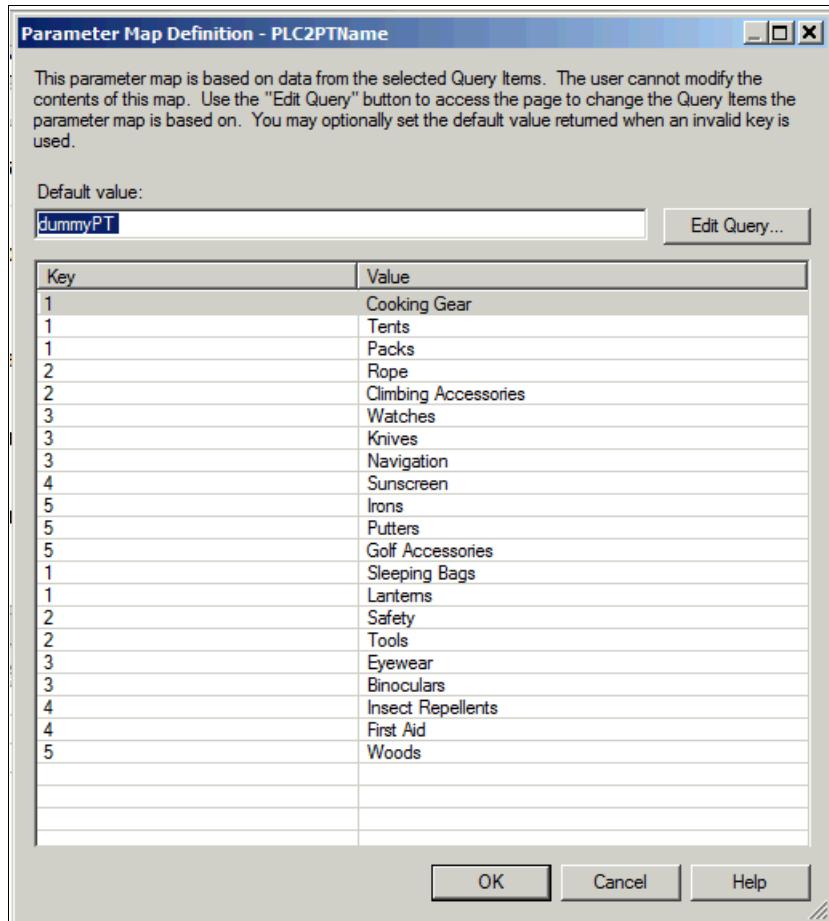


Figure 4-2 Parameter map with multi-valued keys

The value of a parameter can be another parameter. However, you must enclose the entire value between number sign (#) characters. The limit when nesting parameters as values is five levels.

When you use a parameter map as an argument to a function, you must use a percent sign (%) character instead of a dollar sign (\$) character.

Do not base a parameter map on a query item or table with a large result set (50,000 rows or more). Each time you use the parameter map in an expression or in SQL, the query service executes the large query and performance is slowed. Parameter maps should be used for smaller lookup tables only.

4.4 Session parameters

A session parameter is a variable that Cognos BI associates with a session. For example, user ID and preferred language are both session parameters. Because session parameters are key value pairs, you can think of each session parameter as an entry in a parameter map. You use a session parameter in the same way that you use a parameter map entry, although the syntax for session parameters is slightly different.

There are two types of session parameters: environment and model. Environment session parameters are predefined and stored in the Cognos BI content store database. By default, the following session parameters are displayed in Framework Manager:

- ▶ `runLocale`: Returns the code for the current active language in Framework Manager. The model content is shown in this language. You can change the active language at any time for your current session only. In future sessions, the model continues to open in the design language.
- ▶ `account.defaultName`: Specifies the name of the current user as defined in the authentication provider (for example, user's first and last name). If you log on anonymously, you will see `Anonymous`.
- ▶ `account.personalInfo.userName`: Specifies the user ID used to log on to Cognos BI. If you log on anonymously, you will not see this parameter.
- ▶ `current_timestamp`: Specifies the current date and time.
- ▶ `machine`: Specifies the name of the computer where Framework Manager is installed.

If your authentication source supports other parameters and you entered information about them in the authentication source, you can use other session parameters, such as `account.personalInfo.email` or `account.personalInfo.surname`.

Figure 4-3 depicts some of the session parameters shown in the Insertable objects pane of the expression editor in Framework Manager.

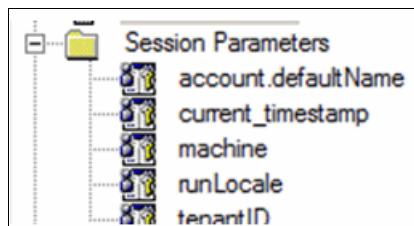


Figure 4-3 Session parameters shown in Framework Manager

Additional session parameters are available in Report Studio. Similar to the parameters in Framework Manager, the session parameters in Report Studio give access to information about the report, such as report start time, report name and report path.

Table 4-1 shows session parameters that are available to report authors but do not appear in the Report Studio user interface.

Table 4-1 Additional session parameters not shown in Report Studio interface

Name	Sample Value
contextID	/content/package[@name='gosales']/report[@name='a_macro_session_parms'];reportRender_Request;
reportPath	/content/package[@name='gosales']/report[@name='a_macro_session_parms']
REMOTE_ADDR	127.0.0.1
HTTP_HOST	localhost:81
queryName	Query1
report	a_macro_session_parms
startTime	2013-01-31T18:21:29.455Z
modelPath	/content/package[@name='gosales']/model[@name='model']

You can define additional parameters by using model session parameters. Model session parameters are stored in a parameter map named _env. They are set in the project and can be published with a package. Model session parameters must have their values set within the scope of objects in the Framework Manager model. The scope can include the use of existing environment session parameters, and also static values.

You can map user attributes from your LDAP authentication provider into new session parameters. To configure this, you must add these attributes as custom properties for the LDAP namespace in Cognos Configuration. For the procedure, see the product documentation at the following location:

<http://ibm.co/17qQQf0>

Each session parameter must have a name and a default value. You can define an override value to test the results that the value returns. The override value is valid only when you have the model open, and it is not saved when you save the model. If no override value exists, the query service uses the default value when it executes a query that contains a session parameter.

The following rules, in addition to others, govern the use of parameters:

- ▶ All possible return values must have the same data type.
- ▶ Only one value can be defined.

4.5 Advanced examples

This section describes some advanced ways to use macros effectively, all of which have been used by Cognos BI customers in the past. The intent of these examples is to give you inspiration to solve other problems.

4.5.1 Member unique name for next year

This is an example of creating a member unique name (MUN) based on the current year. Generating a MUN is particularly useful for drill-through or master-detail relationship-based applications when relating pure relational queries to OLAP queries.

OLAP sources organize data into dimensions. These dimensions contain hierarchies. The hierarchies contain levels and the levels contain members. An example of a dimension is Locations. A Locations dimension may contain two hierarchies: Locations by Organization Structure and Locations by Geography. Either of these hierarchies may contain levels such as Country and City.

Members are the instances in a level. For example, New York and London are members in the City level. A member may have multiple properties, such as Population, Latitude, and Longitude. Internally, a member is identified by a Member Unique Name (MUN). The method by which a MUN is derived depends on the cube vendor. When authoring reports, referencing a member through its MUN will typically perform faster than referencing a member through dimensional (MDX) functions.

The generated MUN for this example needs to be in the format shown in Example 4-5, which is based on the Great Outdoors Warehouse sample database that is included with all Cognos BI products.

Example 4-5 Example member unique name (MUN)

[Great Outdoors].[Years].[Years].[Year]->:[PC].[@MEMBER].[20040101-20041231]

The [20040101-20041231] section of the MUN shown in Example 4-5 identifies the member that represents data for all the days in 2004. This MUN can be generated using the date at run time with the macro expression shown in Example 4-6.

Example 4-6 Macro expression generating MUN dynamically based on current time

```
#  
'[Great Outdoors].[Years].[Years].[Year]->:[PC].[@MEMBER].[  
+  
timestampMask(_add_years($current_timestamp,-8),'yyyy')  
+  
'0101-'  
+  
timestampMask(_add_years($current_timestamp,-8),'yyyy')  
+  
'1231]'  
#
```

4.5.2 Turning promptmany result into a rowset

The result of the macro function *promptmany* is a single value, not an array of values. This limitation is a side effect of needing to remain compatible with macros that were written before the array data structure was introduced.

Therefore, consider the following expression:

```
# join ( '*' , promptmany('pp', 'string') )#
```

When the values for parameter pp are aa and bb, then the result of this expression is as follows:

```
'aa';'bb'
```

Notice that the values are surrounded by quotation marks and separated with a semicolon. The semicolon is the typical way that the *promptmany* function generates the list separator. If the macro had been defined for an SQL statement, then the separator would have been a comma instead of a semicolon. To get the result 'aa'*'bb' you can use the macro expression shown in Example 4-7.

Example 4-7 Macro expression joining string values

```
#  
join('*' , // 3 -> string 'aa'*'bb'  
    split(';',// 2 -> array with 2 elems: 'aa' and 'bb'  
          promptmany('pp','string'))// 1 -> 'aa';'bb'  
    )  
)  
#
```

Example 4-8 displays the expression for a data source query subject that takes the response to the *promptmany* macro function and transposes it into the rows of an inline values clause.

Example 4-8 Macro expression transposing inputted values into rows of a result set

```
with  
inputSet as  
(select * from ( values  
#  
'( ' +  
join ( ' ),( ' ,  
split ( ',' , promptmany ('setofdata') ) // split on , not on ;  
)  
+ ' )'  
#  
) T ( C1 ) )  
select  
inputSet.C1 as C1  
from  
inputSet
```

Assuming that the values for the setofdata parameter from Example 4-8 on page 52 are the strings FL, NY, BLAH and JAH, the macro will expand to what is shown in Example 4-9.

Example 4-9 Expanded result of the macro expression in Example 4-8 on page 52

```
with
inputSet as
(select * from (
values
( 'FL' ),( 'NY' ),( 'BLAH' ),( 'JAH' )
) T ( C1 ))
select
inputSet.C1 as C1
from
inputSet
```

You can apply this technique in various circumstances, such as in part of a filter or by combining it using an **EXCEPT** operation with another query that is compatible with the **union** operation. Bear in mind, however, that this technique is using a row constructor in a **select** statement, which IBM DB2 supports but many other database vendors may not. If your vendor does not support such processing, then the Cognos BI server performs it, although this action is at a cost to performance.

The split/join macro functions do not allow you to define different leading and trailing character strings (such as ‘timestamp(‘and ‘)’) so ensure that you embed the appropriate repeating text in the join, as shown in Example 4-10.

Example 4-10 Macro expression with join and split functions

```
column in (
#
'timestamp('' +
join ( '''),timestamp('',
split (',', 'abc,def,ghi,jkl' )
)
+ '')'
# )
```

The code in Example 4-10 expands to what is shown in Example 4-11 before any query is sent to the database.

Example 4-11 Expanded result of the macro expression in Example 4-10

```
column in ( timestamp('abc'),timestamp('def'),timestamp('ghi'),
timestamp('jkl') )
```

4.5.3 Dynamic column drill

The scenario in this example requires a list report with hierarchy and measure columns. The hierarchy column is a dynamic hierarchy using a data item expression. It shows measure values by different hierarchies based on the user's selection of a View by prompt. Changing these measure values can be accomplished either by using a parameter map lookup or a **case** statement macro expression.

Example 4-12 shows a parameter map-based solution that can be called upon by a calculation in the model or in the report whose expression is as follows:

```
#$pmap{prompt('View by', 'string')}#
```

Example 4-12 Parameter map-based solution

```
parameterMap : pmap
    default: [Provider].[Provider].[Provider].[Provider]
    key: Provider      entry: [Provider].[Provider].[Provider].[Provider]
    key: Practitioner  entry: [Practitioner].[Practitioner].[Specialty].[Specialty]
```

Cognos BI version 10.2 introduced support for a **case** statement macro function. An expression such as the one shown in Example 4-13 can be employed instead of the parameter map.

Example 4-13 Expression to use instead of a parameter map

```
# case prompt('View by', 'string')
when 'Provider'
    then [Provider].[Provider].[Provider].[Provider]
when 'Practitioner'
    then [Practitioner].[Practitioner].[Specialty].[Specialty]
else ([Provider].[Provider].[Provider].[Provider])
end
#
```

Alternatively, you can use the macro expression shown in Example 4-14.

Example 4-14 Alternate macro expression

```
# case prompt('View by', 'token')
when 'Provider'
    then [Provider].[Provider].[Provider].[Provider]
when 'Practitioner'
    then [Practitioner].[Practitioner].[Specialty].[Specialty]
else ([Provider].[Provider].[Provider].[Provider])
end
#
```

Note the use of the data type *token* in the second argument to the prompt function. This is used to match against the simple strings in the when clauses. The default data type is 'string', which results in a value surrounded by single quotation marks that would not match the values 'Provider' or 'Practitioner' in the when clauses. The values in the when clauses must be changed to "'Provider'" and "'Practitioner'" (notice the quotation marks).

The equivalent macro expression using the data type *string* is shown in Example 4-15.

Example 4-15 Macro based solution with string data type in prompt

```
# case prompt('View by')
when "'Provider'"
    then [Provider].[Provider].[Provider].[Provider]
when "'Practitioner'"
    then [Practitioner].[Practitioner].[Specialty].[Specialty]
else ([Provider].[Provider].[Provider].[Provider])
end
#
```

4.5.4 Filtering for internal and external customers

A common scenario is a report that must handle various types of parameters depending on the privileges of the user who is running the report. This example involves a session parameter that has the pattern 1234_FMUSER (for external users that have logged in) or SYSADMIN01 (for internal users that have logged in). For internal users, the application must prompt the user for the customer number. For external users, the application must restrict all data to just the leading digits that represent the customer number in the database, so the user is not prompted for a customer number.

These requirements can be satisfied by creating an embedded filter in a query subject with the expression shown in Example 4-16.

Example 4-16 Macro expression with case construct based on session parameter

```
# '[gosales_8_2].[CUSTOMER_HEADER_10_2].[CUSTOMER_NUMBER] = ' +
// isolate the customer number from 123_FMUSER or issue a prompt
// when there is no underscore in session parameter pc1
case join('', grep('_', array($pc1)))
when '' then
    // session parameter pc1 has no underscore
    '?start_number? '
else
    // isolate the number before the underscore e.g. 123 from 123_FMUSER
    'cast (' +
    join('', grep('!\|', split('_', join('_', split('_', $pc1) ) ) ) ) +
    ', varchar(10)) '
end
#
```

Other examples

Additional examples of using macros, session parameters, and parameter maps are in the product documentation at the following location:

<http://ibm.co/17qUwhI>



Report authoring

IBM Cognos Business Intelligence (BI) is an integrated business intelligence suite that provides a wide range of functionality to help you understand your organization's data. Everyone in your organization can use Cognos BI to create (or *author*) and view business reports, analyze data, and monitor events and metrics so they can make effective business decisions.

You use the web-based authoring interfaces of Cognos BI to create and update your reporting and analysis applications. This chapter describes considerations for authoring high-performing applications that satisfy analytical requirements of users.

The chapter contains the following sections:

- ▶ 5.1, “Authoring interfaces” on page 58
- ▶ 5.2, “Processing report executions” on page 59
- ▶ 5.3, “Database functions” on page 60
- ▶ 5.4, “Dimensional and relational reporting styles” on page 61
- ▶ 5.5, “Suppression” on page 62
- ▶ 5.6, “Dimensional summaries” on page 63
- ▶ 5.7, “Advanced features in Report Studio’s Query Explorer” on page 64

5.1 Authoring interfaces

Cognos BI integrates many business intelligence activities in one web-based solution. Table 5-1 outlines how the primary Cognos BI user interfaces can help you do your job. The two most popular authoring interfaces for new applications, Cognos Workspace Advanced and Report Studio, are described in more detail later in this section.

Table 5-1 Authoring interfaces in Cognos BI

Interface	Activity
Cognos Workspace	Create and share interactive dashboards
Cognos Workspace Advanced	Author simple reports and explore your data
Report Studio	Author professional reports intended for a wide audience
Event Studio	Manage events and alerting
Query Studio	Query your data on an ad hoc basis
Analysis Studio	Explore your dimensional data

Note: For advanced techniques of Cognos BI report authoring, see the Reporting section of the Business Analytics developerWorks page:

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

5.1.1 Cognos Workspace Advanced

Cognos Workspace Advanced is used for advanced data exploration and authoring simple reports.

When you are in Cognos Workspace and want to perform deeper analysis and report authoring, you can seamlessly graduate to Cognos Workspace Advanced, where you can do more advanced data exploration, such as adding more measures, conditional formatting, and advanced calculations. You can also launch Cognos Workspace Advanced directly from the Cognos Connection portal.

With Cognos Workspace Advanced, you can create reports with relational or dimensional data sources, and then show that data in lists, crosstabs, and charts.

See the information center for Cognos Workspace Advanced version 10.2.1:

http://pic.dhe.ibm.com/infocenter/cbi/v10r2m1/nav/3_7

5.1.2 Cognos Report Studio

With Cognos Report Studio, report authors can create, edit, and distribute a wide range of professional reports. Report Studio is ideal for certain kinds of reports:

- ▶ Reports intended for wide audiences
- ▶ Reports that will require maintenance as requirements and data change
- ▶ Reports whose appearance must be controlled in fine detail

With Report Studio, you can create any reports that your organization requires, such as invoices, statements, and weekly sales and inventory reports. You can also author sophisticated, multiple-page, multiple-query reports against multiple data sources.

Report Studio provides powerful functionality, such as bursting, prompts, maps, and advanced charting, and provides many ways to customize reports. Report Studio is also where you can author Cognos Active Reports, which enable an interactive analytics experience in a self-contained application for browsing and exploring data offline.

See the information center for Report Studio version 10.2.1:

http://pic.dhe.ibm.com/infocenter/cbi/v10r2m1/nav/3_5

5.2 Processing report executions

When you create a report, you are actually creating an XML report specification. The report specification defines the queries that are used to retrieve data and the layouts and styles that are used to present the data. For simplicity, the report specification is named the same as the report. Figure 1-1 on page 4 presents the workflow of communications when a report is run.

Running reports and performing analysis requires the processing of data that consists of computing calculations, joins, unions, filters, grouping, sorting, and other operations on data. The Cognos BI query service, which operates in a Java process, and the underlying data source are responsible for this data processing.

Further processing is required to render the processed data in the requested format such as HTML, Adobe PDF, Microsoft Excel, and so on. The report service and batch report service, which operate within BI BusTKServerMain processes, are responsible for this rendering activity, which includes rounding and all other data formatting.

When users run reports, users must wait for both data processing and rendering processing to be completed before the output is displayed. This combined processing time can occur in less than one second or it can take considerably longer, depending on the complexity of the operations required and the amount of data involved. Understanding the processing that occurs when reports are run will help you minimize user wait times.

5.2.1 Local and database processing

Cognos BI supports answering a rich variety of analytical questions. Although the underlying database may be able to answer some of these questions, many complex questions require the Cognos server to compensate by performing further data processing locally.

Data processing performed by the Cognos BI server is referred to as *local processing*. Data processing performed by the database server is referred to as *database processing*.

Except for when the requested data already resides within the query service's in-memory cache, database processing is typically faster than local processing for the following reasons:

- ▶ Database processing can use indexes and other accelerators defined on the stored data.
- ▶ Database processing occurs closer to where the data is stored.
- ▶ Database processing reduces the amount of data transferred from the database server to the Cognos BI server.

To the extent possible, the Cognos BI server exploit the query language supported by the database. For databases with a limited query language, Cognos BI still allows users to ask questions of their business data without being restricted by the limitations of their database. This is because Cognos BI supports many types of queries that many databases do not support, which enables a consistent experience with the Cognos software regardless of what technology is storing the data. Users can create reports that require a particular type processing even if the underlying database does not support that processing; if necessary, the Cognos BI server will do that processing. As explained previously, there is typically a performance cost to local processing, usually because unnecessarily large amounts of data are sent from the database to the Cognos BI server. For these reasons, performance is best when as much processing as possible is delegated to the database.

IBM InfoSphere® BigInsights™, an enterprise class derivative of Apache Hadoop, can help illustrate the contrast between local processing and database processing. BigInsights 2.1 includes an Apache Hive 0.9 interface, but for performance reasons, its Big SQL interface is the preferred way of interoperating Cognos BI and BigInsights. When the Cognos BI server connects to BigInsights through the Big SQL interface, it uses industry-standard SQL, which is much richer than Hive query language. This means Cognos can derive more benefit from the massive parallel processing of a BigInsights cluster than from connecting through the Hive interface. For example, windowed aggregates is a type of processing where aggregation is performed on a certain window of data, such as for a particular time period. The concept of windowed aggregates is common in business intelligence scenarios, so, naturally, the Cognos BI server supports it locally. However, to optimize performance, Cognos will submit the windowed aggregate processing to any database that supports it. When windowed aggregates are required and Hive 0.9 is being used, the Cognos server must compute those aggregates itself. But if Big SQL is being used, the Cognos server can rely on the BigInsights cluster to compute the windowed aggregates.

5.3 Database functions

The authoring interfaces that are independent of data source in Cognos BI offer users a consistent experience across all supported data store technologies. Supporting those authoring interfaces is a robust query engine that interprets user gestures and report specifications and translates them into queries that are tailored to the technology being used. The Cognos BI server optimizes the queries it generates for performance, which typically involves submitting as much as possible of the necessary data processing down to the database.

When you connect your Cognos BI server to an analytic data store such as SAP HANA, Oracle Exadata, or one of the IBM PureData™ systems, the Cognos server detects which version of the database software has been loaded onto the system. It then employs the native query functions that are supported by that software, such that the data appliance does as much of the processing as possible and only a minimized result set is returned. So if you are using a 2013 version of IBM PureData for Analytics, the Cognos software knows that it is interoperating with IBM Netezza® version 7 software and will utilize native Netezza 7 functions in its queries.

Most of the authoring interfaces of Cognos BI, including Report Studio and Cognos Workspace Advanced, offer expression editors. An expression is any combination of operators, constants, functions, and other components. You build expressions to create calculation and filter definitions.

When the query service parses an expression that an author entered into a report, it recognizes scalar functions from its library and then determines if it needs to process a

particular function locally or if it can re-map that function to an equivalent native database function. When an unrecognized function is encountered, the query service submits it to the database. If it is a valid function on the database, it is processed successfully, otherwise the error that is returned from the database is written into the Cognos BI server logs.

You typically do not have to import user-defined scalar database functions into the Framework Manager model before you can use them in the expressions you compose. An exception is when the database requires you to qualify the reference and does not provide an ISO-SQL-style search path with which to locate non-qualified functions, for example HOTEL.BAR('HELLO') instead of BAR('HELLO').

5.4 Dimensional and relational reporting styles

There are two distinctive report authoring styles in Cognos BI: *dimensional style* and *relational style*.

The relational reporting style is used for the pure relational analytics option, which is explained in 1.4.1, “Pure relational analytics” on page 7. This style is often used for lists because lists are consistent with the structure of tables in a relational database. In relational reporting, you summarize data by using headers and footers in lists, summary functions, and *within detail* aggregation. You refine data in relational reporting with summary or detail filters.

The dimensional reporting style is employed for the dimensionally modeled relational (DMR) option and online analytical processing (OLAP) data sources. Dimensional data is best represented by crosstabs, maps, and charts. This data is shown in dimensions, hierarchies, levels, and members. In dimensional reporting, you summarize or roll up data by using member summaries and *within set* aggregation. You refine or focus data by adding only the relevant members to the edge of a crosstab or to the context filter. You can also enable drilling up and drilling down in dimensional reports.

The preference is that a single reporting style, either relational or dimensional, be used when developing a query. Unexpected results can occur if you mix the two styles within a query.

When working with DMR or an OLAP data source, the authors recommend that you not use relational functions, such as substring and concatenation functions, in any report that also contains a measure with the Aggregate Function property set to Calculated or Automatic. If you do so, you might encounter unexpected results. For example, some summaries are calculated using the minimum function instead of the aggregate function derived from the individual query items.

In the expression editor, an exclamation mark (!) that precedes a function indicates that the function is not naturally supported for that data source. In such cases, the Cognos BI server uses a local approximation for the non-supported function. Because an approximation is used, performance can be degraded and the results may not be what you expect.

Note: For more information about reporting styles, see the following pages in the product information center:

- ▶ Relational reporting:
<http://ibm.co/15VQwta>
- ▶ Dimensional reporting
<http://ibm.co/18BxLsK>

5.5 Suppression

Sparse data can result in crosstabs showing empty cells. For example, a crosstab that matches employees with products will show multiple empty rows for the revenue measure if the employee does not sell those products. A product that has no sales for a given quarter may result in a very large report with thousands of cells that contain no data. So suppressing rows and columns that contain only null values makes a report more concise and easier to read.

All authoring interfaces of Cognos BI offer suppression buttons on their toolbars. You can suppress rows or columns or rows *and* columns based on divide by zero, missing, and overflow values.

The time required to evaluate a query to determine which rows and columns contain only null values is mainly determined by the number of cells in the result set. Other factors such as the nesting levels on the axes and the use of complex calculated columns might also affect the time required.

The number of cells in a result set of a dimensional query is determined by the number of cross joins, which are the Cartesian product of member sets. The number of cross joins are calculated as follows:

$$\text{crossjoin } (\{a_1, a_2\}, \{b_1, b_2\}, \{c\}) = \{(a_1, b_1, c) \ (a_1, b_2, c) \ (a_2, b_1, c) \ (a_2, b_2, c)\}$$

Figure 5-1 presents an example where the resolved edge has $1 \times 1 \times 170 \times 818 \times 818 = 113,751,080$ tuples, or cells, to process.

<#Products#>	<#Product line#>	<#Product type#>	<#Product#>	<#Product details#>
				<#Product details#>
1	1	170	818	<#Product details#>
		<#Product#>	818	<#Product details#>
			818	<#Product details#>
				<#Product details#>
				<#Product details#>
				<#Product details#>

Figure 5-1 A crosstab containing five nested sets

The Cognos BI query service applies optimizations for suppression. For example, if a cell's expression evaluates to null, it does not have to be computed during query evaluation, which reduces the number of cells that need to be processed.

Avoid requesting large, sparse result sets to prevent long wait times while suppression is being applied. This will happen naturally if you work only with one meaningful view of your data at a time. The preferred approach to analysis is to remove any data that is not pertinent to your review before taking any step that expands the amount of data returned.

If you know which members will have the data of interest, explicitly reference or keep only those members in your report, particularly when a large majority of cells will be null otherwise. Likewise, if you are interested in only a slice of the data that involves hierarchies that do not need to be displayed in the report, add appropriate members to the Context filter or Slicer areas of the authoring interface.

To avoid the lengthy processing times required for large, sparse result sets, use a TopCount() function for dimensional reports or a rank() function for relational reports to show only the top values of interest.

Another method for suppressing null cells in reports is the use of filters to ensure that calculations take suppression into account. You can achieve better report performance because the filtering is done by the data source. For example, insert a set expression in your crosstab and use the following expression:

```
filter (descendants ([Set]) is not null)
```

If your crosstab includes three or more levels within the same dimension on an edge, use the following expression:

```
filter (descendants (currentMember([Hierarchy])) is not null).
```

If your crosstab includes nested sets, you can improve performance by filtering the sets using a cascading approach. For example, first filter the outermost (or highest nested level) set and then proceed inward to filter the remaining sets.

5.6 Dimensional summaries

You summarize data in reports to obtain totals, averages, and so on. All authoring interfaces of Cognos BI offer toolbar buttons to generate summaries.

Users analyzing OLAP data sources such as IBM Cognos Dynamic Cubes, IBM Cognos TM1, Microsoft Analysis Services, or Oracle Essbase typically want to see summaries of their detailed data. However, when applied to large volumes of data, summaries can be expensive to compute, and in extreme cases can slow response time to the point of becoming unusable. This is particularly true when the data is sparse and suppression is applied.

This section describes techniques to avoid performance problems from dimensional summaries.

Remove the summaries

The simplest technique is to remove the summaries if they are not necessary for analytical needs of users, or if shorter user wait times are more important than the availability of summary values.

Use parent members

In most dimensions, every non-leaf member is the rollup (the natural summary) of its children. In most cases, referencing a parent member can perform better than requesting a summary of that parent member's child members. If you reference the parent member in your reports, the value can come directly from the data source, but if you define a summary the value must be computed at run time. OLAP data sources typically optimize the rollups in each member for better performance, and in some cases store the pre-aggregated rollups of all non-leaf members.

Summaries are calculated at run time to ensure that the summary is correct even when the members that are displayed do not roll up into an accessible parent member. Avoiding such summaries in favor of using the desired parent member avoids the associated runtime performance costs.

You cannot use this technique if you need a summary set of members that is not a complete set of children, for example a TopCount() set. In such scenarios, the summaries must be computed on demand.

Use automatic summaries

When using the Summary toolbar buttons, requesting an automatic summary instead of an explicit summary (such as Total) allows the summaries, when possible, to be delegated to and optimized by the data source system, resulting in better performance. This optimization is especially useful when detail summaries are required, such as in a list report.

When using the expression editor, the function that computes automatic summaries is Aggregate().

You cannot use this technique if you need a summary set of members that is not a complete set of children, for example a TopCount() set. In such scenarios, the summaries must be computed on demand.

5.7 Advanced features in Report Studio's Query Explorer

Report Studio offers advanced reporting functionality that is not available in the other authoring interfaces of Cognos BI, such as the ability to create and modify queries using Query Explorer. Query Explorer provides an alternative way to modify existing reports or to author new reports. To enter Query Explorer while in Report Studio, place your mouse pointer over the **Query Explorer** button and then click **Queries**.

You can use Query Explorer for complex tasks and other actions that are difficult to do when in the regular report layout. Query Explorer can do the following functions, among others:

- ▶ Improve performance by changing the order in which items are queried from the database.
- ▶ Incorporate SQL statements that come from other reports or reports that you write.
- ▶ Create complex queries using union operations and joins.

This section provides details about the most useful Query Explorer features.

5.7.1 Reference queries

Use reference queries to control the order in which items are queried from the underlying data sources. This can help you to improve performance or generate complex computations in an order that is tailored to your needs.

To produce reference queries in Query Explorer, create the link between the child query and the parent query by dragging the parent query to a position on the right side of the child query. Figure 5-2 on page 65 presents two reference queries. In the first query, Query1 is the parent query and Query2 is the child query. In the second query, Query1 is the parent query and Query3 is the child query.

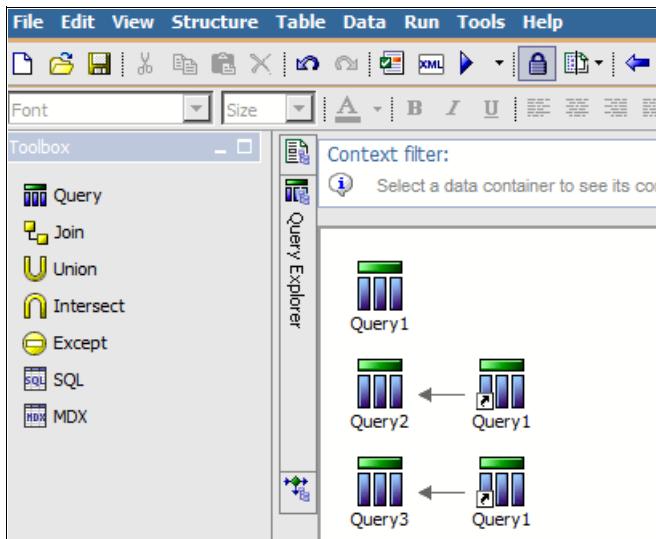


Figure 5-2 Two reference queries

With reference queries, you can ensure that a series of calculations and filters are applied in the order you want, because the parent query is always processed before the child query.

Reference queries also help you control caching and avoid fetching more data from the database, because child queries are processed from the cached data of parent queries (except when caching has been disabled). Using the example presented in Figure 5-2, with default settings, the query from Query1 will be submitted only one time, and Query2 and Query3 will be based on the cached result set from Query1.

When you create a child query in Report Studio, you can reference items only from its parent or from other queries. For example, if you add a filter to a child query, the only items that you can insert into the expression are items that exist in other queries defined in the report.

5.7.2 Union, intersect, and except queries

Create a *union* query to combine two or more queries into one result set. You can combine queries that use different data sources. For example, you can combine a query that returns data from a dimensional data source with a query that returns data from a relational data source.

An *intersect* query takes the results of two queries and returns only the rows that appear in both result sets. An *except* query evaluates the result set of one query and returns the rows that do not appear in a second query's result set.

In some scenarios, the union, intersect, and except operations can be delegated to the underlying data source. More typically, however, these types of set queries must be computed through local processing and therefore carry a cost in terms of performance. So, avoid these operations when requirements can be satisfied without them.

Note: For more information about union queries, see the following topic in the product information center:

<http://ibm.co/15rhtBN>

5.7.3 Join relationships

A join relationship joins two queries. To avoid the performance cost of local processing, join relationships are typically created in the IBM Cognos Framework Manager model. But if your requirement cannot be modeled in Framework Manager, you have the option to create the necessary join relationship in IBM Cognos Report Studio.

One reason to define a join relationship in Report Studio is so you can initiate the join on the aggregated values of either or both of the associated queries instead of on the corresponding detail records.

Note: For more information about join relationships, see the following topic in the product information center:

<http://ibm.co/1cDLN18>

5.7.4 Master detail relationships

A master detail relationship helps you deliver information that would otherwise require two or more reports. For example, you can combine a list with a chart. The list can contain product lines and the chart can show details for each product line.

Master detail relationships must appear in nested frames to produce the correct results. You can create a master detail relationship in two ways:

- ▶ Use a parent frame for the master query and a nested frame for the detail query.
- ▶ Associate a report page with the master query and use a data container, such as a list or crosstab, for the detail query.

You can use a master detail relationship to show data from separate data sources in a single report. However, the data sources must be contained in the same package.

For dimensional queries, master detail reports with a crosstab report object are optimized to use a single query whenever possible, rather than using a separate query for each report object instance. To reduce the required number of queries to the data source and therefore reduce execution time, the optimization combines the master query with the detail query in a crosstab. The data that appear in each detail report are now a subset of the result returned by the new combined query, instead of the result of a separate query with a detail filter to select the current value from the master.

For relational queries, the master detail relationship definition is used by default to filter the detail query for every record in the master query, resulting in multiple detail queries being issued to the underlying database. As the master query record set increases, the number of detail queries increases also, slowing overall report performance. As of Cognos BI version 10.2.1, there is a governor in Framework Manager called (DQM) Master-detail optimization. Use this governor to control whether detail query caching occurs for a relational master detail query. To minimize the amount of SQL execution against the database for detail queries, cache the detail query. For example, if you have 1,000 detail queries, then only one SQL execution will occur. By default, detail queries are not cached, so for 1,000 detail queries, 1,000 SQL executions occur.

Note: For more information about master detail relationships, see the following product information center page:

<http://ibm.co/1dMZ0xr>



Optimizing SQL for performance

The IBM Cognos Business Intelligence (BI) server generates Structured Query Language (SQL) queries to retrieve data from relational databases. Users must wait while the database responds to such queries. This chapter provides guidance for minimizing these wait times.

The chapter contains the following sections:

- ▶ 6.1, “Remember that less is faster” on page 68
- ▶ 6.2, “Make use of enforced and non-enforced constraints” on page 68
- ▶ 6.3, “Use indexes and table organization features” on page 69
- ▶ 6.4, “Review column group statistics” on page 69
- ▶ 6.5, “Avoid complex join and filter expressions ” on page 70
- ▶ 6.6, “Reduce explicit or implicit conversions” on page 71
- ▶ 6.7, “Minimize complexity of conditional query items” on page 71
- ▶ 6.8, “Review the order of conjunctions and disjunctions” on page 80
- ▶ 6.9, “Avoid performance pitfalls in sub-queries” on page 81
- ▶ 6.10, “Avoid unnecessary outer joins” on page 84
- ▶ 6.11, “Avoid using SQL expression to transpose values” on page 84
- ▶ 6.12, “Apply predicates before groupings” on page 86
- ▶ 6.13, “Trace SQL statements back to reports” on page 87

6.1 Remember that less is faster

The most important aspect to learn from this chapter regarding SQL queries is that *less is faster*. If all other factors are the same, a simpler SQL statement is satisfied in less time than a more complex SQL statement. Likewise, requests for more data take longer than requests for less data, all else being equal.

As reports are executed, the Cognos query service will plan SQL statements that it requires to obtain data from one or more relational data sources. The physical SQL statements that are generated are dependent upon the SQL semantics and data types supported by the underlying database. The complexity of the generated SQL statements can introduce performance costs both for the underlying data source and for the Cognos server when it needs to perform additional processing locally.

Cognos BI applications that are layered on operational databases frequently require complex joins and expressions to navigate through the data and present values in business terms. In contrast, applications that are layered on cleansed reporting structures, such as star schemas, can benefit from the data transformations applied by the publishing extract, transform, and load (ETL) processes. Reducing the complexity of the joins and expressions in queries can help the relational database management system (RDBMS) plan queries more efficiently and, in turn, reduce processor and memory consumption.

Cognos BI administrators can work with their database administrators to determine which SQL statements return a large number of rows where a small percentage of the row data is presented in a report. Although such SQL statements might not be complex or expensive for the RDBMS to process, they can result in large amounts of data being transferred to the Cognos BI server to be locally processed.

Many of the recommendations in this chapter are also common preferred practices that many RDBMS vendors suggest to improve runtime performance.

6.2 Make use of enforced and non-enforced constraints

Tables in a database can declare constraints that can be considered by the RDBMS query engine for strategies such as join eliminations, query rewrites, and expression optimizations. Primary key, unique key, and foreign key constraints (but not null and table constraints) can be declared for this purpose. Depending on the vendor, these constraints can be declared as either non-enforced or enforced. In a normalized table design including snowflake schemas, non-primary key columns are functionally dependent on the primary key.

To plan SQL statements for the RDBMS to process, the Cognos query service uses enforced constraints defined in a Framework Manager model, such as determinants and join relationships between query subjects. These Framework Manager objects are often created during one of the initial steps of creating a model, but more common is that they are manually defined by the Framework Manager modeler.

Enforced constraints can be defined in a Framework Manager model using join relationships between query subjects and determinants, and can be used during SQL planning by the Cognos query service as it plans SQL statements for the RDBMS to process. These Framework Manager objects are often created in one of the first steps when creating a Framework Manager model, but they are more commonly manually defined by the Framework Manager modeler.

A Framework Manager model can also be constructed on top of databases that expose application objects through SQL views. Those views should be reviewed by the database administrator with respect to the tables that the views reference, because the Framework Manager modeler might not be aware of those tables yet.

When an RDBMS does not support ISO SQL windowed aggregates, an SQL statement will likely be generated using two or more derived tables that include rows at different levels of grouping. The rows from the derived tables will be joined in the SQL statement with predicates (the grouping columns). If the database metadata does not use not null constraints, then the predicate must compare the columns to determine if they are equal in value or if they are both null values. These additional expressions can affect the performance in the RDBMS.

6.3 Use indexes and table organization features

A common challenge for a database administrator is to anticipate the ways that applications attempt to navigate the database. This includes which tables the queries will combine and which tables predicates will be applied against. Using a representative workload, the database administrator can review which tables are most frequently accessed and, in particular, which local set of columns is used to filter and group columns in tables.

Using that knowledge, the database administrator can usually identify indexes or table organization strategies that enable the database to more efficiently select the required rows. The candidate workloads must reflect any ad hoc analysis and exploration of data that can occur within an application. This is important when the database administrator is constrained in terms of what covering indexes or table organizations they can define, which might bias the solution toward the most frequent cases. For example, an application might predominantly categorize measures based on time, customer geography, and product perspectives for which the database administrator can optimize the table designs.

A Framework Manager model can also be constructed on top of databases that expose application objects through SQL views. Such views must be reviewed by the database administrator with respect to the expressions within the view or any projected query items about which the Framework Manager modeler might not be aware.

6.4 Review column group statistics

Using a representative workload, the database administrator must review any instances where predicates reference several columns of the same table, such as when data is filtered by Country, Country-Region, and Country-Region-City.

These local predicates allow the database administrator to consider using multi-column indexes that improve the performance associated with the predicates, and to gather relevant statistics to improve cardinality estimation.

Note: A review of column group statistics often identifies predicates with inefficient data types, such as character strings, as bottlenecks in query processing. To overcome this, user prompts in models and reports can be configured to display meaningful names (character strings) while more efficient data types, such as integers, are sent for processing by the database. Figure 6-11 on page 76 shows a Report Studio dialog where you set Use and Display values for a prompt. The Use values are what are computations use; the Display values are what users see.

6.5 Avoid complex join and filter expressions

The complexity of expressions in the `where` and `join` clauses of a SQL statement can impede planning for the RDBMS, query rewrites to materialized views, or other forms of query acceleration. This section describes two common types of complex expressions and explains several important factors to consider when using them.

6.5.1 Temporal expressions

In many applications, data is selected within a calendar context that is either designated by the user or is based on standard business periods, such as the current month or day. The input values define the range of data to select either in absolute terms or as expressions that are applied to values to derive end points. Operational databases and star schemas can benefit from a common set of extended date attributes that eliminate complex date expressions in SQL. Models and reports that use these tables and columns can present to the database simple predicates instead of complex expressions.

The Cognos BI query service exposes a family of functions that provide common user expressions such as adding and subtracting days and years from dates. These expressions are remapped to the equivalent expressions in the RDBMS and increase the portability of common business temporal expressions. Although the SQL standard defines interval types such as `Year_to_Month` and `Day_to_Second`, these interval types might not be supported by a particular vendor's RDBMS. Expressions that use or result in interval types, especially in predicates, can cause query decomposition and an increase in compensatory local processing.

6.5.2 Expressions on table columns in predicates

A predicate is best applied to a table column, not to an expression. If the left side of a predicate applies expressions to a column, it can impede the use of indexes and produce a less accurate estimate of selectivity by the database. Figure 6-1 shows the application of a string scalar function to perform a case-blind string comparison.

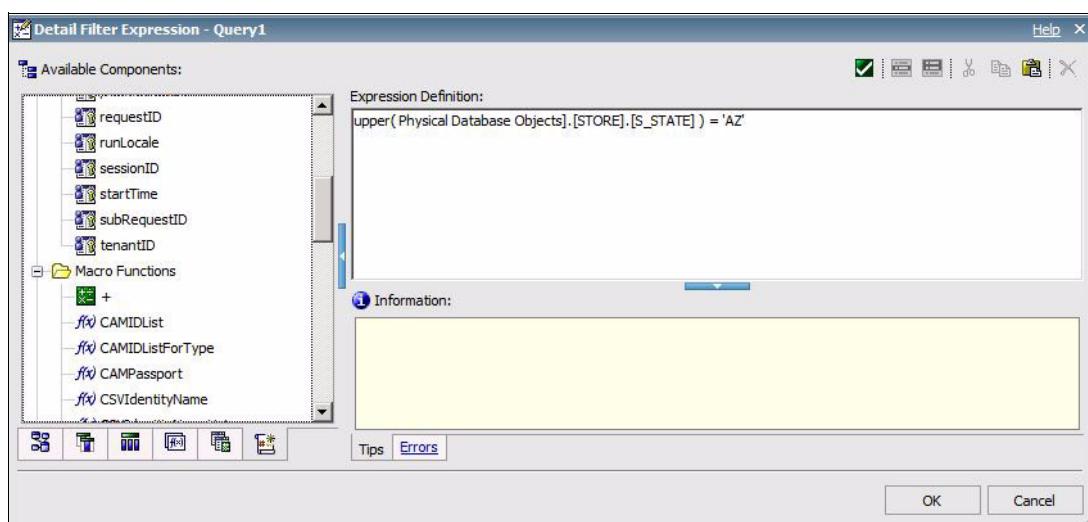


Figure 6-1 Applying expressions on columns in predicates

You can change the expression so that it uses only functions on the right side of the predicate. Alternatively, the database tables can be extended with columns to hold the computed value. Some RDBMS vendors provide the ability to define virtual table columns that are based on expressions that can be indexed.

6.6 Reduce explicit or implicit conversions

Ideally, an expression that serves as a key in a join relationship between tables resolves to the same data type as the corresponding key on the opposite side of the join relationship. This prevents constraining the RDBMS from considering certain join strategies, such as a hash join, because of incompatible data types. The database administrator can determine if the data types of the columns used in table joins are of the same type or of types that do not impede the RDBMS. The Framework Manager modeler must also determine if the join relationships between query subjects and stand-alone filters include expressions that might force implicit or explicit data type conversions.

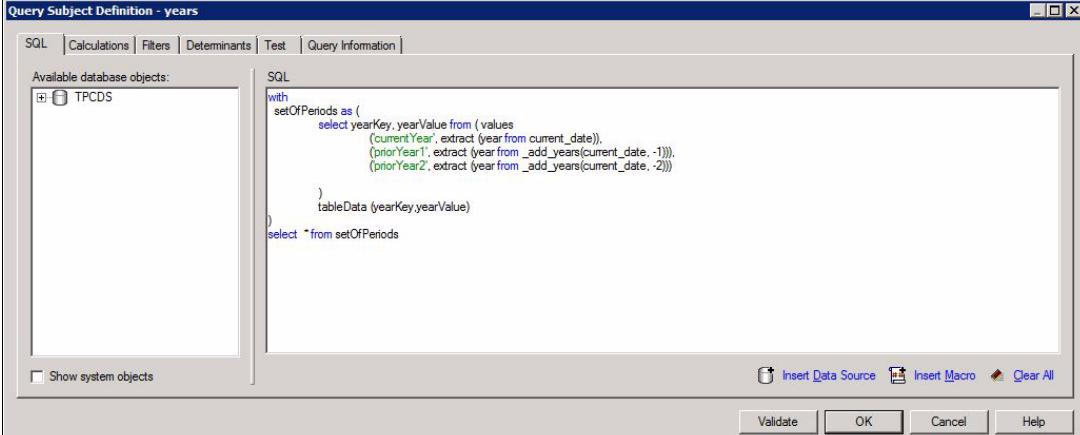
6.7 Minimize complexity of conditional query items

Reports are frequently designed with complex conditions used in predicates, groupings, and aggregations. Often, conditional expressions are employed so users can choose, at run time, how they want the information customized. These expressions can result in many large conditional expressions, which are more costly for the RDBMS to process than simple column references, literals, or other, more compact expressions.

Cognos BI features, such as Active Reports, can support many interactive user requirements in a manner that is not dependent on query-based approaches. If the queries cannot be avoided altogether, then use query items defined in a Framework Manager model or report, which can eliminate or reduce the complexity of SQL expressions through the use of prompts and parameter maps,

For example, consider reports that must present grouped data where several aggregates are dynamically determined based on conditional logic. The conditional logic is repeated within each aggregate and frequently appears in other predicates and expressions in the statement.

Figure 6-2 shows a simple Cognos SQL statement that generates a set of rows using row constructors. In turn, the row constructors populate the parameter map presented in Figure 6-3. Each row returned by the query generates a key that can be referenced by reports. The values associated with the key will be generated in the query at run time.



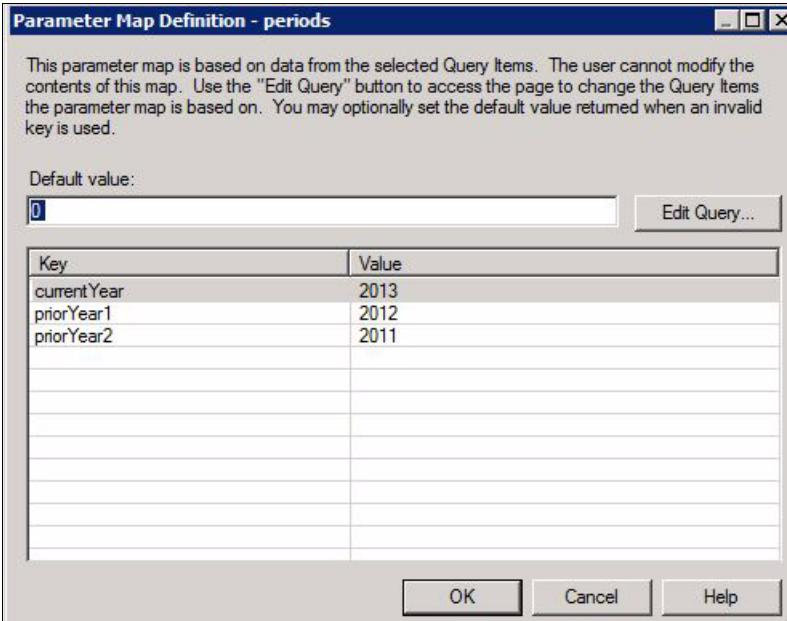
```

Query Subject Definition - years

Available database objects: TPCDS
SQL
with
setOfPeriods as (
    select yearKey, yearValue from ( values
        ('currentYear', extract (year from current_date)),
        ('priorYear1', extract (year from _add_years(current_date, -1))),
        ('priorYear2', extract (year from _add_years(current_date, -2)))
    )
    tableData (yearKey,yearValue)
)
select * from setOfPeriods

```

Figure 6-2 Cognos SQL generating a set of rows using row constructors



This parameter map is based on data from the selected Query Items. The user cannot modify the contents of this map. Use the "Edit Query" button to access the page to change the Query Items the parameter map is based on. You may optionally set the default value returned when an invalid key is used.

Key	Value
currentYear	2013
priorYear1	2012
priorYear2	2011

Figure 6-3 Parameter map definition that can be populated by a query

Figure 6-4 shows a query subject with expressions that reference the keys of the parameter map instead of applying an actual calculation on the current date and extracting the year. The resulting SQL, shown in Figure 6-5, contains case expressions with literal values that were retrieved from the parameter map.

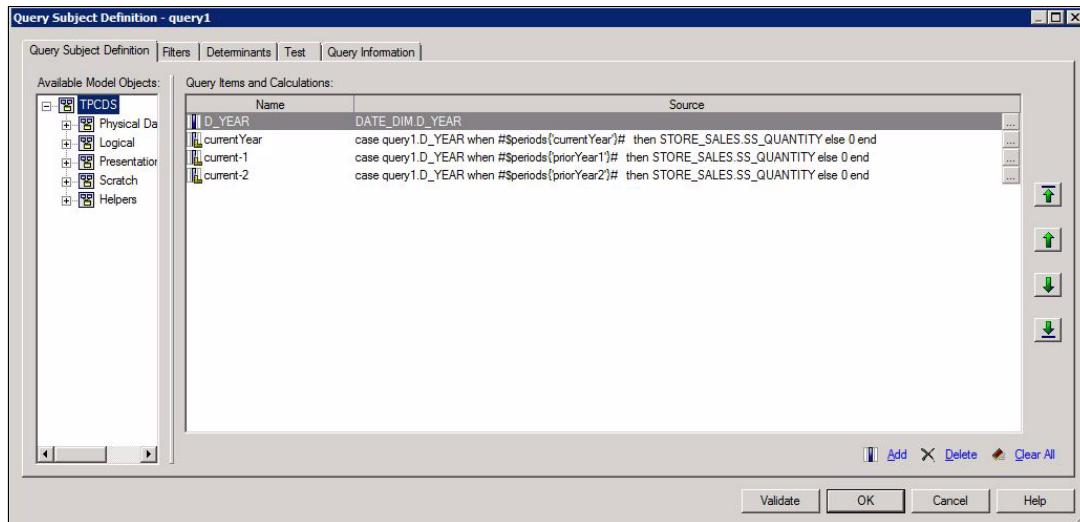


Figure 6-4 Query subject with conditional query times referencing parameter maps

```

Cognos SQL
SELECT
    DATE_DIM.d_year AS D_YEAR,
    CASE DATE_DIM.d_year
        WHEN 2013 THEN STORE_SALES.ss_quantity
        ELSE 0
    END AS currentYear,
    CASE DATE_DIM.d_year
        WHEN 2012 THEN STORE_SALES.ss_quantity
        ELSE 0
    END AS current_1,
    CASE DATE_DIM.d_year
        WHEN 2011 THEN STORE_SALES.ss_quantity
        ELSE 0
    END AS current_2
FROM
    TPCDS.tpcds.informix.date_dim DATE_DIM
    INNER JOIN TPCDS.tpcds.informix.store_sales STORE_SALES
        ON DATE_DIM.d_date_sk = STORE_SALES.ss_sold_date_sk
WHERE
    DATE_DIM.d_year BETWEEN 2011 AND 2013

```

Figure 6-5 SQL based on parameter map values

The set of keys and values in query-based parameter maps can be dynamically calculated using SQL concepts supported by a database. For example, Figure 6-6 shows a recursive common table expression that is used to calculate a parameter map representing a rolling, 12-month period based on the current date. The result set generated by this expression is presented in Figure 6-7.

The screenshot shows the 'Query Subject Definition - years' dialog box. The 'SQL' tab is selected. On the left, under 'Available database objects', there is a tree view with 'TPCDS' expanded. The main area contains the following SQL code:

```

with
setOfPeriods (PKey, C_Month, Ctr, Curr_Month) as
(
    select PKey, C_Month, Ctr, Curr_Month
    from ( values (
        cast('Curr_Month' as varchar(12)),
        extract(month from current_date),
        0,
        extract(month from current_date)
    ) seedQuery (PKey, C_Month, Ctr, Curr_Month)
    union all
    select
        'C_month' || cast(Ctr + 1 as char(2)),
        case
            when Curr_Month + Ctr + 1 > 12
            then Curr_Month + (-12 + Ctr + 1)
            else Curr_Month + Ctr + 1
        end,
        Ctr+1,
        Curr_Month
    from setOfPeriods
    where Ctr < 11
)
select PKey, C_Month, Ctr, Curr_Month from setOfPeriods

```

At the bottom of the dialog, there are buttons for 'Insert Data Source', 'Insert Macro', 'Clear All', 'Validate', 'OK', 'Cancel', and 'Help'.

Figure 6-6 Recursive common table expression that is used to calculate a parameter map

PKEY	C_MONTH	CTR
Curr_Month	7	0
C_month1	8	1
C_month2	9	2
C_month3	10	3
C_month4	11	4
C_month5	12	5
C_month6	1	6
C_month7	2	7
C_month8	3	8
C_month9	4	9
C_month10	5	10
C_month11	6	11

Figure 6-7 Resulting rolling period of rows from expression in Figure 6-6

Another approach to present grouped data with dynamically determined aggregation is to use a simple control table with logic to retrieve the requested results from either sets of rows, stored procedures, or other vendor-specific RDBMS mechanisms to generate the series of rows you want. As with the previous examples, the intent is to significantly reduce the number of complex expressions that need to be evaluated in an SQL statement.

Query items in a model or report can also use Cognos BI prompt syntax and macro functionality to reduce expression complexity. A prompt can be defined in terms of values that are displayed to (and selected by) a business user, such as a country name, and the value that is passed to a query based on the name the user selected. The displayed value is typically either a typed-in literal value (such as Market) or a value derived from a query used to populate the prompt. As a result, prompt values can be presented to users with friendly business names for sales territories, and the executed query uses more efficient integer key values that are associated with the display names.

Figure 6-8 shows a simple query item in a report that returns a different column based on the user's selection. The query item can be used several times in the query for filtering, grouping, and sorting the data, but this requires the expression to be repeated multiple times in the SQL statement, as shown in Figure 6-9.

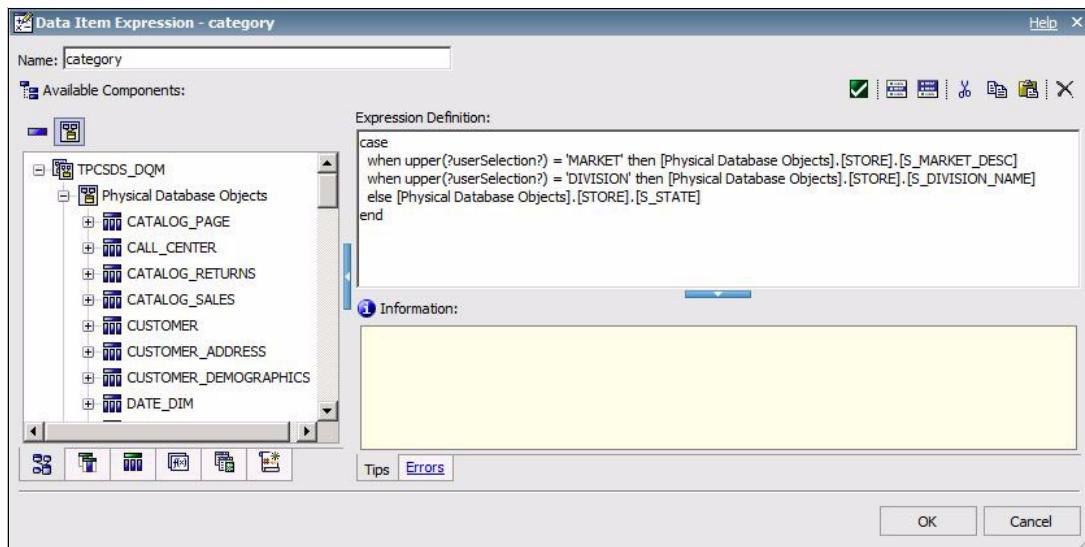


Figure 6-8 Query item that retrieves data from different columns, depending on user input

```

SELECT
CASE
    WHEN UPPER(:userSelection:) = 'MARKET' THEN STORE.s_market_desc
    WHEN UPPER(:userSelection:) = 'DIVISION' THEN STORE.s_division_name
    ELSE STORE.s_state
END AS category
FROM
    TPCDS.tpcds.informix.store STORE
GROUP BY
CASE
    WHEN UPPER(:userSelection:) = 'MARKET' THEN STORE.s_market_desc
    WHEN UPPER(:userSelection:) = 'DIVISION' THEN STORE.s_division_name
    ELSE STORE.s_state
END

```

Figure 6-9 SQL statement generated from the case expression of Figure 6-8

Another way you can avoid long-running SQL case expressions is by defining prompts that accept valid Cognos BI expressions (tokens). Figure 6-10 shows a prompt macro that is defined to accept a token data type. The token type is provided to the macro expression at run time based on the prompt value the user selects.

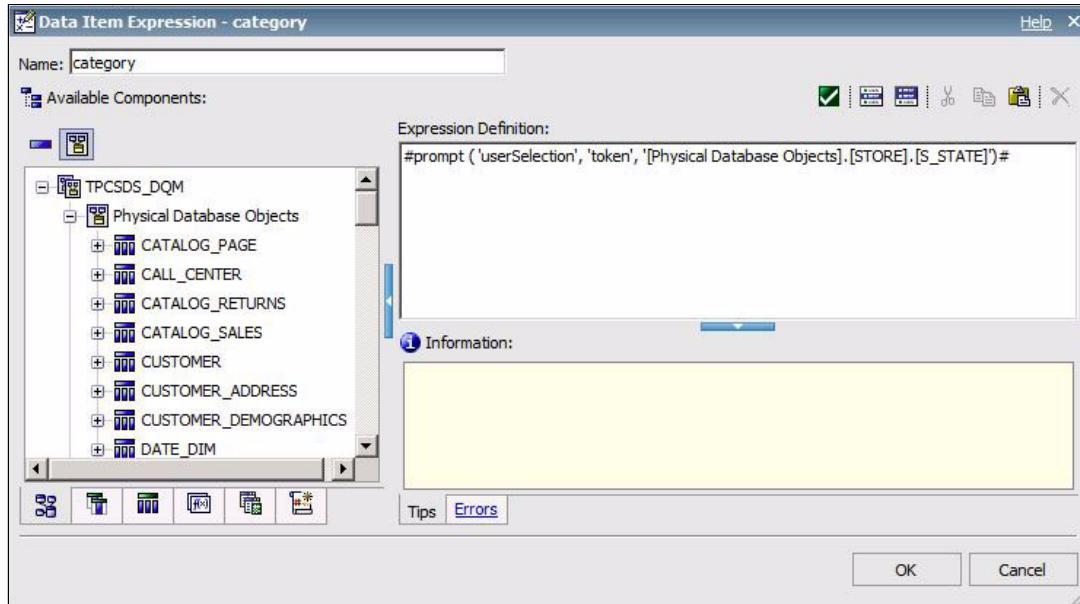


Figure 6-10 Prompt macro using token type

Figure 6-11 shows the Static Choices definition screen for a value prompt in Report Studio. The simple values defined in the Display column are presented to users; the corresponding expressions for each of the values are defined in the Use column.

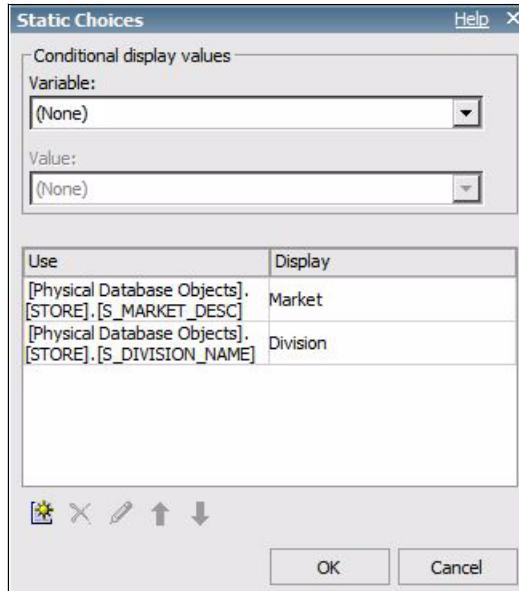


Figure 6-11 Defining Use and Display values for a prompt

Another form of substitution can be defined using the simple *case* macro expression, as the example in Figure 6-12 shows.

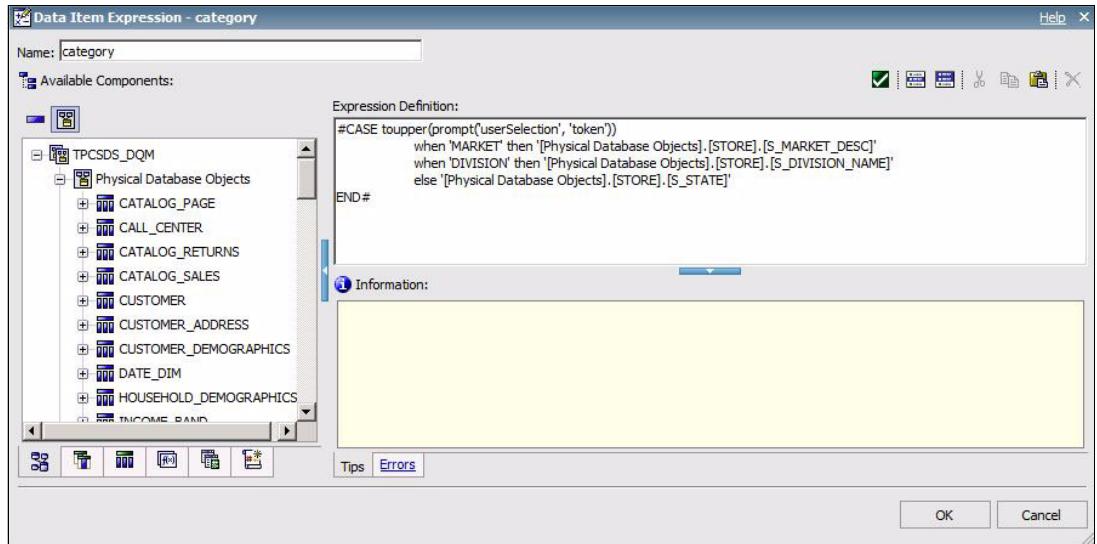


Figure 6-12 Case macro expression to ensure substitution occurs before SQL is submitted

When possible, the Cognos BI server attempts to apply dead code elimination techniques during query planning. In Figure 6-13, the value provided by a prompt is directly compared to a literal value that can be evaluated during planning.

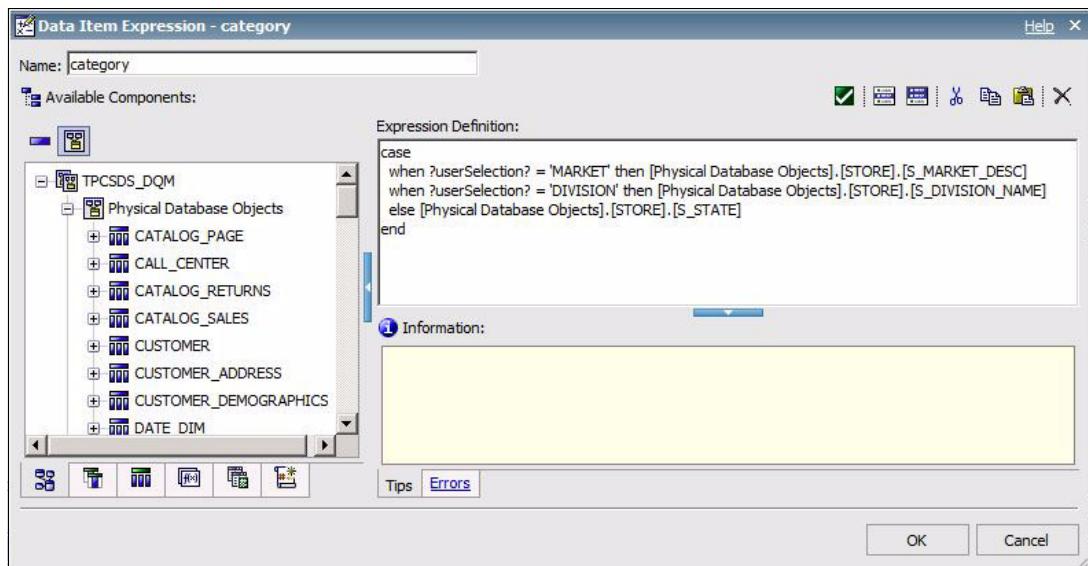


Figure 6-13 Expression that allows for dead code elimination

The result is a simple column reference in the generated SQL statement, as shown in Figure 6-14.

```
SELECT
  STORE.s_state AS category
FROM
  TPCDS.tpcds.informix.store STORE
GROUP BY
  STORE.s_state
```

Figure 6-14 SQL after code elimination techniques have been applied

This dead code elimination strategy can be used by authors to prune complex branches of logic from a query. For example, Figure 6-15 shows a complex body of logic in a filter that combines expressions and prompts. This logical expression is likely to expand into a more complex expression in the SQL statement that is sent to the RDBMS, as Figure 6-16. shows.

```
[x].[Transaction].[Gain Loss Adj Ind] is null)
or ((?TypeFilter? = '[TransDateTypePrompt]' = [ResolveDateLBDPrompt]
      and [x].[Transaction].[TransGLType] contains 'Y'
      and [x].[Transaction].[GL LBD Ind] contains 'Y')
  or (?TypeFilter? = '[TransDateTypePrompt]' in_range ?DateRangeParam?
      and [x].[Transaction].[TransGLType] contains 'Y'
      and [x].[Transaction].[Random Range Trade Date Same FromTo]contains 'Y')
  or (?TypeFilter? = '[TransDateMTDPrompt]' between [ResolveDateBeginMTDPrompt] and
      [ResolveDateLBDPrompt]
      and [x].[Transaction].[TransGLType] contains 'Y'
      and [x].[Transaction].[GL MTD Ind] contains 'Y')) )
```

Figure 6-15 Complex expressions with prompts

```
and ((F006_TRANSACTION.CGL_ADJ_PRINT_IND is NULL)
      or (((['TransDateTypePrompt'] = [ResolveDateLBDPrompt]' = '[TransDateTypePrompt]' =
      [ResolveDateLBDPrompt]')
            and (((substring(F006_TRANSACTION.CGL_ADJ_PRINT_IND from 2 for 1) ||
            substring(F006_TRANSACTION.CGL_ADJ_PRINT_IND from 5 for 1)) ||
            substring(F006_TRANSACTION.CGL_ADJ_PRINT_IND from 8 for 1)) ||
            substring(F006_TRANSACTION.CGL_ADJ_PRINT_IND from 11 for 1)) like '%Y%'))
            and (substring(F006_TRANSACTION.CGL_ADJ_PRINT_IND from 1 for 3) like
            '%Y%'))
      or ((['TransDateTypePrompt'] = [ResolveDateLBDPrompt]' = '[TransDateTypePrompt]' in_range ?DateRangeParam?)
            and (((substring(F006_TRANSACTION.CGL_ADJ_PRINT_IND from 2 for 1) ||
            substring(F006_TRANSACTION.CGL_ADJ_PRINT_IND from 5 for 1)) ||
            substring(F006_TRANSACTION.CGL_ADJ_PRINT_IND from 8 for 1)) ||
            substring(F006_TRANSACTION.CGL_ADJ_PRINT_IND from 11 for 1)) like '%Y%'))
            and (substring(F006_TRANSACTION.CGL_ADJ_PRINT_IND from 10 for 1) like
            '%Y%'))
      or ((['TransDateTypePrompt'] = [ResolveDateLBDPrompt]' = '[TransDateMTDPrompt]' between
      [ResolveDateBeginMTDPrompt] and [ResolveDateLBDPrompt])
            and (((substring(F006_TRANSACTION.CGL_ADJ_PRINT_IND from 2 for 1) ||
            substring(F006_TRANSACTION.CGL_ADJ_PRINT_IND from 5 for 1)) ||
            substring(F006_TRANSACTION.CGL_ADJ_PRINT_IND from 8 for 1)) ||
            substring(F006_TRANSACTION.CGL_ADJ_PRINT_IND from 11 for 1)) like '%Y%'))
            and (substring(F006_TRANSACTION.CGL_ADJ_PRINT_IND from 4 for 3) like '%Y%'))))
```

Figure 6-16 Physical SQL formed from complex expression

The original expression from Figure 6-15 on page 78 can be refactored, as shown in Figure 6-17, to take advantage of constant folding optimizations during query planning that will simplify the generated expression. In this scenario, the expressions are restructured to allow simple column and literal evaluations during planning, which result in more compact SQL at run time.

```

([x].[Transaction].[Gain Loss Adj Ind] is null)
or (1 = (case ?TypeFilter?
when '[TransDateTypePrompt]' = [ResolveDateLBDPrompt]'
then
    case
    when [x].[Transaction].[TransGLType] contains 'Y' and
        [x].[Transaction].[GL LBD Ind] contains 'Y'
    then 1
    end
when '[TransDateTypePrompt]' in_range ?DateRangeParam?
then
    case
    when [x].[Transaction].[TransGLType] contains 'Y' and
        [x].[Transaction].[Random Range Trade Date Same FromTo] contains 'Y'
    then 1
    end
when '[TransDateMTDPrompt]' between [ResolveDateBeginMTDPrompt] and
[ResolveDateLBDPrompt]
then
    case
    when [x].[Transaction].[TransGLType] contains 'Y' and
        [x].[Transaction].[GL MTD Ind] contains 'Y'
    then 1
    end
end)
)

```

Figure 6-17 Refactored expression to exploit code elimination

Figure 6-18 shows the simplified SQL that is generated.

```

((F006_TRANSACTION.CGL_ADJ_PRINT_IND is NULL)
 or (1 = case when (((substring(F006_TRANSACTION.CGL_ADJ_PRINT_IND from 2 for 1) ||
substring(F006_TRANSACTION.CGL_ADJ_PRINT_IND from 5 for 1)) ||
substring(F006_TRANSACTION.CGL_ADJ_PRINT_IND from 8 for 1)) ||
substring(F006_TRANSACTION.CGL_ADJ_PRINT_IND from 11 for 1)) like '%Y%' )
and (substring(F006_TRANSACTION.CGL_ADJ_PRINT_IND from 1 for 3) like '%Y%')) then 1
end))

```

Figure 6-18 Simpler generated SQL from re-factored expression

6.8 Review the order of conjunctions and disjunctions

Complex expressions can include terms that are combined with a conjunction (AND), or a disjunction (OR). Although many expression engines attempt to terminate solving expressions as early as possible, a concept called *early out*, the order of the operations can be optimized by reordering the terms. When rearranging a disjunction, place the most likely conditions first. For conjunctions, place the least likely conditions first.

For example, Figure 6-19 shows an expression that filters rows based on a state and city name. The second OR condition will evaluate whether the state is Texas (TX) before performing potentially long-character comparisons where many of the leading characters of city names can be similar. Subject to the distribution of the data, the comparisons of the city names might need to be reordered to maximize performance.

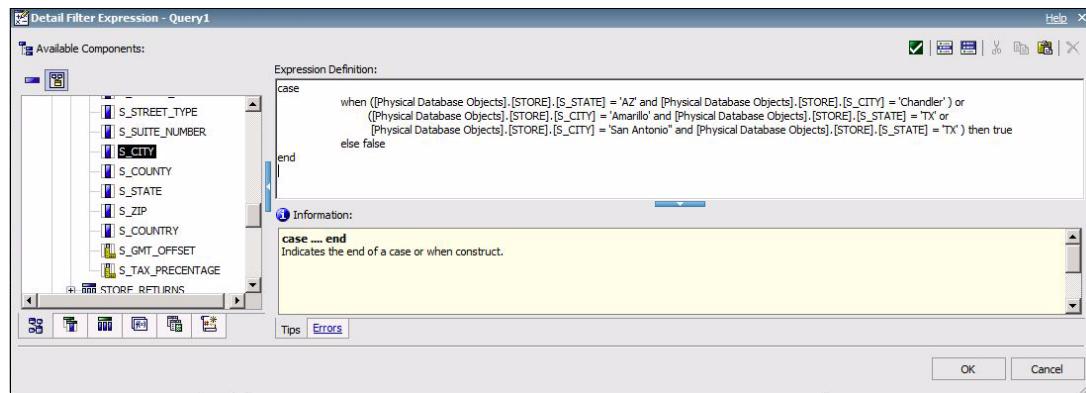


Figure 6-19 Case statement with conjunctions and disjunctions

Queries that allow users to input large in-lists to filter rows should review whether the values can be expressed using shorter string values or other data types, such as integers, to enable faster evaluation of the values. Very large in-lists might also indicate a report design issue that is allowing or causing users to select a large set of values from a data-driven prompt. With some RDBMS, large in-lists can result in a statement that fails to execute on the database.

6.9 Avoid performance pitfalls in sub-queries

Within a query subject, you can define filters that determine if one or more column values exist in a set of rows returned by one or more sub-queries. For example, a query subject modeled with two detailed filters, as shown in Figure 6-20, results in multiple sub-queries in the SQL statement, as shown in Figure 6-21.

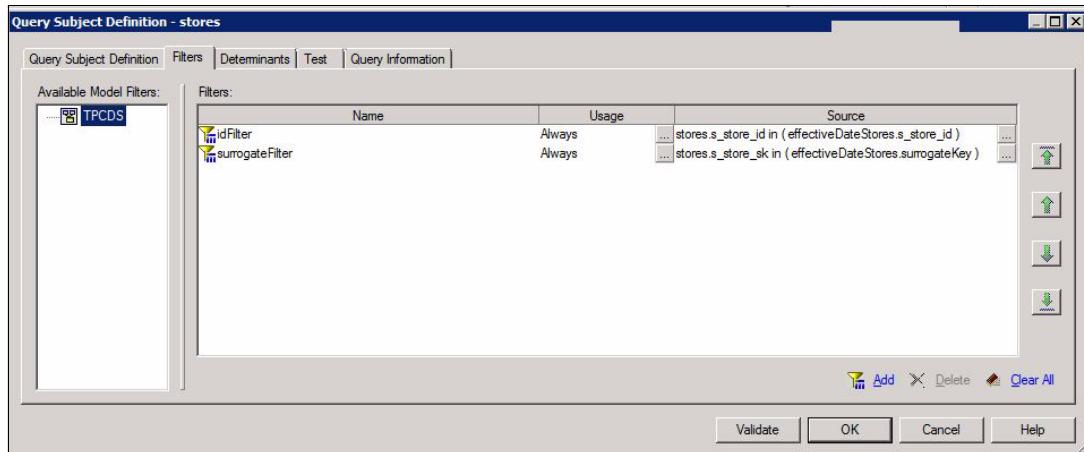


Figure 6-20 Multiple detail filters using sub-queries

The screenshot shows the 'Query Subject Definition - stores' dialog with the 'Query' tab selected. The SQL code is displayed in the 'Cognos SQL' section:

```
SELECT
    store.s_store_sk AS s_store_sk,
    store.s_store_id AS s_store_id,
    store.s_city AS s_city,
    store.s_county AS s_county,
    store.s_state AS s_state,
    store.s_zip AS s_zip
FROM
    TPCDS.tpcds.informix.store store
WHERE
    store.s_store_id IN (
        SELECT DISTINCT
            store.s_store_id AS s_store_id
        FROM
            TPCDS.tpcds.informix.store store
        WHERE
            store.s_rec_start_date >= :startDate: AND
            (store.s_rec_end_date <= :endDate: OR
            store.s_rec_end_date IS NULL) ) AND
    store.s_store_sk IN (
        SELECT DISTINCT
            MAX(store.s_store_sk) AS surrogateKey
        FROM
            TPCDS.tpcds.informix.store store
        WHERE
            store.s_rec_start_date >= :startDate: AND
            (store.s_rec_end_date <= :endDate: OR
            store.s_rec_end_date IS NULL) )
```

The dialog includes tabs for 'Query' and 'Response', and buttons at the bottom for 'Print', 'Test Sample', 'Total Rows', 'Options', 'Validate', 'OK', 'Cancel', and 'Help'.

Figure 6-21 SQL statement with multiple sub-queries in a filter

Many types of RDBMS attempt to apply transformations on statements that use sub-queries to optimize performance. Your database administrator can review the execution plans for time-consuming statements involving sub-queries and consider the following information:

- ▶ If multiple predicates are used in a filter, consider whether reordering the predicates will improve execution times.
- ▶ If multiple predicates are used in a filter and each predicate references another query, consider whether modeling a query subject with equivalent join relationships will improve execution times.
- ▶ If a filter uses an equality predicate (=), consider using = ANY() or using IN() instead.

Figure 6-22, Figure 6-23, and Figure 6-24 on page 83 illustrate how defining a relationship can help prevent costly sub-queries.

Figure 6-22 shows a model query subject that references items from another query subject that computes a set of keys corresponding to a given time period. The relationship between the two query subjects is defined with a predicate using two columns. Only one row per key will be returned in this scenario.

Query Items and Calculations:	
Name	Source
surrogateKey	effectiveDateStores.surrogateKey
s_store_id	effectiveDateStores.s_store_id
s_city	store.s_city
s_county	store.s_county
s_state	store.s_state
s_zip	store.s_zip

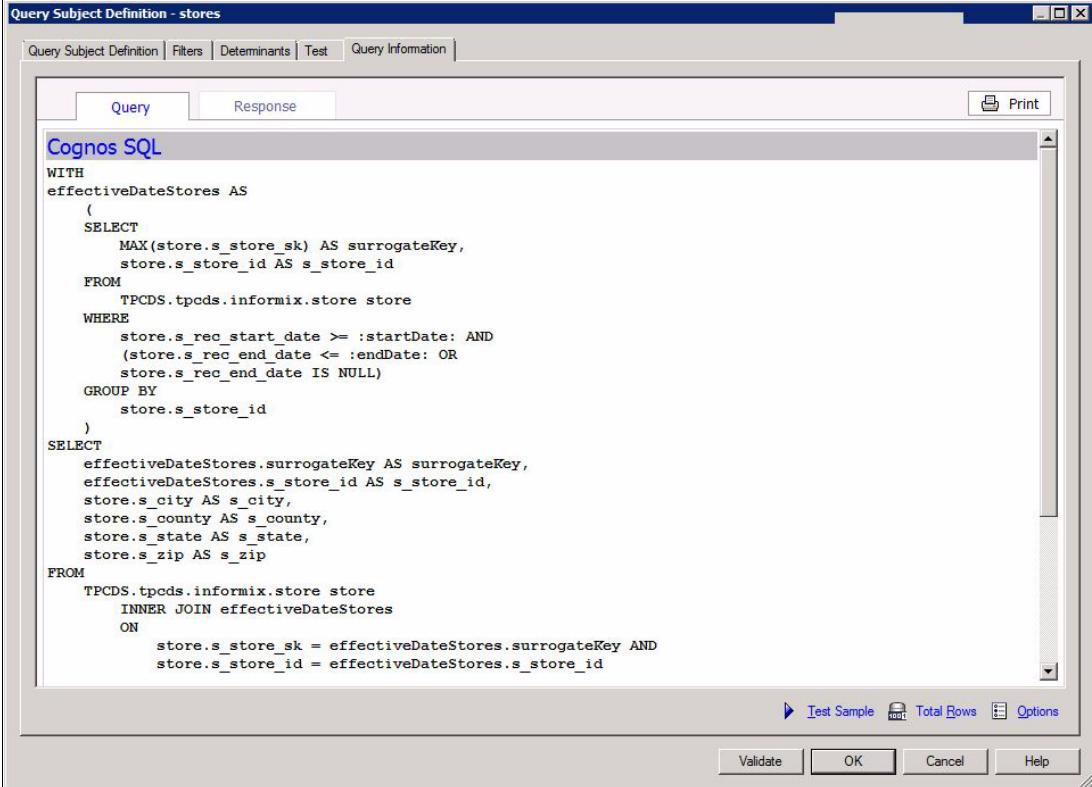
Figure 6-22 Model query subject joined to another query subject

Figure 6-23 shows the other query subject that computes the desired set of stores based on date criteria provided by the user. The query subject defines a determinant that groups the data by store and computes the highest applicable key using an aggregate.

Query Items and Calculations:	
Name	Source
s_store_id	store.s_store_id
surrogateKey	maximum (store.s_store_sk)

Figure 6-23 Query subject that groups data and computes the key using an aggregate

The SQL that is generated is shown in Figure 6-24. Sub-queries are avoided because the query subjects are referenced through a join relationship instead of a detail filter using a predicate. In more complex statements that reference the effective-date query subject multiple times, the SQL statement might include a named query within a common table expression if the RDBMS supports that construct. Otherwise, a new sub-query is generated several times as a derived table.



The screenshot shows the 'Query Subject Definition - stores' dialog box. The 'Query' tab is selected. The SQL code generated is:

```

Cognos SQL
WITH
effectiveDateStores AS
(
  SELECT
    MAX(store.s_store_sk) AS surrogateKey,
    store.s_store_id AS s_store_id
  FROM
    TPCDS.tpcds.informix.store store
  WHERE
    store.s_rec_start_date >= :startDate: AND
    (store.s_rec_end_date <= :endDate: OR
     store.s_rec_end_date IS NULL)
  GROUP BY
    store.s_store_id
)
SELECT
  effectiveDateStores.surrogateKey AS surrogateKey,
  effectiveDateStores.s_store_id AS s_store_id,
  store.s_city AS s_city,
  store.s_county AS s_county,
  store.s_state AS s_state,
  store.s_zip AS s_zip
FROM
  TPCDS.tpcds.informix.store store
  INNER JOIN effectiveDateStores
  ON
    store.s_store_sk = effectiveDateStores.surrogateKey AND
    store.s_store_id = effectiveDateStores.s_store_id

```

Figure 6-24 Generated SQL using query subjects with join relationships

Note: A join to a query must not change the cardinality of the required results. Be sure to verify whether the query will return distinct rows by default. A sub-query can be changed to a query subject that generates a derived table that contains a distinct or group by operation that removes duplicates.

6.10 Avoid unnecessary outer joins

Outer joins enable applications to return result sets when one or more tables in a statement lack associated data. Queries that use outer joins restrict various join optimization strategies and join ordering that the RDBMS sometimes uses with inner joins. A model might be constructed to always use outer joins that may not, in fact, be required by the business questions posed by a report. In these cases, the model can be extended with additional query subjects that describe inner join relationships.

Report authors can also consider using master-detail relationships that will retain data from the master query even if there are no details. This is similar to the intent of a left outer join.

In a star schema design, fact rows should be associated to members in the corresponding dimensions. In some cases, the actual business key for a fact might not be known as the fact data is delivered. It is preferred that the dimensions include a designate member that represents an unknown member. This enables the join relationships to avoid using outer joins and thus simplify the reporting experience for business users who want to select and group measures based on the unknown category.

6.11 Avoid using SQL expression to transpose values

Application authors who are familiar with SQL can construct expressions that attempt to massage database values for display. In several cases, such expressions can be replaced by using the available data type formatting, layout, and report expression features in the Framework Manager model and Cognos BI report authoring interfaces. Using the available formatting and layout facilities can reduce overhead in the RDBMS and provide locale-aware rendering.

Example 6-1 demonstrates how you can initiate multiple data type conversions, substrings, and concatenations to display a date value in a particular way rather than using the Data Format rendering option available in various authoring interfaces.

Example 6-1 Date formatting through data processing instead of rendering processing

```
Substring(Cast ( dateField, char(10)),6,2) || '-' || Substring(Cast ( dateField, char(10)),9,2) || Substring(Cast ( dateField, char(10)),1,4)
```

Application authors can define data-driven prompts that allow users to input values in a form that must be converted before the values can be used in a predicate. Ideally, the input values are transformed within the prompt definitions. Cognos BI provides a set of macro expressions that enable various forms of string expressions to be parsed and converted into the appropriate type of literal in the SQL statement.

Figure 6-25 shows a macro function that uses a mask specification to extract the year and month from the current date in a predicate where an application is storing an integer column that is intended to allow all days in a month to be selected. This solution does not require a between predicate within a _first_of_month and _last_of_month expression, which can take a relatively long time to process.

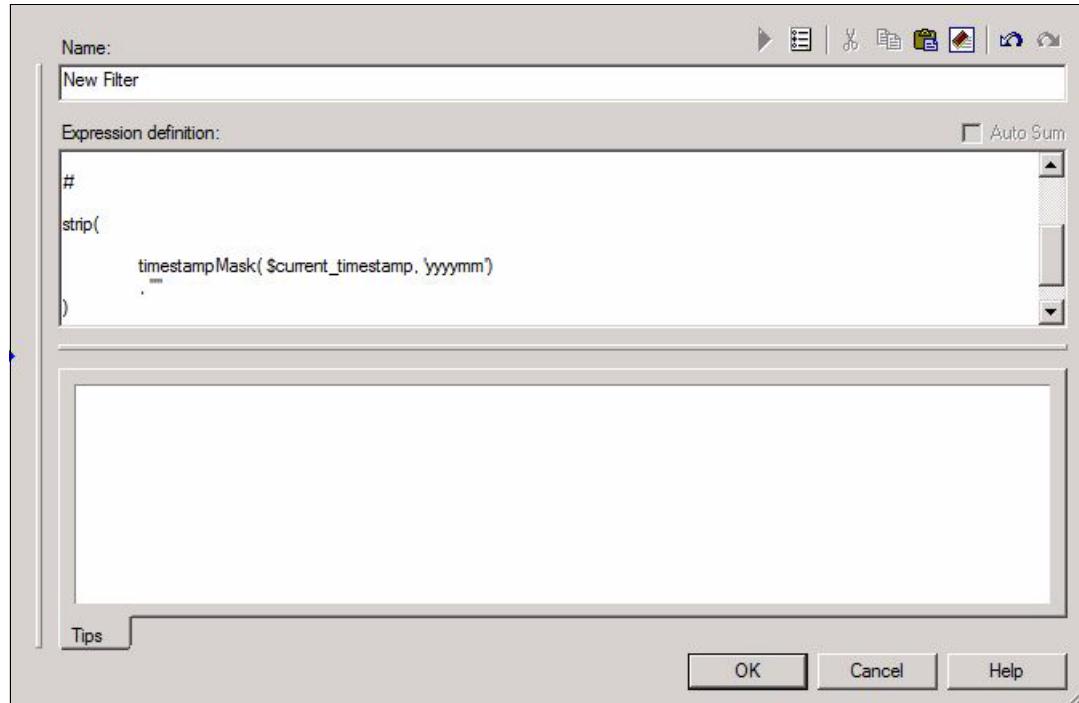


Figure 6-25 Macro function using a mask specification to extract year and month from current date

Note: Although relatively uncommon, the timestampMask() macro can be used for data sources that do not provide equivalent scalar functions. This is important for many business reports that frequently filter or group data by a temporal context, because it can simplify the SQL statements that are submitted to the underlying database.

Figure 6-26 shows how an inputted string value is transformed from dd/mm/yyyy format into yyyy-mm-dd format to accommodate the method in which a particular application stores character representations of date values.

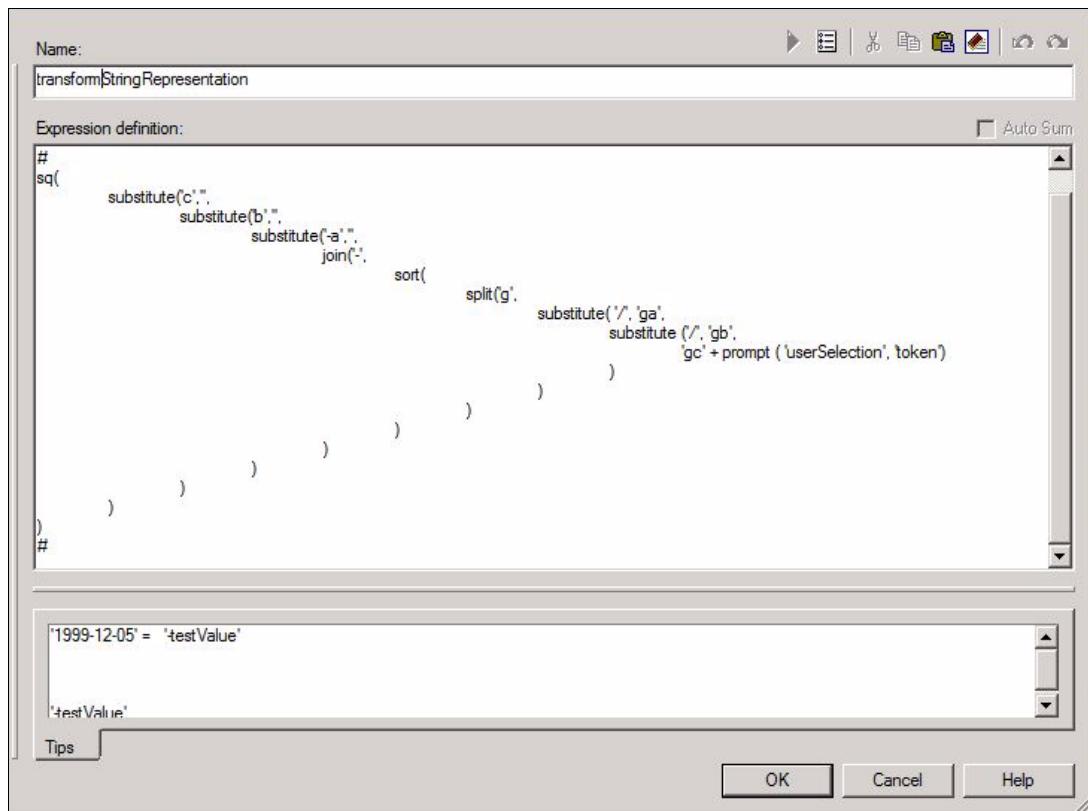


Figure 6-26 Macro expression to transform character representations of date values

6.12 Apply predicates before groupings

You can take steps to improve performance even when the RDBMS is unable to apply predicates prior to a grouping operation, or when the RDBMS ignores candidate materialized views. The predicate is likely to be applied to a value expression that is computing the minimum value of an attribute in association with a determinant in the Framework Manager model.

A Framework Manager model governor, Grouping of measure attributes, can be changed to include the attributes in the grouping list. Figure 6-27 on page 87 shows a query that projects three columns from a model query subject. The country and state columns are defined in as a group by determinant with state as an attribute that is determined by the two columns. Based on the setting of the model governor, the generated SQL statement can include the attribute as a column in the group by clause or with an aggregate. If a report attempts to filter data using that attribute, the RDBMS might not push the predicate ahead of the grouping operation, or it might be unable to match a materialized view.

```

Details:
WITH
store0 AS
(
  SELECT
    store.s_country AS s_country,
    store.s_state AS s_state,
    MIN(store.s_gmt_offset) AS s_gmt_offset
  FROM
    TPCDS.tpcds.informix.store store
  GROUP BY
    store.s_country,
    store.s_state
)
SELECT
  store0.s_country AS s_country,
  store0.s_state AS s_state,
  store0.s_gmt_offset AS s_gmt_offset
FROM
  store0
WHERE
  store0.s_gmt_offset < -5
GROUP BY
  store0.s_country,
  store0.s_state,
  store0.s_gmt_offset
ORDER BY
  s_country ASC,
  s_state ASC

```

Figure 6-27 Attribute predicate applied to an aggregated column

6.13 Trace SQL statements back to reports

The SQL statements generated by the Cognos BI query service can optionally contain embedded comments that can include application context. This enables administrators to gather workloads and see the reports and packages to which the SQL statements correspond.

The ability to see comments in dynamic SQL at the database server level depends on whether the database vendor supports the concept, and requires that the client driver not remove the comments during parsing.

Many of the macro functions described in Chapter 4, “Macros” on page 43 can be used to customize the information about the request’s context. You can use macros and session parameters to tie queries to the particular user that ran the report.

Previously authored reports have user-defined names; ad hoc analysis and query gestures are assigned system-generated names. Both user-specified names and system-generated names can help administrators to monitor workloads.

Figure 6-28 on page 88 displays SQL that the Cognos BI query service submitted to a database. The first line of the SQL is a comment string with the following items:

- ▶ The name of the authenticated user, or anonymous when no authentication was used
- ▶ The location and name of the report that was executed
- ▶ The name of the business query in the executed report
- ▶ An internal request identifier that can be used to link to the audit database data

```

/* user=Anonymous reportPath= queryName=channelPerformance requestID=sd9vj2829Cqh9qv9hsy8984yMl2jdlG9j24lMs4 */
SELECT
CASE
    WHEN NOT ("FS1"."DATE0" IS NULL) THEN "FS1"."DATE0"
    WHEN NOT ("FS2"."DATE0" IS NULL) THEN "FS2"."DATE0"
    WHEN NOT ("FS4"."DATE0" IS NULL) THEN "FS4"."DATE0"
    WHEN NOT ("FS3"."DATE0" IS NULL) THEN "FS3"."DATE0"
    WHEN NOT ("FS6"."DATE0" IS NULL) THEN "FS6"."DATE0"
    ELSE "FS5"."DATE0"
END AS "DATE0",
"FS4"."SS_QUANTITY" AS "SS_QUANTITY",
"FS3"."SR_RETURN_QUANTITY" AS "SR_RETURN_QUANTITY",
"FS2"."CS_QUANTITY" AS "CS_QUANTITY",
"FS1"."CR_RETURN_QUANTITY" AS "CR_RETURN_QUANTITY",
"FS6"."WS_QUANTITY" AS "WS_QUANTITY",
"FS5"."WR_RETURN_QUANTITY" AS "WR_RETURN_QUANTITY"
FROM
(
SELECT
    "FS6_inner"."DATE0" AS "DATE0",
    "FS6_inner"."WS_QUANTITY" AS "WS_QUANTITY",
    SUM(1)
    OVER(

```

Figure 6-28 Cognos generated SQL with comments appended for auditing purposes

You can enable or disable query service comment logging in SQL statements by using the Generate comments in native SQL setting that is shown in Figure 6-29. The steps are as follows:

1. Launch the IBM Cognos Administration portal page.
2. On the Configuration tab, select **Dispatchers and Services**.
3. Select the **Query Service** and then select the **Set properties** action.
4. On the Settings page, select the **Logging** category and change the value for the **Generate comments in native SQL** name.

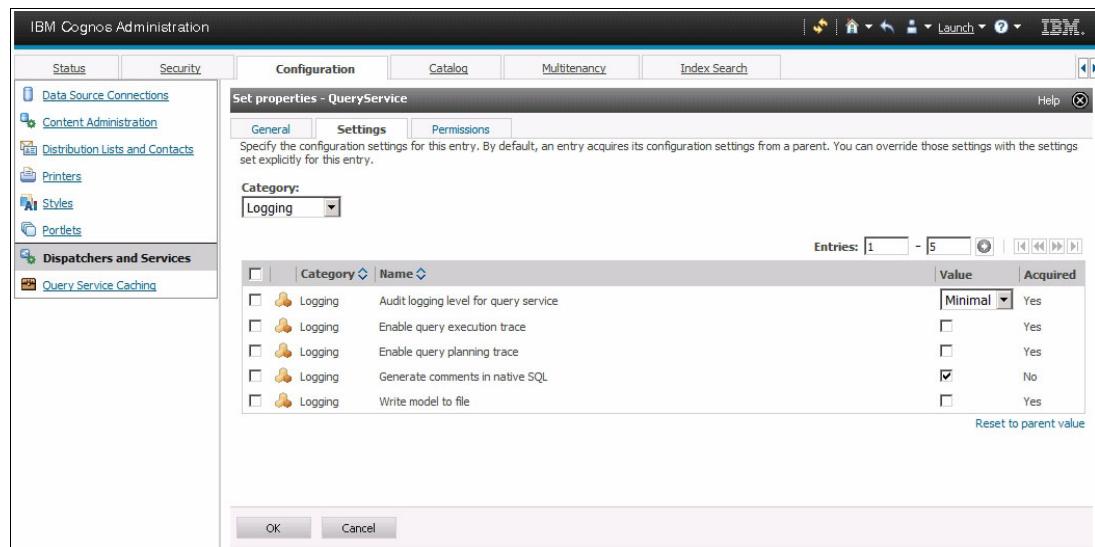


Figure 6-29 The Generate comments in native SQL setting of the Query Service



Troubleshooting

Troubleshooting is a systematic approach to solving a problem. The goal of troubleshooting is to determine why something does not work as expected and how to resolve the problem.

This chapter provides guidance on troubleshooting issues related to the dynamic query layer of IBM Cognos Business Intelligence (BI). Among the topics addressed are problem solving strategy, error messages, log files, the Cognos Dynamic Query Analyzer, and working with IBM Technical Support.

The chapter contains the following sections:

- ▶ 7.1, “Problem solving strategy” on page 90
- ▶ 7.2, “Error messages” on page 92
- ▶ 7.3, “Log files and tracing” on page 93
- ▶ 7.4, “Dynamic Query Analyzer” on page 95
- ▶ 7.5, “IBM technical support” on page 102

7.1 Problem solving strategy

Cognos BI is typically the user-facing front end of a complex ecosystem of enterprise software. The complexity of these systems is such that the symptoms of a problem can have many possible causes.

Sometimes, the solution to a problem is immediately apparent, such as when the details in an error message sufficiently describe what must be corrected.

For issues where the solution is not immediately apparent, you start by simplifying the scenario to reduce the number of possible causes. Any variable that is not directly related to the problem occurrence is a distraction from resolving the problem. A solution can become apparent most easily when you know the minimum factors under which the problem occurs.

A simple, two-step procedure can help you isolate the specific cause of a problem symptom:

1. Evaluate your most recent changes and find the point where the problem does not exist and the problem symptoms are not present (values are correct, performance meets expectations, no errors are returned). One approach to this process is explained in 7.1.1, “The half-split method” on page 90.
2. Implement your additional application requirements one-by-one, testing after every change. If you instead test after making multiple changes, you do not always know which change caused the problem behavior.

7.1.1 The half-split method

In the context of troubleshooting Cognos BI applications, the *half-split* method is a process of elimination that involves iteratively removing half or more of the variables until only the minimum factors under which the problem occurs are present. This approach is particularly useful if you are troubleshooting an application that you did not develop yourself.

Consider a report that returns an error when executed from IBM Cognos Report Studio. Under the half-split approach, you delete approximately half of the report objects (such as queries, charts, pages, or query items) and then execute the report again to test whether the error still occurs. If the error does not occur, then you restore the previously deleted objects and delete the others. If the error continues to occur, then delete half of the objects that were previously retained. If potential solutions are still not clear after you reduce the report to as few objects as possible under which the same error occurs, you might need to expand your investigation into the Framework Manager model.

The half-split method is often the most efficient way to identify the root cause of a problem. The best part of the half-split method is that little technical expertise is required to apply it. Even the most novice users can make considerable troubleshooting progress with this approach.

The half-split method can be particularly useful for troubleshooting professionally authored Report Studio reports that have multiple pages of content. These reports tend to be complex, which means narrowing the scope of the problem as your first step is even more important.

One technique for applying the half-split method in Report Studio is to use the View Tabular Data operation that is displayed when you right-click a query in the Query Explorer pane. As shown in Figure 7-1, you can use the View Tabular Data operation to test each query independently of all other queries in a report. This way can help you determine whether a particular query is causing the problem.

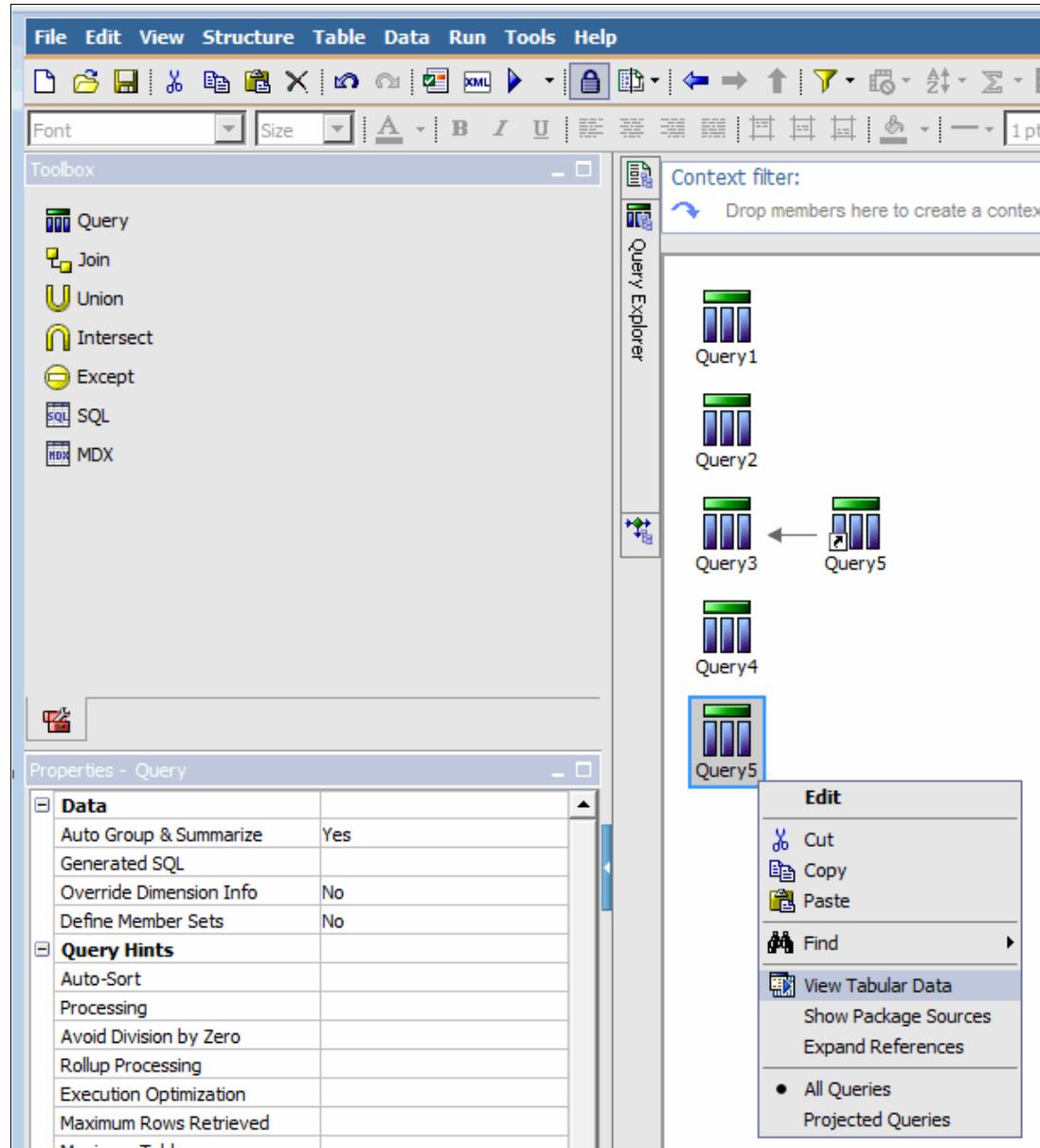


Figure 7-1 View Tabular Data operation in the Query Explorer pane of Report Studio

A similar technique uses the Run Page operation that is displayed when you right-click a page in the Page Explorer pane of Report Studio. As shown in Figure 7-2, with the Run Page operation, you can test each page independently of all other pages in a report. This can help you learn whether the contents of a particular page are causing the problem.

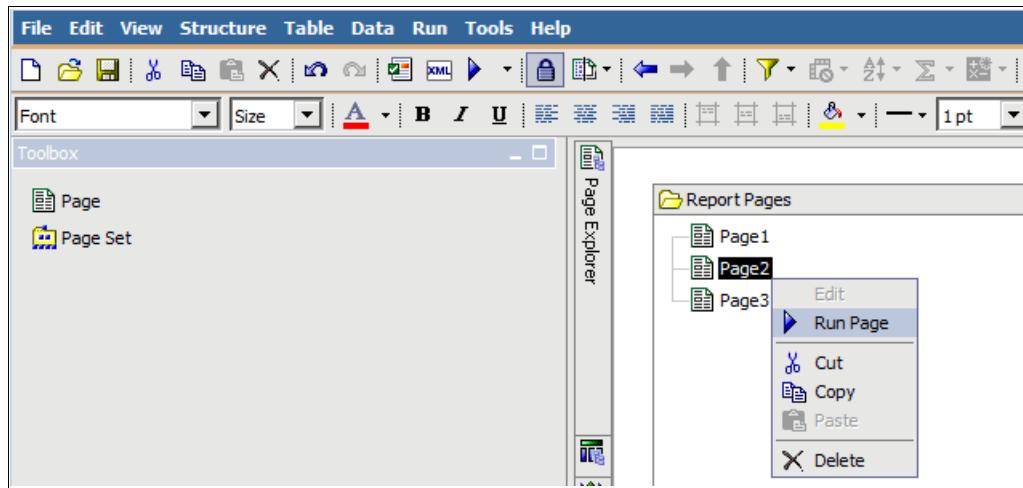


Figure 7-2 Run Page operation in the Page Explorer pane of Report Studio

7.2 Error messages

The first indication of a problem is often an error message. Error messages contain information that can be helpful in determining the cause of a problem. Example 7-1 shows the error message generated by the query service for a report with nonadjacent levels from the same hierarchy.

Example 7-1 Error message generated by the query service

XQE-PLN-0212 The report nests more than one level from the same hierarchy but they are not adjacent to each other. Please make levels from the same hierarchy adjacent.

Error messages returned by the Cognos BI server include an error code with characters that indicate the component and sub-component that encountered a software exception, and a number that identifies the exception. Error codes that are returned by the query service carry the prefix XQE. Error codes that are returned by the report service carry the prefix RSV. Other components within the Cognos BI server have their own error code prefixes. Include any relevant error codes in your search terms when you seek solutions online.

When you run a report from a browser and get an error, the short version of the error message is displayed for you. You can click the Details hyperlink to see the full error message, if one exists, and assuming you have the correct privileges to view error information. If the troubleshooting details you need are not shown, you can examine log files to learn details about the failure or exception that occurred.

Error messages submitted to the query service by the underlying data source may be presented to the user in the details of the XQE error message or recorded in log files on the Cognos BI server. Error messages from the underlying data source will provide the most detail about the root cause of the exception that occurred on the data source. Typically, you see a XQE-DAT-0001 Data source adapter message before any error message generated by the underlying data source software.

7.3 Log files and tracing

Log files can help you troubleshoot problems by recording the activities that take place when you work with a product. By default, operations performed by the query service are recorded in various log files in the \c10_location\logs\XQE directory for tracking purposes. An explanation of how to change the log file output directory for the query service is in the product information center:

<http://ibm.co/17z0yzb>

You can troubleshoot query-related issues in the dynamic query mode by using tracing capabilities that expand the information that is presented in log files. You can access settings for tracing in the properties of the QueryService service in IBM Cognos Administration, as described in the product information center:

<http://ibm.co/1a79Y75>

Important: Enabling query service tracing, particularly the query planning trace, results in the creation of large log files and can affect overall query performance. Avoid enabling query service tracing in production environments, and disable tracing as soon as you capture the information you need.

Because of the complexity of the information contained in query service tracing log files, the preferred approach is to apply the half-split method to simplify a report as much as possible before enabling tracing for the report. Analyzing the log files from a simple report is much easier than for a complex report. Dynamic Query Analyzer, which is described in 7.4, “Dynamic Query Analyzer” on page 95, can assist you in analyzing query service tracing log files.

If enable query service tracing is enabled, when a report is executed a new directory is created with the following naming convention

<yyyy-mm-dd>_<hhmmss>_<reportname>

The first part of the directory name is the date and time on the Cognos BI server where the report was executed. If the report that was executed has not been assigned a name, the name of the package is used instead. In the directory is the manifest.xml file, which contains contextual information related to the report's execution. The manifest.xml file contains the following information:

- ▶ Date: The Cognos BI server date and time when the request was received
- ▶ Report Name: The name of the report (if the report has been saved in the content store)
- ▶ Report Path: The location of the report in the content store
- ▶ Package Name: The name of the package associated with the report
- ▶ Model Path: The location of the package in the content store
- ▶ Request ID: The request identifier associated with the user-initiated transaction
- ▶ Operation Name: The name of the command for the transaction
- ▶ Sub-Request Id: An identifier for each sub-request, in cases where a request (user-initiated transaction) is divided into multiple sub-requests
- ▶ Expected Number of Passes: The number of passes used to plan the query

7.3.1 Query planning trace

The query planning trace writes information related to the transformation of the query to the plan tree log file. Use the query planning trace when you want to determine how the execution plan was determined by the query service.

The query planning component is a sophisticated system based on rules. It uses an inference engine that applies transformations in a sequence of passes. The initial query plan is transformed into a final execution plan over a series of transformations. The query planning trace logs contain the following information:

- ▶ The query state before the application of a transformation
- ▶ The reason a transformation was applied (or not applied)
- ▶ Each change that was applied to the query state as a transformation was applied
- ▶ The query state after the application of a transformation

When query planning tracing is enabled, log files are generated whenever each report is executed. The log files follow specific naming conventions:

- ▶ Tree log files:
`timestamp_reportName\planningLog.xml`
- ▶ Profiling log files:
`timestamp_reportName\planningLog_pass_log_number.xml`

For example, executing a report named Retailers results in the following planning log file:

`2012-01-10_11h33m700s_Retailers\planningLog.xml`

It also results in several pass logs with sequential file names such as these examples:

`2012-01-10_11h33m700s_Retailers\planningLog_pass_001.xml`
`2012-01-10_11h33m700s_Retailers\planningLog_pass_002.xml`.

Some reports require the execution of sub-queries. Sub-query trace files, including planning logs and pass logs, are stored under a separate directory within the main report directory. So, if the Retailers report requires the execution of one or more sub-queries, the trace files for those sub-queries are stored in the `2012-01-10_11h33m700s_retailers\subqueries` directory.

7.3.2 Query execution trace

The query execution trace writes information such as the native SQL or MDX statements to a run tree log file. Profile information is written to one or more separate logs. Profiling logs include execution and waiting-time metrics for query constructs.

The query execution trace logs display execution times in nanoseconds for every unit of processing over the various nodes of the query execution plan.

Log files are generated every time each report is executed. The log files follow specific naming conventions:

- ▶ Tree log files:
`timestamp_reportName\runtreeLog.xml`
- ▶ Profiling log files:
`timestamp_reportName\profilingLog-log_number.xml`

To extend the previous example, executing a report called Retailers results in the following log tree file:

2012-01-10_11h33m700s_Retailers\runtreeLog.xml

It also results in several profiling logs with sequential file names such as these examples:

2012-01-10_11h33m700s_Retailers\profilingLog-0.xml

2012-01-10_11h33m700s_Retailers\profilingLog-1.xml

Some reports require the execution of sub-queries. Trace files for sub-queries, including run tree logs and profiling logs, are stored under a separate directory within the main report directory. So, if the Retailers report requires the execution of one or more sub-queries, the trace files for the sub-queries are stored in the following directory:

2012-01-10_11h33m700s_retailers\subqueries.

The following XML element attributes are in the query execution trace log files:

- ▶ totalElapsedTime: The total elapsed time in the execution node, including time spent executing any descendant nodes
- ▶ ownElapsedTime: The total elapsed time in the execution node, excluding time spent executing any descendant nodes
- ▶ totalCPUTime: The total time directly attributed to the processing in the node and its descendants, excluding any other processing the operating system performed on unrelated activities
- ▶ ownCPUTime: The total time directly attributed to the processing of the node, not including its descendants and excluding any other processing the operating system performed on unrelated activities

7.4 Dynamic Query Analyzer

Some content in this section was previously published in IBM developerWorks¹.

Cognos Dynamic Query Analyzer is a tool that provides graphical representations for the query tracing logs produced by the query service. Dynamic Query Analyzer includes an Eclipse-based client user interface. Data is presented in a series of visual components called views. You can perform operations on the data contained in the active view. The results of these operations affect the content of other views, or cause new views to be opened automatically. The following three types of logs can be loaded within the Open Log dialog of Dynamic Query Analyzer:

- ▶ Profile: This is a log of the execution of a report with timing information.
- ▶ Runtree: This is a log of the execution of a report with no timing information. This log will only be shown if there is no profile.
- ▶ Plan: This log depicts the initial query and the final query just before an execution was attempted. The plan log is only loaded if execution fails.

Documentation is available regarding how to open log files and perform other actions with Dynamic Query Analyzer. See the Cognos BI Information Center:

http://pic.dhe.ibm.com/infocenter/cbi/v10r2m1/nav/5_8

¹ Source: *IBM Cognos Proven Practices: IBM Cognos 10 Dynamic Query Analyzer User Guide*
http://www.ibm.com/developerworks/data/library/cognos/infrastructure/cognos_specific/page578.html

7.4.1 Graph nodes

The graph that is displayed when a query log file is opened shows a series of linked nodes. Each node represents either an operation that occurred when the report was run or an attribute of an operation (such as the data that was being processed). Figure 7-3 defines the meaning of the various node representations.

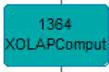
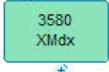
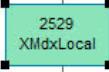
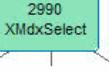
Node	Description
 1364 XOLAPComput	Represents an operation that occurred when the report was run.
 MdxText	Represents an attribute of an operation, such as the data being processed. By default, most of these nodes are suppressed in the graph. To display all of them, click Window, Preferences , and select Visualization, Node filtering .
 2529 XMdxLocal	Represents a collapsed node. To display the hidden nodes, double-click the node, or right-click it and click Show Subtree .
 3580 XMdx	Represents a node that contains hidden children due to the Node filtering settings. To display the hidden nodes, right-click the node and click Expand Filtered .
 2529 XMdxLocal	Represents the node that is currently selected. The properties of this node are displayed in the Properties view. You can select a node by clicking it.
 682 XMdxSelect	Represents a node that has subqueries that can be opened in another graph. To display the subquery graphs, right-click the node and click Open sub queries . Some subqueries do not have an associated node in the parent trace and can be opened using this option.
 2990 XMdxSelect	Represents timing information for the node. The color red represents the time for the report spent in the node. The color yellow represents the proportion of time spent in the children of the node. The color gray represents the time spent outside the node and its children. If the node is selected, the times are also displayed in the Properties view.

Figure 7-3 Node representation definitions

The nodes shown in a Dynamic Query Analyzer log graph are a hierarchical representation of the XML generated in the log. When the node has information that can be displayed, the node depiction in the graph is colored or filled with text. When a node has useful information that can be displayed, the node depiction in the graph is colored and contains text. Colored nodes represent elements used in the execution of the report. Non-colored nodes are superfluous data in the logs that are generally not useful for performance analysis. The colors for the node types can be viewed in the summary view and changed in the Colors and Fonts preference page. The colors beside the node show the timing information for the node.

Generally, the node properties presented in the graph are the node name and the ID of the operation the node executed. Square nodes, which are not shown in the graph by default, do not have an ID because they represent information that was not part of an execution of an operation. You can make square nodes visible in the graph using the Nodes Filtering page. All of the properties of the node can be seen in the Properties view. You can also click on the XML tag to see the contents of the log file as raw XML.

7.4.2 Views

Dynamic Query Analyzer offers several views to help you focus on the information that interests you. All views within Dynamic Query Analyzer can be opened using the Show View option in the Window menu. Some views are opened by default when a graph opens. This section describes the views that are available in Dynamic Query Analyzer are described in this section.

Navigation view

The Navigation view is a tree representation of a graph. Each graph has its own Navigation view, so the contents do not change when you select another graph. This allows you to compare graphs by opening multiple Navigation views.

You can enable the **Link to editor** setting on the toolbar so that the editor reflects what is selected in the graph, and vice versa. The Navigation view allows you to navigate to entries in the tree and to focus on them by double-clicking individual entries.

Figure 7-4 shows the Navigation view of a graph. The node that is highlighted in the graph was selected in the Navigation view because the Link to editor setting is enabled there.

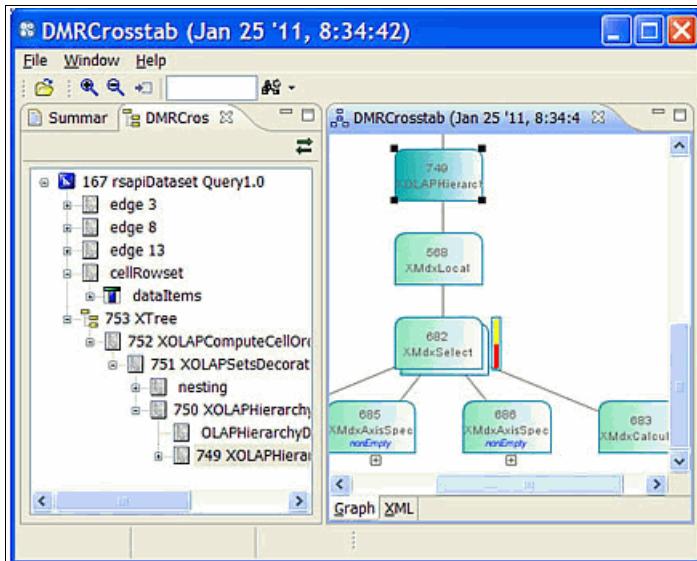


Figure 7-4 Navigation view displaying the graph structure as a tree

Summary view

The Summary view shows the overall information about a graph in a succinct format.

The Summary view has four sections:

- The Summary section shows the name, package, and time the report was run, with a graphic to quickly indicate whether the data source type is relational, OLAP, or dimensionally modeled relational (DMR).

Note: A system generated name will be assigned to any report that was run before being saved to the content store.

- The Timing section is where users doing profiling spend most of their time. The working and waiting time for each node is shown in descending order. Double-clicking on any of the entries takes you to the named node.
- The Analysis section shows additional information about the report that was run, including whether planning was successful, any query hints that were applied, and any filters that were applied. These factors can change the performance of a report significantly; be sure to check it if two seemingly identical reports have different timing characteristics.
- The Node shapes and colors section is a legend that explains the node types and colorings used in the Summary view.

Figure 7-5 shows the Summary view as it is opened by default, indicating the summary and timing information.

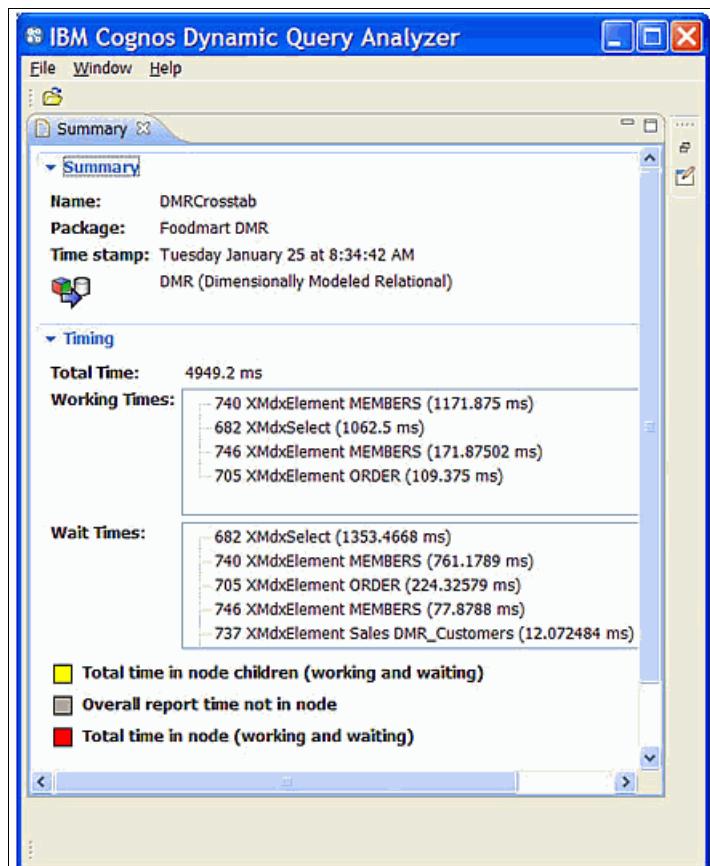


Figure 7-5 Summary view displaying execution time information

Query view

The Query view shows the MDX or SQL query that was executed to generate a report. For convenience, you can test the SQL from within Dynamic Query Analyzer by clicking the **Re-run the SQL** statement on the server option. The MDX query is much more tightly linked to the report execution and can be used to find the places in the graph that match the commands in the MDX. Selecting an MDX command in the Query view will highlight the corresponding node in the graph, assuming you enabled the **Link MDX to graph** setting. Figure 7-6 shows an MDX command selected in the Query view and the corresponding node highlighted after the **Link MDX to graph** setting is selected.

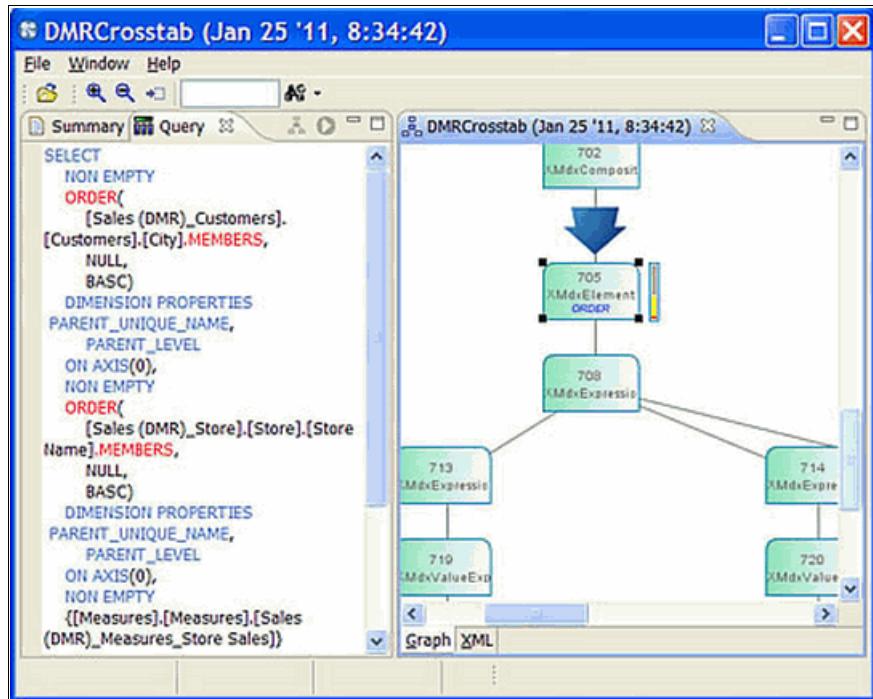


Figure 7-6 Query view with MDX command selected in graph

Report Logs view

The Report Logs view depicts all of the logs currently available on the Cognos BI server. The same logs appear when you click **File → Open log** on the Open Log dialog. The Report Logs view is a convenient way to locate the log that interests you.

Figure 7-7 shows a report execution log expanded to indicate its main profile and the sub-queries that occurred during execution of the report.

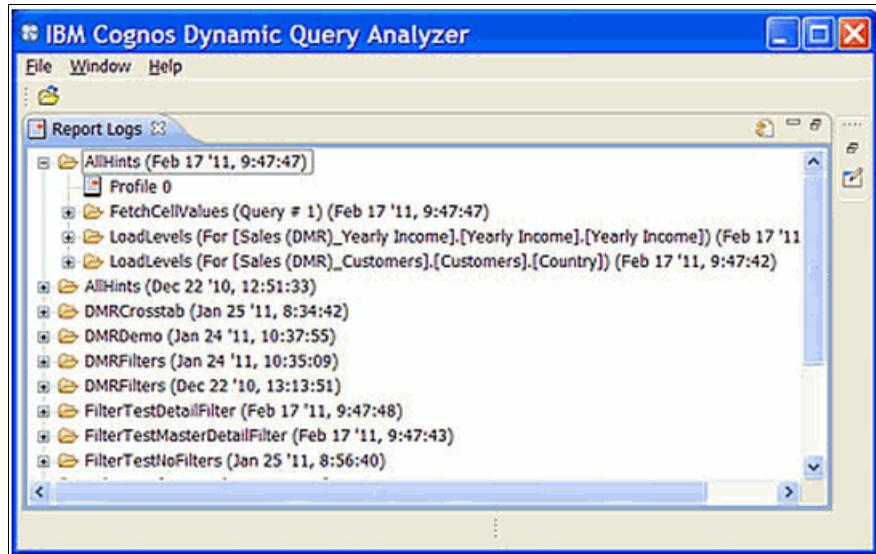


Figure 7-7 Report log view with command selected in graph

Content Store view

The Content Store view shows the reports that are available in your environment and enables you to run any report directly from Dynamic Query Analyzer. The list of folders and reports is the same as that in the Cognos Connection web portal. Unlike reports run from the portal, reports run from Dynamic Query Analyzer can generate logs on a report-by-report basis.

After a report runs, you can see the log for the run under the listing for the report in the content store view. If you do not see your log, look in the Report Logs view to see if it is there. Figure 7-8 shows the content store view with the log of a report run beneath the report entry.

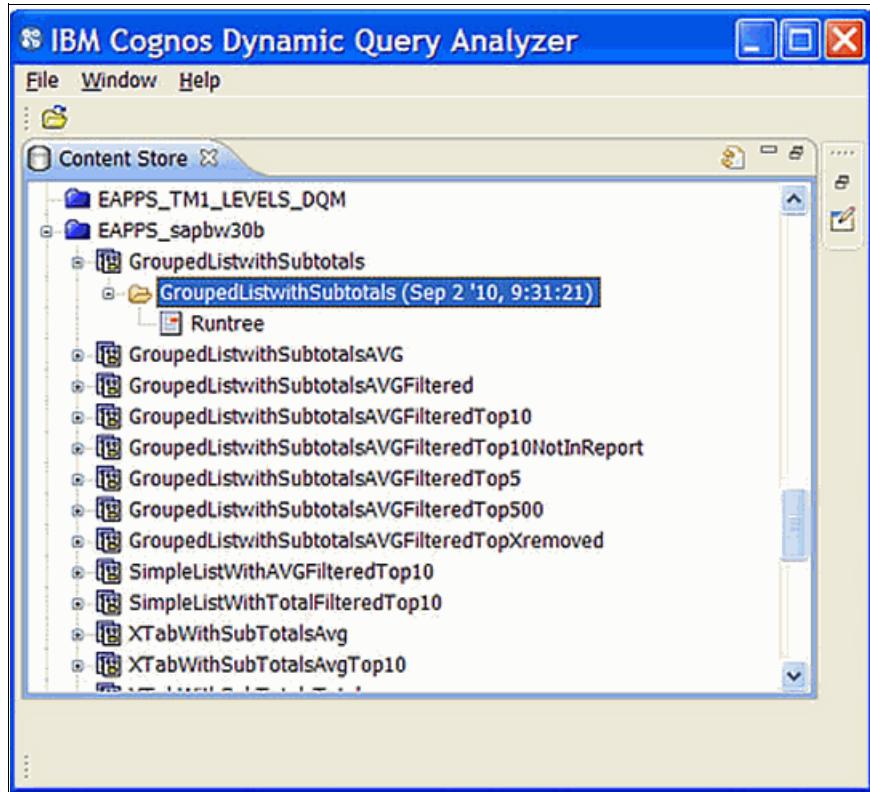


Figure 7-8 Content Store view with a report-generated log selected

DQM Server Logs view

The DQM Server Logs view shows the contents of the main server logs. These logs are for all events on the server, not only reports. These logs also provide details about operations that are not done by the query service. If you want to see if any entries are tied to an open report, select **File → Show in Server Log**.

The entries in this view are organized by server session. You can filter the entries appearing in the DQM Server Logs on any column that appears in this view. These filters can be cleared using the Clear Filters operation.

Figure 7-9 shows the latest server log expanded to indicate all entries that were written to that log. A new server log is initiated each time the server is restarted.

Component	Group	Level	Thread	Request Id
110708-0940300821				
XQEODP	junit.data.query	ERROR	1	
XQEBaseTestCase	Unspecified	ERROR	1	77885d91:131
XQE	Test	ERROR	1	
XQE	InitTerm	ERROR	1	
XQE	InitTerm	ERROR	1	
XQE	InitTerm	ERROR	1	
XQE	ProviderManager	ERROR	1	
XQE	Exception	ERROR	39	
XQE	InitTerm	ERROR	1	
XQE	InitTerm	ERROR	1	

Figure 7-9 Server logs view displaying the server entries for the latest session

7.5 IBM technical support

IBM prides itself on delivering world-class software support with effective online resources and a highly skilled, customer-focused staff. This section describes several important technical support offerings available to assist you in resolving problems.

7.5.1 IBM Support Portal

The IBM Support Portal is a unified, centralized view of all technical support tools and information for all IBM systems, software, and services. Find it at this address:

<http://www-947.ibm.com/support/entry/portal/overview>

The portal lets you access all the IBM support resources from one place. You can tailor the pages to focus on the information and resources you need for problem resolution and

prevention. Consider familiarizing yourself with the IBM Support Portal by viewing the demonstration videos available at the following web address:

<http://www.youtube.com/user/IBMElectronicSupport>

7.5.2 Service requests and PMRs

Service requests are also known as problem management reports (PMRs). These requests can be submitted by using the PMRs tool available at this address:

<https://www-947.ibm.com/support/servicerequest/Home.action>

Before contacting IBM Support, you should attempt to apply the troubleshooting strategy explained in 7.1, “Problem solving strategy” on page 90 so that you can effectively describe the scope of the problem. If you can effectively summarize the problem and symptoms before contacting software support, the problem solving process typically moves faster. Be sure you are as specific as possible when you explain a problem or question to software support specialists.

When you work with IBM technical support, you can accelerate the work by providing information that is detailed enough for the technical support analyst to be able to reproduce your problem. For issues that can be reproduced with the Cognos BI samples, you can submit the report specification or Framework Manager model that demonstrates the problem, along with an indication of sample database you were using. No data is required to reproduce issues related to Cognos BI SQL generation; submitting scripts to create the empty database tables should be sufficient. Issues related to data integrity often require you to submit your data, or a version of your data with the highly confidential elements removed or altered, to enable IBM technical support to reproduce your problem.

To exchange information with technical support, go to the following address:

<http://www-05.ibm.com/de/support/ecurep/index.html>

The service request escalation process for IBM Business Analytics products is at the following web location:

<http://www.ibm.com/software/analytics/cognos/customercenter/escalation.html>

7.5.3 IBM Fix Central

IBM Fix Central provides fixes and updates for your system’s software, hardware, and operating system. When you enter Fix Central, use the drop-down menu to navigate the fixes for your product.

You can access Fix Central at the following address:

<http://www-933.ibm.com/support/fixcentral/>

Fix lists and release schedules for IBM Fix Packs are at the following address:

<http://www.ibm.com/software/analytics/cognos/support/fixpacks.html>

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this book.

IBM Redbooks

The following IBM Redbooks publications provide additional information about the topic in this document. Note that some publications referenced in this list might be available in softcopy only.

- ▶ *IBM Cognos Dynamic Cubes*, SG24-8064
- ▶ *Big Data Analytics with IBM Cognos Dynamic Cubes*, TIPS0942
- ▶ *IBM Cognos Business Intelligence V10.1 Handbook*, SG24-7912
- ▶ *Gaining Insight with IBM Cognos Business Intelligence V10.1*, TIPS0947

You can search for, view, download or order these documents and other Redbooks, Redpapers, Web Docs, draft and additional materials, at the following website:

ibm.com/redbooks

Online resources

These websites are also relevant as further information sources:

- ▶ IBM Cognos Business Intelligence 10.2.1 Information Center:
<http://pic.dhe.ibm.com/infocenter/cbi/v10r2m1/index.jsp>
- ▶ Business analytics proven practices:
<http://www.ibm.com/developerworks/analytics/practices.html>

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services



Redbooks

IBM Cognos Dynamic Query

(0.2"spine)
0.17" <-> 0.473"
90<->249 pages



IBM Cognos Dynamic Query



Discover how Cognos accelerates query performance

Learn how to administer the Cognos query service effectively

Maximize the return on your analytic investments

This IBM Redbooks publication explains how IBM Cognos Business Intelligence (BI) administrators, authors, modelers, and power users can use the dynamic query layer effectively. It provides guidance for determining which technology within the dynamic query layer can best satisfy your business requirements. Administrators can learn how to tune the query service effectively and preferred practices for managing their business intelligence content.

This book includes information about metadata modeling of relational data sources with IBM Cognos Framework Manager. It includes considerations that can help you author high-performing applications that satisfy analytical requirements of users.

This book provides guidance for troubleshooting issues related to the dynamic query layer of Cognos BI.

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