



Product Information

This document applies to IBM Cognos Version 10.1.0 and may also apply to subsequent releases. To check for newer versions of this document, visit the IBM Cognos Information Centers (<http://publib.boulder.ibm.com/infocenter/cogic/v1r0m0/index.jsp>).

Copyright

Licensed Materials - Property of IBM

© Copyright IBM Corp. 2005, 2010.

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

IBM, the IBM logo, ibm.com, Impromptu, ReportNet, TM1, and Cognos are trademarks or registered trademarks of International Business Machines Corp., in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at www.ibm.com/legal/copytrade.shtml.

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

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

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

Table of Contents

Introduction	11
Chapter 1: What's New?	13
New Features in 10.1.0	13
Changed Features in 10.1.0	15
Deprecated Features in 10.1.0	15
Removed Features in 10.1.0	15
New Features in 8.4	15
Changed Features in 8.4	16
Deprecated Features in Version 8.4	17
Conformed Dimensions on SAP BW Data Sources	17
New Features in Version 8.3	17
Changed Features in Version 8.3	18
Deprecated Features in Version 8.3	18
Framework Manager Security Filters for SAP BW: Notice of Intent to Change the Default Setting	19
IQD Externalize Method	19
Removed Features in Version 8.3	19
Chapter 2: Getting Started with Framework Manager	21
Analyze the Problem	21
Building IBM Cognos Business Intelligence Applications	22
Objects You Will Use	23
Create a Project	26
Open a Project	27
The Project Page	28
The Project Viewer	28
Change Options for Projects	30
Reorder Objects	31
The Explorer Tab	32
The Diagram Tab	32
The Dimension Map Tab	34
The Properties Pane	34
The Tools Pane	36
Naming Conventions for Objects in a Project	38
Sample Models	39
The Great Outdoors Warehouse Model	39
The Great Outdoors Sales Model	40
Chapter 3: Model Design Accelerator	43
Explorer Tree	43
Explorer Diagram	44
Model Accelerator Workspace	45
Query Subject Diagram	45
Relationship Editing Mode	45

Model Warning View	46
Change the Settings for Diagrams	47
Create a Project	47
Create a Star Schema	48
Managing Your Star Schema	50
Chapter 4: Importing Metadata from Data Sources	51
Data Sources	51
Data Source Security	51
Types of Data Source Connections	52
Native Metadata	52
Working With Data Source Connections	53
Create a Data Source Connection	56
Importing Metadata	58
Import Metadata from a Relational Database	59
Import Metadata from an IBM Cognos Model	61
Import Metadata from an Architect Model or an Impromptu Catalog	61
Import Metadata from IBM Cognos DecisionStream or IBM Cognos Data Manager	62
Import from IBM Metadata Sources	66
Import Metadata From Third Party Metadata Sources	69
Troubleshooting Metadata from Other Sources	74
Import Metadata Using XML as a Data Source	75
Import Objects with the Same Name	76
Chapter 5: Modeling Relational Metadata	77
Relationships	78
Cardinality	78
Modify a Relationship	82
Create a Complex Expression for a Relationship	82
Create a Relationship	83
Create a Relationship Shortcut	83
Detect and Generate Relationships	84
Query Subjects	85
Data Source Query Subjects	85
Model Query Subjects	87
Stored Procedure Query Subjects	88
Determinants	92
Create a Model Query Subject Based on Existing Objects	96
View Related Objects	97
Create a Query Set	97
Test a Query Subject or Query Set	101
Validate a Query Subject	104
Update Query Subjects	104
Convert a Query Subject into a Dimension	105
Convert a Model Query Subject into a Data Source Query Subject	106
Edit the SQL	106
Change the Type of SQL	107
Change How the SQL Is Generated	112
Dimensions	114
Normalized Data Sources	114

Create a Regular Dimension	115
Sort Members of a Level	121
Roles	122
Create a Measure Dimension	124
Convert a Measure into a Query Item	125
Scope Relationships	126
Create a Regular Dimension Based on Existing Objects	127
View Related Objects	127
Test a Dimension	128
Convert a Regular Dimension into a Query Subject	130
Multilingual Metadata	131
Setting Up a Multilingual Reporting Environment	131
Using a Macro to Model Multilingual Data	134
Add a Language to a Project	134
Export a Translation Table	135
Import a Translation Table	136
Example - Create a Multilingual Project for Relational Metadata	136
Query Items	138
Modifying How Query Items Are Aggregated	141
Format Query Items	148
Define a Prompt Control	149
Convert a Query Item into a Measure	154
Adding Business Rules	155
Create a Calculation	155
Create a Filter	158
Apply a Filter	160
Example - Show the Currency Name for Each Country	162
Create a Parameter Map	163
Example - Specifying a Language Value for Relational Metadata	165
Create a Session Parameter	165
Using Parameters with Relational Data Source Query Subjects	167
Creating Prompts with Query Macros	168
Organizing the Model	179
Create a Star Schema Group	180
Use Shortcuts	184
Create a Folder or Namespace	187
Create a Query Item Folder	188
Create a Measure Folder	188
Create a Durable Model	189
Analyze a Model	191
Chapter 6: Working with SAP BW Metadata	197
Import from an SAP BW Data Source	197
Mapping SAP BW Objects to Framework Manager	203
Dimensions	204
Modify a Regular Dimension	205
Roles	210
Modify a Key Figures Dimension	212
View Related Objects	213

Test a Dimension or Other Object	213
Working with Model Query Subjects	216
Query Items	219
Modifying How Query Items Are Aggregated	222
Format Query Items	225
Define a Prompt Control	226
SAP BW Variables	230
Numeric Variable Property Values	234
Characteristic Variable Property Values	235
Picklist Prompts	237
Adding Business Rules	237
Create a Calculation	237
Create a Filter	239
Apply a Filter	242
Create a Parameter Map	243
Create a Session Parameter	245
Organizing the Model	246
Use Shortcuts	247
Create a Folder or Namespace	248
Chapter 7: Publishing Packages	251
Verify a Model or Package	251
Create or Modify a Package	254
Security	256
Users, Groups, and Roles	256
Add Data Security	257
Add or Remove Object Security	259
Modify Package Security	261
Specify Languages	262
Set Suppression Options	263
Externalizing Query Subjects and Dimensions	263
Publish a Package	266
Publish a Package Based on an OLAP Data Source	268
Publishing a Package by Running a Script	269
Update a Report to Use the Latest Version of a Package	270
Chapter 8: Managing the Project	271
Understanding the Metadata in Your Model	271
Explore a Package	271
View the Distribution of an Object in Packages	272
Create Model Documentation	272
Multiuser Modeling	273
Branching and Merging Projects	273
Segmenting and Linking Projects	280
Using External Repository Control	284
Administering the Metadata	284
Copy, Move, Rename, or Delete a Project	285
Analyze the Impact of Changes to a Package	287
Remap an Object to a New Source	290
Export Metadata	291

Project Reuse	292
Model Portability	294
Synchronize Projects	301
Query Behavior	304
Set Governors	304
Specify Where Aggregate Rollups are Processed	311
Improve Performance by Setting Query Processing Type	312
Improving Performance by Reusing Cached Data When Running a Report	313
Select Function Sets	314
Quality of Service	315
Control and Optimize How Queries Are Run	317
Chapter 9: Guidelines for Modeling Metadata	319
Understanding IBM Cognos Modeling Concepts	319
Relational Modeling Concepts	320
Model Design Considerations	329
Dimensional Modeling Concepts	336
Building the Relational Model	338
Defining the Relational Modeling Foundation	338
Defining the Dimensional Representation of the Model	345
Organizing the Model	348
Chapter 10: The SQL Generated by IBM Cognos Software	351
Understanding Dimensional Queries	351
Single Fact Query	351
Multiple-fact, Multiple-grain Query on Conformed Dimensions	352
Modeling 1-n Relationships as 1-1 Relationships	355
Multiple-fact, Multiple-grain Query on Non-Conformed Dimensions	357
Resolving Ambiguously Identified Dimensions and Facts	360
Query Subjects That Represent a Level of Hierarchy	360
Resolving Queries That Should Not Have Been Split	361
Chapter 11: Upgrading Models	365
Verifying the model before upgrading	365
Opening and upgrading the model	365
Upgrade and governors	366
Upgrade and data types	366
Upgrade and query subjects that are based on SAP BW metadata	367
Verifying and repairing the upgraded IBM Cognos ReportNet model	367
Converting dimension information to either determinants or dimensions	368
Selecting and repairing objects in the upgraded IBM Cognos ReportNet model	371
Upgrading segmented and linked projects	372
Appendix A: Troubleshooting	375
Unable to Compare Two CLOBs in Oracle	375
An Out of Memory Error with ERWin Imported Metadata	375
Framework Manager Cannot Access the Gateway URI	375
Object Names Appear in the Wrong Language	376
Full Outer Joins in Oracle Return Incorrect Results	376
Error When Testing Query Subjects in a Model Imported from Teradata	376

Error for Type-In SQL Query Subject	377
QE-DEF-0259 Error	377
Externalized Key Figures Dimension Retains Old Prompt Value	378
Older Models Display Level Object Security	378
Exporting a Framework Manager Model to a CWM File Fails With Error <i>MILOG.TXT was not found</i>	378
Difference in SQL for Inner Joins After Upgrading to IBM Cognos BI, Version 8.3 and Later	378
Full Outer Joins Not Sent to Oracle 9i and 10GR1	379
Unexplained Discrepancies in Number Calculations	379
Appendix B: Using the Expression Editor	381
Searching for Values May Return Unexpected Results	382
Calculation Components	382
Operators	383
Summaries	391
Member Summaries	402
Constants	405
Constructs	407
Business Date/Time Functions	408
Block Functions	413
Macro Functions	414
Common Functions	424
Dimensional Functions	431
DB2	453
Informix	469
MS Access	475
Nettezza	483
Oracle	490
Red Brick	499
SQL Server	504
Teradata	511
SAP BW	517
Sybase	519
Postgres	527
Vertica	533
Paraccel	538
MySQL	541
Greenplum	545
Report Functions	551
Appendix C: Accessibility features	571
Accessibility features in Framework Manager	571
Keyboard Shortcuts for Framework Manager	571
Keyboard shortcuts for Model Design Accelerator	572
IBM and accessibility	573
Appendix D: Data Formatting Reference	575
Data Formatting Properties	575
"Not Applicable" Characters	575
Any Error Characters	575

Calendar Type	575
Clock	575
Currency	575
Currency Display	576
Currency Symbol	576
Currency Symbol Position	576
Date Ordering	576
Date Separator	576
Date Style	576
Decimal Separator	576
Display AM / PM Symbols	576
Display As Exponent	576
Display Days	577
Display Eras	577
Display Hours	577
Display Milliseconds	577
Display Minutes	577
Display Months	577
Display Months	577
Display Seconds	577
Display Time Zone	577
Display Weekdays	577
Display Years	578
Display Years	578
Divide By Zero Characters	578
Exponent Symbol	578
Group Size (digits)	578
International Currency Symbol	578
Mantissa (digits)	578
Maximum No. of Digits	578
Minimum No. of Digits	579
Missing Value Characters	579
Negative Pattern	579
Negative Sign Position	579
Negative Sign Symbol	579
No. of Decimal Places	579
Numeric Overflow Characters	579
Padding Character	579
Pattern	579
Percentage Symbol	580
Percent Scale (integer)	580
Scale	580
Secondary Group Size (digits)	580
Security Error Characters	580
Thousands Separator	580
Time Separator	580
Time Style	580
Time Unit	581
Use Thousands Separator	581

Zero Value Characters	581
Appendix E: Using Patterns to Format Data	583
Pattern Guidelines	583
Date and Time Symbols	584
Decimal Format Symbols	591
Appendix F: Guidelines for Working with SAP BW Data for Use in Transformer	593
Working with SAP BW Data Using a Package in Framework Manager	593
Creating a BW Query in SAP Business Explorer Query Designer	594
Creating a Package in Framework Manager	597
Creating a Model in Transformer	600
Working with SAP BW Data Using Externalized CSV Files in Framework Manager	602
SAP BW Query Requirements	603
Framework Manager Considerations	605
Building PowerCubes from SAP BW Data	606
Appendix G: Reserved Words	609
Appendix H: XML Data Types	611
Glossary	613
Index	617

Introduction

IBM® Cognos® Framework Manager is a metadata modeling tool. A model is a business presentation of the information in one or more data sources. When you add security and multilingual capabilities to this business presentation, one model can serve the needs of many groups of users around the globe.

This document includes the procedures, examples, notes, tips, and other background information to help you prepare a model for reporting and deploying a package.

Note: For information about modeling for use with Dynamic Query Mode, see the *Dynamic Query Guide*.

Audience

This document is intended to help data modelers use Framework Manager. Before using Framework Manager, you should understand data modeling and how to write queries.

Finding information

To find IBM® Cognos® product documentation on the web, including all translated documentation, access one of the IBM Cognos Information Centers at <http://publib.boulder.ibm.com/infocenter/cogic/v1r0m0/index.jsp>. Updates to Release Notes are published directly to Information Centers.

You can also read PDF versions of the product release notes and installation guides directly from IBM Cognos product disks.

Using quick tours

Quick tours are short online tutorials that illustrate key features in IBM Cognos product components. To view a quick tour, start IBM Cognos Connection and click the **Quick Tour** link in the lower-right corner of the Welcome page. Quick Tours are also available in IBM Cognos Information Centers.

Forward-looking statements

This documentation describes the current functionality of the product. References to items that are not currently available may be included. No implication of any future availability should be inferred. Any such references are not a commitment, promise, or legal obligation to deliver any material, code, or functionality. The development, release, and timing of features or functionality remain at the sole discretion of IBM.

Samples disclaimer

The Great Outdoors Company, GO Sales, any variation of the Great Outdoors name, and Planning Sample depict fictitious business operations with sample data used to develop sample applications for IBM and IBM customers. These fictitious records include sample data for sales transactions, product distribution, finance, and human resources. Any resemblance to actual names, addresses, contact numbers, or transaction values is coincidental. Other sample files may contain fictional

data manually or machine generated, factual data compiled from academic or public sources, or data used with permission of the copyright holder, for use as sample data to develop sample applications. Product names referenced may be the trademarks of their respective owners. Unauthorized duplication is prohibited.

Accessibility features

Accessibility features help users who have a physical disability, such as restricted mobility or limited vision, to use information technology products. This product has accessibility features. For information on these features, see the accessibility section in this document.

Chapter 1: What's New?

This section contains a list of [new](#), [changed](#), [deprecated](#), and [removed](#) features for this release of IBM® Cognos® Framework Manager. It also includes a cumulative list of similar information for previous releases. It will help you plan your upgrade and application deployment strategies and the training requirements for your users.

For information about upgrading, see the IBM® Cognos® *Installation and Configuration Guide* and ["Upgrading Models"](#) (p. 365).

For information about new features for this release, see the IBM® Cognos® *New Features Guide*.

To review an up-to-date list of environments supported by IBM Cognos products, such as operating systems, patches, browsers, Web servers, directory servers, database servers, and application servers, visit www.ibm.com.

New Features in 10.1.0

Listed below are new features since the last release. Links to directly-related topics are included.

Model Design Accelerator

The Model Design Accelerator is a graphical utility designed to guide both novice and experienced modelers through a simplified modeling process. Novice modelers will have the capability of easily building powerful models without extensive experience and training. Experienced modelers can accelerate the modeling process so that the overall time to build a model is reduced. For more information, see ["Model Design Accelerator"](#) (p. 43).

Durable Models

When you create an IBM® Cognos® Framework Manager model using the durable model capability, you can rename the query items in your model without breaking references to the changed names in existing reports. For more information, see ["Create a Durable Model"](#) (p. 189).

Dynamic Query Mode

In Framework Manager, you can specify that a package based on a supported data source can use the new dynamic query mode, which is an enhanced Java™-based query execution mode. This mode offers improved query performance and functionality, security-aware caching, and native data interfaces to leverage 64-bit technology. For more information about supported data sources, see the *Administration and Security Guide*.

Use dynamic query mode by setting the **Use Dynamic Query Mode** switch on the **Options** page of the Publish Wizard. For more information, see ["Publish a Package"](#) (p. 266).

More Information about Dynamic Query Mode

For more information about dynamic query mode, see the documents listed in the following table.

What are you looking for?	Where to find the information
An overview of the dynamic query mode, its benefits, and considerations when using it.	<i>Dynamic Query Guide</i>
Detailed information about techniques and product behaviors of the dynamic query mode.	IBM Cognos 10 <i>Dynamic Query Cookbook</i>
Information about enabling connectivity for data sources supported by the dynamic query mode.	<i>Installation and Configuration Guide</i>
Information about query service administration, including caching and query service properties.	<i>Administration and Security Guide</i>
Information about publishing packages for the dynamic query mode.	<i>Framework Manager User Guide</i>
Information about testing reports in the dynamic query mode prior to upgrade.	<i>Lifecycle Manager User Guide</i>
Information about using the IBM Cognos Software Development Kit to administer query service properties and develop client applications to use dynamic query mode.	IBM Cognos Software Development Kit <i>Developer Guide</i>

Creating Packages for SAP BW Data Sources

You can create packages for SAP BW cubes and queries directly in IBM® Cognos® Connection. This does not change the Framework Manager functionality for creating and publishing SAP BW packages. Regardless of where the packages are created, they are identical and can be used in the same way by the IBM Cognos studios.

For more information, see the section about packages in the *Administration and Security Guide*.

Using SAP BW Time-dependent Hierarchies

Time-dependant hierarchies now automatically reflect hierarchy or structure changes. When a structure is imported into Framework Manager, each SAP BW time hierarchy is depicted as an individual level. Report Studio users can use these structures to report on and compare levels that are valid for a specific time period.

Function Description Improvements

The functions that you can use to create calculations now include improved descriptions and more examples. The descriptions and examples appear in the Framework Manager user interface and in this user guide. For more information about functions, see ["Using the Expression Editor" \(p. 381\)](#).

New Vendor-specific Functions

Functions that are specific to Postgres, Vertica, Netezza, Paracel, MySQL, and Greenplum now appear in the expression editor in the Vendor Specific Functions folder. For more information, see ["Using the Expression Editor" \(p. 381\)](#).

Changed Features in 10.1.0

Listed below are changes to features since the last release. Links to directly-related topics are included.

Deprecated Features in 10.1.0

A deprecated feature is one that is being replaced by a newer version or a better implementation. The intent is to discontinue the use of the feature and provide recommendations for adapting to this change over multiple releases.

Listed below are deprecated features:

Removed Features in 10.1.0

The functionality listed below has been removed from version 10.1.

Repository Control

Native support for CVS and Microsoft® Visual SourceSafe repositories was removed from this release. If you use this feature, and still require project versioning functionality, we recommend that you read the section ["Using External Repository Control" \(p. 284\)](#). This section explains how to maintain project versioning outside the modeling application, and is applicable for use with all version control systems.

Conformed Dimensions on SAP BW Data Sources

Support for conformed dimensions generated by IBM® Cognos® Framework Manager for SAP BW data sources has been removed from this release.

New Features in 8.4

Listed below are new features since the last release. Links to directly-related topics are included.

- Options for configuring global test options are now available [\(p. 30\)](#).
- You can configure IBM® Cognos® Framework Manager to automatically save projects at defined time intervals [\(p. 30\)](#).
- You can reorder root namespace objects listed in the Project Viewer. Objects can be reordered in ascending or descending order, based on their names [\(p. 31\)](#). You can choose to include children or all descendants when reordering objects.

- The project folder includes new files (p. 23). These files include archive-log.xml, customdata.xml, session-log.xml, and session-log-backup.xml. The project folder no longer includes the persistence.txt file.
- You can do screen captures in the content explorer as well as in the Diagram.
- Over time, log files for a project can become large. For improved performance, you can now archive entries in log files (p. 301).
- You can print diagrams in the Context Explorer, preview before printing, and change page layout options using **Page Setup**.
- You can set suppression options for published packages "**Set Suppression Options**" (p. 263).
- For dimensionally modeled relational metadata, you can specify sort characteristics on a dimension. You can now also specify sorting on individual levels within the dimension (p. 121).
- New governors have been added to allow further control of system resources and performance (p. 304).
- A new property, Allocation Rule, allows you specify the type of allocation defined for the measure "**Query Items**" (p. 138).

Changed Features in 8.4

Listed below are changes to features since the last release. Links to directly-related topics are included.

- You can upgrade segmented projects via the main project. The child segments are automatically upgraded.
- Query reuse is the default setting in IBM® Cognos® Framework Manager models and IBM Cognos Report Studio reports. To disable query reuse, change the **Allow Usage of Local Cache** governor.
- You can use Visual Source Safe 2005 as a repository to manage your projects in Framework Manager.
- Prompt types set on attributes are now processed. The report user will see the prompt that matches the prompt type on the attribute. Because prompt types on attributes were not processed in the previous release, some differences may occur.
- Many commands are available from shortcut menus. For example, to quickly rename a project, right-click the project name in the Project Viewer and click **Rename**. Many commands on shortcut menus have been reorganized to make them easier to find and use. Some examples include:
 - In the Diagram tab of the Project Viewer, you can cut, copy, paste, and delete a selected model object. In the previous release, the commands for these actions were listed on the first level of the right-click menu. In this release, these commands are grouped using an Edit command. For example, to copy a model object, right-click the object and click **Edit, Copy**.

- In the Explorer tab of the Project Viewer, you can view details about a selected object. In the previous release, the Diagram Settings command was listed on the first level of the right-click menu. In this release, this command appears under a Navigate Diagram command, along with related commands for working with diagrams.

Composite Information Server is Replaced By IBM Cognos 8 Virtual View Manager

Composite Information Server was available with earlier releases of IBM Cognos 8. In the current release, Composite Information Server is replaced by IBM Cognos 8 Virtual View Manager, which is an IBM proprietary product that is based on a new version of Composite Information Server. In this release, the default repository is changed, from Microsoft® SQL Server to IBM® Informix®. If you have Composite data sources defined in IBM Cognos Connection, you must migrate the existing repository to the new default repository. For more information about migrating the repository, see the Virtual View Manager User Guide. For more information about data source connections, see the *Administration and Security Guide*.

Deprecated Features in Version 8.4

A deprecated feature is one that is being replaced by a newer version or a better implementation. The intent is to discontinue the use of the feature and provide recommendations for adapting to this change over multiple releases.

Listed below are deprecated features:

Conformed Dimensions on SAP BW Data Sources

Conformed dimensions on SAP BW data sources generated by IBM® Cognos® Framework Manager will continue to be supported in this release, but will not be enhanced. Support for conformed dimensions on SAP BW data sources generated by Framework Manager will be deprecated in the next major release of IBM Cognos 8.

New Features in Version 8.3

Listed below are new features since the last release. Links to directly-related topics are included.

- You can analyze the metadata in a model by using the **Model Advisor**, an automated tool that applies current modeling guidelines and identifies inconsistencies and areas in your model that you need to examine. For more information, see ["Analyze a Model" \(p. 191\)](#).
- You can specify the behavior of shortcuts by using the **Shortcut Processing** governor and the **Treat As** property. For more information, see ["Set Governors" \(p. 304\)](#) and ["Use Shortcuts" \(p. 184\)](#).
- You can minimize the effect of model changes and data source changes by using the **Remap To New Source** command. This command remaps higher-level model objects so that they continue to run and return correct data. You can remap query items, measures, calculations, and filters. You can remap individual objects manually or you can remap multiple objects at the same time. For more information, see ["Remap an Object to a New Source" \(p. 290\)](#).

- You can understand the full impact of potential changes in the model by using the **Show Object Dependencies** command to find the objects that depend on an object. For more information, see ["Show Object Dependencies"](#) (p. 289).
- You can branch and distribute a model for development and later collect and merge the changes made by multiple modelers by using the **Branch to** and **Merge from** commands. For more information, see ["Branching and Merging Projects"](#) (p. 273).
- You can import SAP BW queries that contain dual structures and use the structures in IBM® Cognos® 8 queries to control the amount and order of information that your users see by using the **SAP BW Dual Structures Support** check box. For more information, see ["SAP BW Structures"](#) (p. 201).

Changed Features in Version 8.3

Listed below are changes to features since the last release. Links to directly-related topics are included.

- You can now publish packages from IBM® Cognos® Framework Manager into any folder in IBM Cognos Connection.
- With the appropriate capabilities and permissions, your users can now add PowerCubes directly from IBM Cognos Connection without having to access Framework Manager.
- Before publishing a package and running reports, you can now see more information about how the changes you make to a model will affect the package and the reports that use it by using the **Analyze Publish Impact** command. For more information, see ["Analyze the Impact of Changes to a Package"](#) (p. 287).
- The **Verify Model** dialog box is improved. You can now select the level of verification that you want to perform. You can sort and group the reported items by type and by severity of the message. For more information, see ["Verify a Model or Package"](#) (p. 251).
- Diagrams in the **Diagram** tab are improved for performance, stability, and usability. You can now control the settings for diagrams and the **Context Explorer**. For more information, see ["The Diagram Tab"](#) (p. 32) and ["Change the Settings for Diagrams"](#) (p. 33).
- Search and replace is improved so that you can do bulk searching and replacing. Use the **Search** tab to quickly find objects by applying different search criteria, such as the location, class, condition, or property. Text properties such as Name, Description, and Screen Tip, are writable so you can easily replace multiple values. For more information, see ["The Search Tab"](#) (p. 36) and ["Replacing Multiple Property Values"](#) (p. 34).

Deprecated Features in Version 8.3

A deprecated feature is one that is being replaced by a newer version or a better implementation. The intent is to discontinue the use of the feature and provide recommendations for adapting to this change over multiple releases.

Listed below are deprecated features.

Framework Manager Security Filters for SAP BW: Notice of Intent to Change the Default Setting

In all shipped versions of IBM® Cognos® ReportNet® and IBM Cognos 8 including IBM Cognos 8.2, the following behavior has been enabled by default.

Multiple security filters defined within IBM Cognos Framework Manager on metadata imported from SAP BW sources are combined using 'AND' logic, effectively an intersection of a particular user's permissions. This behavior is contradictory to corresponding behavior on relational data sources where similar filters are combined using 'IN' and 'OR' (union) logic to facilitate cases where users belong to one or more group and require a union of their permissions. The current default behavior for SAP BW datasources has been determined to be a product defect and will be changed in the IBM Cognos 8.3 release to align with the behavior of relational data sources. In the IBM Cognos 8.1 Mr2 and IBM Cognos 8.2 releases, it will be possible to get the union of filters behavior by modifying the following switches in the qfs_config.xml configuration file. Under `<provider name="OlapQueryProvider" libraryName="oqp">`, add the following new `<parameter>` element:

```
<parameter name="ORingSecurityFiltersWhenUserBelongsToMultipleGroups"
value="true"/>
```

Effect

When multiple security filters are defined in the Data Security setting in Framework Manager for a query subject, if a user belongs to more than one user group associated with these filters, the effect of this switch is to perform a union of these filters instead of an intersection. For example, if a user Joe belongs to a corporate group allowing him to see data for Asia, Europe, and America, and also belongs to a regional group allowing him only to see data for Europe, the effect of the switch will be to let Joe see data for Asia, Europe, and America (as compared to just Europe) when logging on to IBM Cognos 8 and authoring or running a report. Note that security filters that are based on other security filters in the Data Security settings for a query subject continue to be intersected, just as before, independent of whether or not the new switch is activated.

As of IBM Cognos 8.3, this behavior will become the new default for SAP BW metadata in Framework Manager. There will be no change to the behavior of security filters applied on relational sources. The switch will continue to be available and can be set to false by your administrator if required to maintain existing application behavior.

IQD Externalize Method

IQDs generated by Framework Manager will continue to be supported in this release, but will not be enhanced. Support for IQDs generated by Framework Manager will be deprecated in the next major release of IBM Cognos 8. Transformer will continue to support IQDs generated by Impromptu in the next major release.

Removed Features in Version 8.3

The functionality listed below has been removed from version 8.3.

- The **Embedded** externalize method has been removed from the current release. There is no impact on product functionality.

Chapter 2: Getting Started with Framework Manager

IBM® Cognos® Framework Manager is a metadata modeling tool that drives query generation for IBM Cognos software. A model is a collection of metadata that includes physical information and business information for one or more data sources. IBM Cognos software enables performance management on normalized and denormalized relational data sources and a variety of OLAP data sources. When you add security and multilingual capabilities, one model can serve the reporting, ad hoc querying, and analysis needs of many groups of users around the globe.

Before doing anything in IBM Cognos Framework Manager, we recommend that you thoroughly understand the reporting problem that you want to solve ([p. 21](#)).

To get started, we recommend that you do the following:

- ☐ Learn about the objects you will use ([p. 23](#)).
- ☐ Create ([p. 26](#)) or open ([p. 27](#)) a project.
- ☐ Explore the panes in Framework Manager ([p. 28](#)).
- ☐ Explore the sample models included with Framework Manager ([p. 39](#)).

Analyze the Problem

Before you start, you must understand the reporting problem that you are trying to solve and what data is available to solve it.

If you cannot address the following questions, talk to your users about their reporting requirements:

- ☐ Do you and your users agree on the reporting requirements?

Issues to resolve can include multilingualism, performance, security, and how to organize and combine query items and filters.

- ☐ Does the data source contain the data and metadata that you need?

Without metadata such as primary keys, indexes, and foreign keys, your reports may take too long to run, or may produce incorrect results. If the data source does not contain the data and metadata that you need, will it be changed, or will you work around it?

- ☐ Does the same data exist in more than one source?

If so, choose the data source that most closely fits your reporting requirements. If a data warehouse is available, it is typically a better choice than an operational database. A data warehouse based on a star schema is ideal. If this does not exist, and you expect that your reporting application will be heavily used, consider arranging for one to be created.

- ☐ Which data source tables are the fact tables, which are the dimensions, and which are both fact table and dimension?

- ❑ What are the keys and attributes of each dimension?
- ❑ Which relationships are required?
- ❑ Are there multiple relationship paths between tables?

If so, what does each path represent? You must define the preferred path for each.

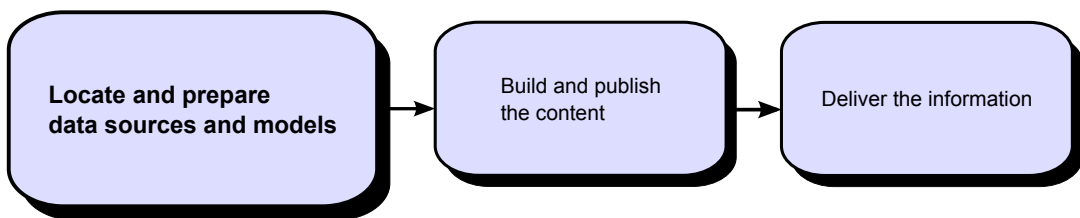
Then you should review the names of data sources, tables, and columns in your data source to ensure that you are not using names reserved by IBM® Cognos®. If you must use a reserved word, enclose the word in quotes in the SQL specification. For example, `select Orderdate, "Timezone"`. For more information, see ["Reserved Words"](#) (p. 609).

Building IBM Cognos Business Intelligence Applications

The lifetime of an IBM® Cognos® Business Intelligence application can be months, or even years. During that time, data may change and new requirements appear. As the underlying data changes, authors must modify existing content and develop new content. Administrators must also update models and data sources over time. For more information about using data sources, see the IBM Cognos *Administration and Security Guide* and the IBM Cognos Framework Manager *User Guide*.

In a working application, the technical and security infrastructure and the portal are in place, as well as processes for change management, data control, and so on. For information about the workflow associated with creating IBM Cognos BI content, see the IBM Cognos *Architecture and Deployment Guide*. For additional information, see the IBM Cognos Solutions Implementation Methodology toolkit, which includes implementation roadmaps and supporting documents. Information about the toolkit is available on www.ibm.com.

The following graphic provides an overview for how to use IBM Cognos BI to build applications across all of your IBM Cognos BI components.



- ❑ **Locate and prepare data sources and models**

IBM Cognos BI can report from a wide variety of data sources, both relational and dimensional. Database connections are created in the Web administration interface, and are used for modeling, for authoring, and for running the application.

To use data for authoring and viewing, the business intelligence studios need a subset of a model of the metadata (called a package). The metadata may need extensive modeling in Framework Manager.

- ❑ **Build and publish the content**

Reports, scorecards, analysis, dashboards and more are created in the business intelligence studios of IBM Cognos BI. Which studio you use depends on the content, lifespan, and audience of the report, and whether the data is modeled dimensionally or relationally. For example, self-

service reporting and analysis are done through IBM Cognos Business Insight Advanced, IBM Cognos Query Studio, and IBM Cognos Analysis Studio, and scheduled reports are created in IBM Cognos Report Studio. Report Studio reports and scorecards are usually prepared for a wider audience, published to IBM Cognos Connection or another portal, and scheduled there for bursting, distribution, and so on. You can also use Report Studio to prepare templates for self-service reporting.

❑ Deliver and view the information

You deliver content from the IBM Cognos portal or other supported portals, and view information that has been saved to portals, or delivered by other mechanisms. You can also run reports, analyses, scorecards, and more from within the business intelligence studio in which they were created.

For information about tuning and performance, see the IBM Cognos *Administration and Security Guide* and www.ibm.com.

Objects You Will Use

When you work in IBM® Cognos® Framework Manager, you work with a number of objects that are contained in a project.

Projects

A project contains a model, namespaces, packages, data sources, and related information for maintaining and sharing model information. A single project can span many data sources or tables.

An IBM Cognos Framework Manager project appears as a folder that contains a project file (.cpf) and the specific .xml files that define the project. The files in a project folder are unique to each project. The project and its associated files are contained in a project folder.

We do not recommend adding secondary files to the project folder because they may be affected by actions such as move, rename, and delete commands on the **Manage Projects** menu. If you decide to add secondary files to the project folders, the files are added with absolute paths. If they are moved from the original location, they must be retargeted.

These are the contents of a project folder.

File name	Description
<project name>.cpf	The Framework Manager project file, which references the .xsd and .xml files that define a project.
archive-log.xml	This file contains the portion of the main log file that was archived.
customdata.xml	This file contains the layout information for the diagram. If this file is deleted, layout information is lost. An automatic layout will be applied.

File name	Description
IDLog.xml	This file tracks objects for models that use branching and merging.
log.xml	A list of all modifications made to the model.
mda_metadata.xml	A Model Design Accelerator file, which contains the metadata imported from data sources.
mda_engine_project.xml	A Model Design Accelerator file, which contains the definition of the star schema.
model.xml	The actual model data created by Framework Manager users.
preferences.xml	The preferences for Framework Manager projects.
session-log.xml	<p>A list of unsaved transactions in the model. When the project is saved, this list is deleted. View contents of this file using View Transaction History.</p> <p>When Framework Manager is started, the existing session-log.xml file is renamed to session-log-backup.xml.</p>
session-log-backup.xml	<p>The session-log.xml from the previous session. Using this file, a modeler can run a script to restore the unsaved model transactions in the event of an unexpected interruption in the current session.</p> <p>This file is deleted each time Framework Manager is started. Ensure you make a copy of this file before exiting the current Framework Manager session if you want to keep a copy.</p>
repository.xml	The logged version history for each project or segment that was added to a repository; this file exists only if you added projects to a repository.
upgradeReport.htm	The content of the upgrade summary message that is displayed after upgrade.

Models

A model is the set of related dimensions, query subjects, and other objects required for one or more related reporting applications.

The Framework Manager model is a metadata layer that adds value to a data source in several ways. Most importantly, it provides a business view of the information in the source data to simplify building reports, analyses, and queries. The business view can

- organize items in folders that represent business areas for reporting

- format items using numeric, currency, date, time, and other formats
- present multilingual folder and item names, descriptions, tips, and data so that users can operate in their language of choice
- automate the generation of SQL queries sent to the relational data source
- specify default prompting

This can include having IBM® Cognos® software prompt the user using a descriptive name while actually filtering on a code or key value for improved query performance.

In particular, you can modify the Framework Manager model to ensure that queries sent to the data source are efficient, well formed, and secure. You can specify the rules governing query generation, restrict user access to specific rows or columns of data, and model data relationships to hide the complexity of data from your users.

Namespaces

A namespace uniquely identifies query items, dimensions, query subjects, and other objects. You import different databases into separate namespaces to avoid duplicate names.

Packages

A package is a subset of the dimensions, query subjects, and other objects defined in the project. A package is what is actually published to the IBM Cognos BI server, and it is used to create reports, analyses, and ad hoc queries.

Dimensions

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 with in IBM® Cognos® Framework Manager are regular dimensions and measure dimensions. In SAP BW, measure dimensions are called key figures.

Query Subjects

A query subject is a set of query items that have an inherent relationship.

In most cases, query subjects behave like tables. Query subjects produce the same set of rows regardless of which columns were queried.

There are different types of query subjects:

- data source

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

- model

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.

- stored procedure

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

Query Items

A query item is the smallest piece of the model that can be placed in a report. It represents a single characteristic of something, such as the date that a product was introduced.

Query items are contained in query subjects or dimensions. For example, a query subject that references an entire table contains query items that represent each column in the table.

For your users, query items are the most important objects for creating reports. They use query item properties of query items to build their reports.

Create a Project

In IBM® Cognos® Framework Manager, you work in the context of a project. The project contains objects that you organize for your users according to the business model and business rules of your organization. You view these objects in the project page ([p. 28](#)).

Before you can import metadata, you must create a project.

For information about creating a project segment, see "[Create a Segment](#)" ([p. 282](#)).

For information about creating a project using the Model Design Accelerator, see "[Create a Project](#)" ([p. 47](#)).

Steps

1. From the **Welcome** page, click **Create a new project**.

Tip: If you are in Framework Manager, click **New** from the **File** menu.

2. In the **New Project** page, specify a name and location for the project, and click **OK**.
3. In the **Select Language** page, click the design language for the project.

You cannot change the language you select after you click **OK**, but you can add other project languages. For more information, see "[Add a Language to a Project](#)" ([p. 134](#)).

For more information about choosing the proper design language for durable models, see "[Create a Durable Model](#)" ([p. 189](#)).

Note: If an SAP BW server does not support the selected language, it uses the content locale mapping in IBM Cognos Configuration. If a mapping is not defined, Framework Manager uses the default language of the SAP BW server.

4. Click **OK** to select the design language.

The **Metadata Wizard** appears.

5. Choose whether to import your metadata now or later:

- To import now, select the import source and click **Next**.

- To delay importing metadata, click **Cancel**.
6. If you chose to import the metadata now, follow the instructions in the **Metadata Wizard**:
- Select a data source connection and click **Next**.
If the data source connection you want is not listed, you must first create it ([p. 56](#)).
 - Select the check boxes for the objects you want to import.
 - Specify how the import should handle duplicate object names. Choose whether to import and create a unique name. If you choose to create a unique name, the imported object appears with a number. For example, you see QuerySubject and QuerySubject1 in your project.
 - If you want to import system objects, select the **Show System Objects** check box, and then select the system objects that you want to import.
 - Specify the criteria to use to create relationships and click **Import**.
For more information, see "[Relationships](#)" ([p. 78](#)).
- You see a list of objects that could not be imported and a count of objects that were imported.
7. Click **Finish**.
- Save the project file (.cpf) and all related files in one folder. When you save a project with a different name or format, ensure that you save the project in a separate folder.

Open a Project

You must open a project before you can import metadata or make changes to existing metadata.

If the project was created using a model schema that is older than the currently supported version, you are prompted to upgrade the model.

If your model is checked into a repository, you cannot upgrade it. Manually check the model out of the source control system and then open it in the new version of IBM® Cognos® Framework Manager.

If you upgrade a segmented model, you must open and upgrade each segment individually. After upgrading each segment, you can then upgrade the top level, or master, project.

Steps

1. From the **Welcome** page, click **Open a project**.
Tip: If you are in Framework Manager, click **Open** from the **File** menu.
2. Browse to locate the project folder and click the .cpf file.
3. Click **OK**.

The Project Page












After you create (p. 26) or open (p. 27) a project, the project page appears. The project page is where you design, package, and publish project metadata. This page contains several panes and views that you can use to view and modify the objects in a project.


















The Project Viewer














The **Project Viewer** shows the objects in a project in a hierarchical view. You can use the **Project Viewer** to view, modify, and create objects.

Relationships are shown in both the **Diagram** and **Explorer** tabs. Imported functions are shown in the **Explorer** tab.

The **Project Viewer** uses the following icons to represent objects and states. A project may use some or all of the icons.

Icon	Object
	Project
	Root namespace or any other namespace in the project
	Data source folder
	Data source
	Parameter map folder
	Parameter map
	Package
	Packages folder
	Published package
	Calculation
	Calculation whose Usage property is set to Attribute

Icon	Object
	Calculation whose Usage property is set to Identifier
	Embedded calculation
	Calculation whose Usage property set to Fact
	Model dimension based on existing model objects
	Dimension that is imported from a data source
	Dimension that is not valid. This dimension is imported from a data source.
	Filter
	Hierarchy
	Level in a hierarchy
	Measure
	Semi-additive measure
	Measure dimension based on existing model objects.
	Measure dimension imported from a data source.
	Invalid measure dimension. This dimension is imported from a data source.
	Query item
	Query item whose Usage property is set to Identifier
	Query item whose Usage property is set to Fact

Icon	Object
	Query item that is located under a shortcut query subject
	Query subject based on existing model objects
	Query subject imported from a data source
	Invalid query subject. This query subject is imported from a data source.
	Query subject based on multidimensional data
	Query subject that contains query items whose Usage property is set to Fact . This query subject is based on existing model objects.
	Query subject that contains query items whose Usage property is set to Fact . This query subject is imported from a data source.
	Query subject that contains query items whose Usage property is set to Fact . This query subject is imported from a data source and is not valid.
	Relationship
	Linked segment or project that was updated. This icon appears over other icons.
	Linked object. This icon appears over other icons.
	Shortcut. This icon appears over other icons.
	Invalid object. This icon appears over other icons.

Change Options for Projects

You can change one or more options for testing projects and saving changes automatically.

Test options apply globally to all tests run for this project. For information about testing selected objects, see ["Modeling Relational Metadata" \(p. 77\)](#) or ["Working with SAP BW Metadata" \(p. 197\)](#).

Saving projects automatically may help prevent loss of data if an unexpected interruption, such as a power outage, occurs. If you choose to save projects automatically, you can choose the time interval between save operations.

If you choose to use the auto save feature, IBM® Cognos® Framework Manager cannot save changes when a dialog box is open if that dialog box locks other areas of Framework Manager. For example, when using the dialog boxes presented by the **Create Folder** wizard, you cannot access commands from the menu bar. After you close the dialog box, Framework Manager will save changes automatically after the specified auto save interval has elapsed.

Steps

1. From the **Project** menu, click **Options**.
2. On the **Test Options** tab, choose the options that you want.

Goal	Action	Persistence
Limit the number of rows retrieved	<p>Select the Restrict the maximum number of rows to be returned check box and type the required number of rows.</p> <p>This setting does not improve performance for retrieving data when testing dimensions, query subjects, and query sets.</p>	<p>This setting applies to all dimensions, query subjects, and query sets in the model.</p> <p>This setting is saved and used in your next session with any project.</p>
Specify the level of detail	<p>Drag the Level of Information shown in Query Information slider to the location that represents the amount of detail you require.</p>	<p>This setting is saved and used in your next session with this project.</p>
Apply relevant design mode filters	<p>Select the Apply all relevant design mode filters when testing check box.</p> <p>This applies all relevant filters whose usage is set to design mode in another dimension, query subject, or query set.</p>	<p>This setting is saved and used in your next session with any project.</p>

3. If you want projects saved automatically, on the **Auto Save** tab, select **On** and type the number that represents the frequency, in minutes, that you want projects saved automatically.
4. Click **OK**.

Reorder Objects

By default, root namespace objects listed in the **Project Viewer** appear in the order they were added. You can change the order of objects based on their names. Objects may be reordered in ascending or descending order.

When selecting objects to reorder, all selections must be at the same level in the metadata tree. However, you can reorder only some of the objects on a level. The location of the reordered objects relative to the objects that were not selected is based on position of the first object in the reordered list. For example, when reordering in ascending order, the object with the name that begins with the letter closest to "A" remains in its current position in the **Project Viewer**, followed by the other selected objects.

You can include the children when reordering selected objects. For example, assume that two query subjects, Retailers West and Retailers East appear on one level. Each query subject has children, which appear one level below their parents. If you reorder Retailers West and Retailers East and choose to include their children, the **Project Viewer** lists Retailers East and then Retailers West on one level. The children of each query subject are grouped below their parent, and are listed in ascending order.

You can also choose to include the descendant of child objects. The time required to reorder objects may increase depending on the type and number of selected objects.

If selections include read-only objects or levels in hierarchies, IBM® Cognos® Framework Manager displays a message indicating that these items cannot be reordered.

Steps

1. In the **Project Viewer**, select the root namespace objects you want to reorder.
Ensure that selected objects are at the same level.
You can also select objects in the **Explorer** tab or **Diagram** tab.
2. From the **Tools** menu, click **Reorder**.
3. Click whether to reorder objects by name in ascending order or descending order.
4. Select whether to reorder the selected objects only, or to reorder the selected objects and their children.
If you choose to reorder children of selected objects, you can also include all descendants of the child objects.
5. Click **OK**.


The Explorer Tab

The **Explorer** tab shows the contents of a project, similar to any file system. Arrange objects by name, class, or description. If you have a large number of objects in a project, it may be easier to locate them in the **Explorer** tab.

You can use the **Explorer** tab to view, create, and modify objects and relationships. You can also create folders and namespaces to group objects.

The Diagram Tab

Use the **Diagram** tab to show the relationships between objects in a project. Relationships between objects are shown as lines with cardinality notation (p. 79). You can expand objects and namespaces to show the object hierarchy and the relationships between objects.

Tip: You can also control the model area that is visible in the diagram. Click and hold the **overview** button in the bottom right corner and drag the pointer over the diagram .

In the **Diagram** tab, you can do any of the following:

- View, create, and modify objects and relationships.
- Create folders and namespaces to group objects.
- Change the settings for the diagrams (p. 33).
- Change the layout of objects to either star layout or standard layout by clicking **Auto Layout** from the **Diagram** menu.
- Focus on an object by clicking **Set Focal Point** from the **Diagram** menu.
- Find an object by right-clicking the object in the **Project Viewer** and clicking **Locate in Diagram**.
- Zoom in or out by clicking **Zoom** from the **Diagram** menu.
- Expand or collapse all objects from the **Diagram** menu.
- Save the diagram for printing by clicking **Screen capture** from the **Diagram** menu and specifying the name of the picture.
- Launch the **Context Explorer** by right-clicking an object in the **Diagram** tab and clicking **Launch Context Explorer**.
- Print the diagram in the **Context Explorer** by right-clicking the **Context Explorer** background and click **Print**. A Print button is also available from the **Context Explorer** toolbar. This right-click menu also includes commands for previewing the diagram using **Print Preview** and changing page layout options using **Page Setup**.

Change the Settings for Diagrams

You can change one or more settings for diagrams. The **Diagram Settings** tab defines the settings for the main diagram. The **Context Explorer** tab defines how context diagrams are populated and displayed.

Steps to Define Main Diagram Settings

1. From the **Diagram** menu, click **Diagram settings**.
2. Set the level of detail.

You can select query items, relationships, scope relationships, cardinality, and descriptions.

3. Select the type of notation.

You can use Merise or Crowsfoot notation. By default, IBM® Cognos® Framework Manager uses Merise notation. Merise notation marks each end of the relationship with the minimum and maximum cardinality of that end. You can also use Crowsfoot notation, which provides a pictorial representation of the relationship.

4. To align objects more easily, turn the **Snap options** on.

5. To have a network of evenly spaced lines in the background, select the **Display grid** check box.
6. Select the font and color for text.
7. Select whether you want these settings to be the default for all new projects.
8. Click **OK**.

The Dimension Map Tab

You can use the **Dimension Map** tab to view, create, and modify hierarchies and levels for the dimension you selected in the **Project Viewer**. You can also view and modify scope relationships.

The **Measures** and **Attributes** tabs appear after you click the **Dimension Map** tab. Use the **Measures** tab to view or modify all the measures and scope relationships that are available in the model. Use the **Attributes** tab to view or modify the role of the selected query item.

You can also embed calculations in the query item.

Tip: The best way to view SAP BW metadata is in the star layout. From the **Diagram** menu, click **Auto Layout Diagram**, and then select **Star**.

The Properties Pane

The **Properties** pane shows the properties of the objects that you last selected in the **Project Viewer**, **Explorer** tab, **Diagram** tab, **Dimension Map** tab, **Dependencies** tab, **Search** pane, or **Summary** pane. Object properties are set during import, and some property values can be modified during modeling. You can use the **Properties** pane to add, modify, or delete the properties of objects.

You can modify the properties for multiple objects at one time. If you select more than one object, IBM® Cognos® Framework Manager shows only the properties that are common to all the objects. You can

- sort property values by double-clicking the property heading
An arrow appears to indicate the direction in which values are sorted. You can toggle between ascending and descending order.
- filter property values by clicking the arrow to the right of the property heading
You can either click a value, or click **Custom** to define the criteria for the rows that you want to view.
- apply a property value to multiple objects by clicking the arrow next to the property and dragging the highlighted area over the properties to which you want to apply that value
- resize the width of the rows and columns by right-clicking the object name in the property pane

If you need more room, you can resize the **Properties** pane or move it so that it floats in the window. For example, if you have more than one monitor, you can then move the **Properties** pane to another monitor.

Replacing Multiple Property Values

You can replace multiple values for each text string property.

When you replace multiple property values, IBM® Cognos® Framework Manager automatically updates elements that are dependent on the values you replaced. If any dependencies cannot be updated, such as data source and parameter map names, you must validate the model manually.

You can match either the entire property value, or part of the value. By matching part of the property value, you can perform multiple replacements on a single text property. For example, if you search for “nation” and replace it with "country", the property value "This nation and that nation" is replaced with "This country and that country". However, "national" will be replaced with "countryal".

Tip: Before replacing property values, enlarge the **Properties** pane and widen the column of the property whose value you want to replace.

Steps

1. In the **Project Viewer** window, select multiple objects.

Tip: To reduce the set of properties to search for before performing the replace, filter the properties first. On the **Properties** tab, right-click the column heading you want to filter on and click **Set Autofilter**.

2. Choose one of the following:

- If your model contains multiple languages, click the **Language** tab.
- If your model contains only one language, click the **Properties** tab.

3. Right-click the column heading for the property whose values you want to replace, and click **Bulk Replace**.

4. In the **Search for** and **Replace with** boxes, type the search and replace text strings.

Tip: To replace empty properties, leave the **Search for** box blank.

5. To perform a case-sensitive search, select the **Match case** check box.

6. To search for the complete property text, select the **Match entire cell contents** check box.

Tip: To replace all property fields, regardless of the text they contain, type one asterisk in the **Search for** box and select the **Match entire cell contents** check box.

7. Do one or more of the following:

- Click **Replace** to replace the currently selected object and find the next match.
- Click **Next** to select the next matching object.
- Click **Replace All** to replace all matching properties, starting from the first matching property in the list.
- Click **Close** if you are finished replacing.

As the changes to a property are made, the results appear on the tab.

When the bulk replace reaches the last object in the list, a message appears showing the number of replacements.

The Tools Pane

The **Tools** pane contains the **Summary** tab, the **Search** tab, and the **Dependencies** tab.

You can change the **Tools** pane into a separate window. To do this, click the title bar of the pane and drag it in any direction until the outline of pane becomes a thick line. You can now drag the **Tools** window to any location on your desktop. To change the window back to a pane inside the IBM® Cognos® Framework Manager window, drag it to one of the Framework Manager borders until the window outline becomes a thin line.

The Summary Tab

The **Summary** tab shows the language, statistics, and tasks available for the selected object in the **Project Viewer**.

The **Project** section shows the design language and the active language. You can change the active language.

The **Statistics** section shows the number of objects, by class, located in the currently selected object. If the selected object contains a folder, the contents of the folder are included in the number count. Selected objects include projects, namespaces, and folders. The default selected object is the project.

The **Tasks** section shows actions that are applicable to the currently selected object, based on the object class. If you select a folder, actions for the folder are listed. If you select an object in that folder, the list includes actions for both the object and the folder.

For more information about the object classes, see the *c10_location\templates\bmt\CR1Model\BMTModelSpecification.xsd* file.

The Search Tab

When you are working with a large project, it can be difficult to locate the objects that you need to complete a task. Use the **Search** tab to quickly find objects by applying different search criteria, such as the location, the class, a condition, or a property.

If your model contains multiple languages, ensure that you specify the **Active language** value in the **Define Languages** dialog box before searching for objects.

Steps

1. In the **Tools** pane, click the **Search** tab.

Tips:

- If the **Tools** pane is not visible, from the **View** menu, click **Tools**. You can drag the **Tools** pane to the bottom of the Framework Manager window and resize it to have a better view of the search results.
 - To reduce the set of search properties, filter the properties first. On the **Properties** tab, right-click the column header you want to filter on, and click **Set Autofilter**.
2. In the **Search String** box, type the text that you want to find.

You can use uppercase, lowercase, or mixed case strings. The search is not case sensitive. Valid wildcard characters are "*" and "?".

3. Click the **double down arrow** button  to show the search criteria boxes.

4. In the **Condition** list, select a condition to apply to the search string.

The **Condition** box determines how the **Search string** value is matched with text in the model. It contains a list of possible search conditions. If you want to search using wildcard characters, use the **equals** condition.

A regular expression is a complex and powerful method for matching text strings. To search using a regular expression, use the **regular expression** condition. A regular expression search is case sensitive. For example, to find all objects that contain the word "Car" search for the string "\Car", without the quotation marks.

The value for the **Condition** box is saved from session to session.

5. In the **Search in** list, select the part of the model hierarchy that you want to search.

The value for the **Search in** box is saved from session to session.

6. In the **Class** list, select the single class of objects that you want to search.

7. In the **Property** list, select the type of property that you want to search.

The (**All Properties**) property searches all properties. The **Object Name** property restricts the search to the name of each object. The **Text Properties** property searches the set of properties that contain text strings, such as Description or Screen Tip, but not including the object name.

The value for the **Property** box is saved from session to session.

8. Click **Search**.

The results are listed at the bottom of the **Search** tab. Search results can contain multiple values, such as text properties, for a single object. When you click an object that has multiple values, all the values that belong to that object are selected. If your model contains multiple languages, the search results include properties for each language in the model.

After you do one search, the **Subset** check box becomes available. If you select the **Subset** check box, the next search operates on the existing search results. The **Subset** check box is cleared after each search. You can do successive subset searches by selecting the **Subset** check box.

9. To see an object in the **Project Viewer**, click an object in the **Search** tab.

10. To see an object in the diagram, right-click an object in the **Search** tab and click **Locate in Diagram**.

The **Bulk Replace** button on the **Search** tab allows you to do a search and replace operation on the set of objects in the search results. When you click the **Bulk Replace** button, the result set appears in the **Properties** tab and the **Bulk Replace** dialog appears. From there, you can select the property to apply the search and replace operation.

You can also initiate a bulk replace from the **Properties** pane. (p. 34).

The Dependencies Tab

The **Dependencies** tab shows the objects that are dependent on a selected object. For more information, see "[Show Object Dependencies](#)" (p. 289).

Naming Conventions for Objects in a Project

All objects in a project must have a unique identifier. The reference can consist of one or more parts, depending upon the type of object. The parts include

- an object name
- a location in the project hierarchy, as expressed in the default language of the project.

Note: If you want two dimensions or query subjects to have the same name in a project, they must be in different namespaces.

One-part Identifiers

Some objects in a project have a one-part identifier. The one-part identifier must be unique across the entire project, even if the namespace contains other namespaces. These objects have a one-part identifier:

- namespaces
- functions
- shortcuts to namespaces
- shortcuts to folders

Two-part Identifiers

Some objects in a project have a two-part identifier consisting of the name of the containing namespace and the name of the object. The object name must be unique in the containing namespace (p. 25). These objects have a two-part identifier:

- regular dimensions
- measure dimensions
- query subjects
- shortcuts to query subjects

For example, a `go_sales` namespace contains a query subject named `Product`. The `Product` query subject has the following name, where the square brackets and periods are the syntax that Framework Manager uses for object identifiers:

```
[go_sales].[Product]
```

Three-part Identifiers

Some objects in a project have a three-part identifier based on the identifier of the containing query subject. Each name must be unique in the containing query subject. These objects have a three-part identifier:

- hierarchies
- measures
- query items

For example, a `go_sales` namespace contains a query subject named `Product`, and a query item named `Product Code`. The `Product Code` query item has the following name, where the square brackets and periods are the syntax IBM® Cognos® Framework Manager uses for object identifiers:

```
[go_sales].[Product].[Product Code]
```

Four-part Identifiers

Levels in a project have a four-part identifier consisting of the namespace name, the dimension name, the hierarchy name, and the level name.

For example, a `go_data_warehouse` namespace contains a dimension named `Account`. A hierarchy in `Account` is `Balance sheet`, which contains a level named `Account`. The `Account` level has the following name, where the square brackets and periods are the syntax Framework Manager uses for object identifiers:

```
[go_data_warehouse].[Account dimension].[Balance sheet].[Account]
```

Five-part Identifiers

Some objects in a project have a five-part identifier consisting of the namespace name, the dimension name, the hierarchy name, the level name, and the query item name. Five-part identifiers are also used for captions and business keys in member unique names (p. 120).

For example, a `go_data_warehouse` namespace contains a dimension named `Account`. A hierarchy in `Account` is `Balance sheet`, which contains a level named `Account`. The `Account` level contains a query item named `Account name`. The `Account name` query item has the following name, where the square brackets and periods are the syntax Framework Manager uses for object identifiers:

```
[go_data_warehouse].[Account dimension].[Balance sheet].[Account].[Account name]
```

Sample Models

Several sample models are included with IBM® Cognos® Framework Manager for you to explore. The sample models have been created using the guidelines for modeling metadata (p. 319).

In each sample model, the query items have default formatting defined. Names and descriptions were translated into many different languages. By using the `Language_lookup` parameter map, each user automatically sees folder and item names and descriptions in their preferred language.

For more information about sample reports and packages, see the *Report Studio User Guide*.

The Great Outdoors Warehouse Model

This model contains financial information and human resources information for the fictional company, The Great Outdoors. The model accesses a dimensional relational data source.

This sample model is located in `c10_location\webcontent\samples\models\great_outdoors_warehouse`.

The sample model contains these views:

- Database view

Contains the objects that were imported from the data source. These objects are grouped into namespaces for each business area and one named GO for the tables that are common to all business areas.

In addition, model query subjects, shortcuts, and filters are organized into namespaces. The namespaces appear as folders to your users. This organization of information into relevant business categories helps users to locate the information that they require.

All joins and determinants are defined in this view.

A few model query subjects were added for those that required determinants and for lookup tables.

- **Business view**

Contains model query subjects that represent data in terms of business or application needs.

Calculations, including ones for language, are defined in this view.

This view improves model portability and prevents the metadata queries that occur when calculations are made directly into the data source query subjects.

- **Dimensional view**

Contains regular dimensions, measure dimensions, and scope relationships that were created in Framework Manager.

- **Analysis view**

Contains the part of the model that is visible in the studios for the analysis (dimensional) package.

Shortcuts to the regular and measure dimensions in the Dimensional view are grouped in star schemas and placed in the root of the model in folders, one for each business area. They are clearly marked with "(analysis)" at the end of the name. They are not in a separate namespace because that would add another level in the metadata tree in the studios.

- **Query view**

Contains the part of the model that is visible in the studios for the query and reporting package.

Shortcuts to the model query subjects in the Dimensional view are grouped in star schemas and are placed in folders, one for each business area. They are clearly marked with "(query)" at the end of the name in the root of the model.

In addition, there are packages in the sample model: one for analysis and one for query and reporting. You cannot use the query and reporting package in IBM® Cognos® Analysis Studio.

The Great Outdoors Sales Model

This model contains sales analysis information for the fictional company, The Great Outdoors. The model accesses a transactional system.

This sample model is located in *c10_location\webcontent\samples\models\great_outdoors_sales*.

The sample model contains these views:

- **Database view**

Contains the query subjects that were imported from the data source. Because this is a transactional data source, most of the fact tables do not have keys. In some cases, we used multiple-part keys and in other cases, model query subjects were added to calculate the keys and to resolve ambiguous cardinality.

All joins and determinants are defined in this view.

- Business view

Contains model query subjects and reference shortcuts that represent the data in terms of business or application needs. Calculations, including ones for language, are defined in this view.

- Dimensional view

Contains regular dimensions, measure dimensions, and scope relationships. The dimensions are based on the model query subjects in the Business view.

This is also where the query subjects are renamed, if needed.

- Analysis view

Contains the part of the model that is visible in the studios for the analysis (dimensional) package.

Shortcuts to the regular and measure dimensions in the Dimensional view are grouped in star schemas and placed in the root of the model in folders, one for each business area. They are clearly marked with "(analysis)" at the end of the name. They are not in a separate namespace because that would add another level in the metadata tree in the studios.

- Query view

Contains the part of the model that is visible in the studios for the query and reporting package.

Shortcuts to the model query subjects in the Dimensional view are grouped in star schemas and are placed in folders, one for each business area. They are clearly marked with "(query)" at the end of the name in the root of the model.

In addition, there are packages in the sample model: one for analysis and one for query and reporting. You cannot use the query and reporting package in IBM® Cognos® Analysis Studio.

Chapter 3: Model Design Accelerator

Model Design Accelerator is an extension of IBM® Cognos® Framework Manager that simplifies the creation of relational star schema models. It helps the modeler create a single fact table relational star schema that follows proven modeling practices. Novice modelers can build models without extensive experience and training. Experienced modelers could reduce the overall time to build a model. Model Design Accelerator evaluates each design step to identify and help you resolve potential issues.

Before starting to design a model, it is necessary to understand the reporting problem you are trying to solve and what data is available to solve it. (p. 21)

To get started with Model Design Accelerator, do the following:

- Create a project and select a data source (p. 47).
- Explore the user interface of Model Design Accelerator (p. 43).
- Create a star schema model (p. 48).
- Review how to manage your star schema model (p. 50).

The User Interface

Model Design Accelerator has several views and dialog boxes to help you create your model:

- Explorer Tree (p. 43)
- Explorer Diagram (p. 44)
- Model Accelerator (p. 45)
- Query Subject Diagram (p. 45)
- Relationship Editing Mode (p. 45)
- Model Warning (p. 46)
- Options (p. 47)







To access pop-up menus in each view, right-click in the view. The **Options** dialog box is available from all pop-up menus.

Explorer Tree

Use the **Explorer Tree** to view objects that you selected using the Metadata Wizard. The data source objects are shown in a hierarchical view, similar to other file systems.


To see a graphical view of your data source, use the **Explorer Diagram**(p. 44). If the data source contains a large number of objects, it may be easier to locate an object using the **Explorer Tree**.

The **Explorer Tree** uses the following icons to represent objects.

Icon	Object
	Project
	Table in the data source
	Column in the data source The data type property of the column identifies it as a measure.
	Column in the data source The properties of the column identify it as a key.
	Column in the data source The properties of the column do not identify it as either a measure or a key.
	Data source

Explorer Diagram

The **Explorer Diagram** shows a graphical view of your data source metadata. Use the **Explorer Diagram** to explore your metadata and view the relationships between objects.

To access the **Explorer Diagram**, select one or more objects in the **Explorer Tree**. Then, click the **Explorer Diagram** icon  located above the **Explorer Tree**.

In the **Explorer Diagram** window, you can do the following:

- View objects and relationships.
- Change the layout of objects to either star layout or standard tree layout.
- Create a preliminary star schema based on any table containing facts.
- Zoom in or out
- Change the settings for the diagrams.

There are two ways of using the **Explorer Diagram**. From the **Explorer Tree**, you can select a subset of objects that you are interested in and add them to the **Explorer Diagram**. From the **Explorer Tree**, you can also add all the objects to the **Explorer Diagram**. You can then keep the objects you are interested in and remove the rest. Use the **Show Related Tables** and **Remove Tables** menu items to create a view of just the objects you want to explore. The diagram is a read-only view of your data source metadata. When you add or remove tables, you are only modifying the diagram. You are not making changes to the data source.

Use the diagram menu options to explore relationships to other objects. To access diagram menu options, select an object in the **Explorer Diagram** and right-click or use the toolbar. Right-click a table to select the option to view the diagram as a star or tree layout around that table. You can also access some of the diagram menu options using the icons.

You can create a preliminary star schema by selecting any one table in the **Explorer Diagram**. Right-click on it and select **Generate a Star Schema from this Table** to create a full star schema design.

You can dock the **Explorer Diagram** onto the workspace by clicking the docking button. From the docked window, you have the option of splitting your screen horizontally or vertically.

Use the **Options** dialog box to customize the default diagram settings. For example, if your tables contain many columns or use long names, the data in the diagrams may not be fully displayed.

Adjust the maximum table size options on the **General** tab to ensure that all your data is displayed.

Use the **Colors** tab to modify default colors used to identify diagram characteristics.

Model Accelerator Workspace

After you create a project (p. 47), you start work in a Model Design Accelerator session. The **Model Accelerator** is the graphical workspace where you build your star schema.

To build your star schema, design the fact query subject using the measures that you want to include in your reports. Then, build model query subjects to provide context and describe those measures. The lines that join query subjects indicate that relationships exist. Use the **Query Subject Diagram** to view the data source tables that were used to create the selected query subject.

Model Design Accelerator evaluates each design step against modeling proven practices to identify and help you resolve potential issues. If your actions result in a potential issue, the **Model Warning** view appears.

After you complete your star schema, generate a model. You can refine the model in Framework Manager. Create a package and publish the model to IBM® Cognos® Connection to begin authoring reports in the studios.

Query Subject Diagram

Use the **Query Subject Diagram** to show the data source tables that were used to create the star schema. These tables are imported into Framework Manager as part of your generated model. To access the **Query Subject Diagram**, double-click a query subject in the **Model Accelerator** workspace.

In the **Query Subject Diagram** window, you can do the following:

- View the data source tables that were used to create the selected query subject.
- Create, modify, delete or override relationships using the **Relationship Editing Mode**.
- Change the settings for the diagrams using the **Options Dialog**.

Use the **Options** dialog box to customize the default diagram settings. For example, if your tables contain numerous columns or use long names, the data in the diagrams may not be fully displayed. Adjust the maximum table size options in the **Options** dialog box to ensure all your data is displayed. Another option controls the default colors that are used to identify diagram characteristics.

Relationship Editing Mode

You can access **Relationship Editing Mode** in two ways:

- If Model Design Accelerator detects a potential issue, the **Model Warning** window appears. The option **Manually re-draw the joins between tables** activates the functionality of Relationship Editing Mode from the Model Warning view. For more information about the Model Warning view, see "[Model Warning View](#)" (p. 46).
- You can double-click a query subject in the **Model Accelerator** workspace to access the **Query Subject Diagram**. From there, click **Enter Relationship Creation Mode**.

Use **Relationship Editing Mode** to create, modify, delete, or override model joins.

Steps

1. Select two tables that you want to join.

If you cannot create a direct join between two tables, use intermediate tables. To add intermediate tables, drag the tables from the **Explorer Tree** onto the **Query Subject Diagram**. Then, create the required joins by linking the tables through the intermediate tables.


Tip: An intermediate table has a dashed outline.

2. Click the **Create a Model Relationship** icon .

The **Modify the Relationship** dialog box is displayed. A new join line is displayed.

3. Select a column from each table to create the new relationship.
4. Select the appropriate relationship cardinality.
5. Click **OK** twice to return to the **Model Accelerator** workspace.

If your action results in an invalid model, the **Model Warning** view appears.

Alternatively, from **Relationship Editing Mode**, you can select the columns you wish to join and then click the **Create a Model Relationship** icon . The **Modify the Relationship** dialog box appears with the join displayed between the selected columns.

Model Warning View

The **Model Warning** view appears when Model Design Accelerator detects a potential issue. To help you build a valid model, Model Design Accelerator evaluates each of your actions. If Model Design Accelerator detects a potential issue, the **Model Warning** view appears. The **Model Warning** view describes the action that caused the issue and, when applicable, provides options for fixing it.

Some presented options are performed automatically when you select them and click **OK**. Other options require you to make manual changes, either within Model Design Accelerator or in the generated model in Framework Manager.

Because Model Design Accelerator cannot always determine the intent of your actions, you must sometimes resolve an issue by creating manual joins. If it is available, the option **Manually re-draw the joins between tables** activates the functionality of Relationship Editing Mode from the Model Warning view.

When you select **Manually re-draw the joins between tables**, join icons appear at the top left of the Model Warning view. You can create manual joins as required to resolve the identified issue.

For more information, see ["Relationship Editing Mode" \(p. 45\)](#).

If there is a series of dependent modeling options, click **Cancel** to undo your last action. If there are no modeling options, click **Cancel** to undo the last action and return to the **Model Accelerator** workspace.

Change the Settings for Diagrams

You can change one or more settings for the diagram views using the **Options** dialog box. There are three diagram views in Model Design Accelerator. They are the **Model Accelerator** workspace, the **Explorer Diagram**, and the **Query Subject Diagram**.

Steps

1. From any of the diagram views, right-click to access the pop-up menu. Select **Options** and click the **General** tab.
2. Select the type of notation to represent relationships.

You can use Simplified Crowsfoot, Standard Crowsfoot, or Numeric Cardinality notation. By default, Model Design Accelerator uses Simplified Crowsfoot notation which provides a pictorial representation of the relationship.
3. Set a maximum table size for the Explorer and Query Subject diagrams.
4. Set a maximum table size for the Model Accelerator workspace.
5. Select whether you want a text description displayed beside each tool icon.
6. Select whether you want the introductory screen displayed every time you start **Model Design Accelerator**.
7. Click the **Colors** tab to change the colors of diagram objects.

Default colors are assigned to the different characteristics of diagram objects. Change the colors to customize your display.

The **View Background Color** indicates that changes made in the active window will result in changes to the star schema design. This allows you to easily identify windows used for editing and windows used only for viewing.

8. Click **OK**.

Create a Project

In Model Design Accelerator, you work in the context of a project.

Steps

1. From the **Welcome** page of IBM® Cognos® Framework Manager, click **Create a new project using Model Design Accelerator**.

If you are in Framework Manager, you can select **Run Model Design Accelerator** from the **Tools** menu. In this case, a new project is not created. The star schema is associated with the open project and the generated model is added to that project.

2. In the **New Project** page, specify a name and location for the project, and click **OK**.
3. In the **Select Language** page, click the design language for the project.
4. Click **OK** to select the design language.

You cannot change the design language but you can add other languages in Framework Manager. The **Metadata Wizard** appears.

5. To choose your data source, follow the instructions in the **Metadata Wizard**:

- Select a data source and click **Next**.

If the data source you want is not listed, you must first create it. [\(p. 56\)](#)

You can import from only one data source at a time. If you want to import from more than one data source, you must perform multiple imports.

- Select the check boxes for the objects you want to use.

You can select as many or as few objects as you wish. All selected metadata is available for you to work with in your star schema design. Only the objects required for the model will be imported into Framework Manager when you generate the model.

- Click **Continue** to enter Model Design Accelerator.

You may see an introductory screen that gives you an overview of the design steps. Close the introductory screen to begin using Model Design Accelerator.

6. Create your star schema model using the objects you selected from the data source. [\(p. 48\)](#)

7. Do one of the following:

Click **Save** to save the contents of the session and continue working.

Click **Close** to save the contents of the session and enter Framework Manager.

Click **Generate Model** to generate a model based on your star schema design and enter Framework Manager.

You can save and close your design as often as you wish without generating a model. For more information, see ["Create a Star Schema" \(p. 48\)](#)

Create a Star Schema

Creating a star schema using Model Design Accelerator is a three-step process:

1. Build the fact query subject with the measures you want in your reports.
2. Build related query subjects to give context to the measures.
3. Generate the model.

At any time, you can hover pause the pointer mouse over an object to view its properties. The properties show the origin of the object in the data source. In the **Explorer Diagram** view, you are also shown if the item is used in the star schema. You can right click on a join to view the **Relationship Dialog**.

You should save your design as you work. If you exit Model Design Accelerator and then return, you can resume working on the saved star schema.

Note: When you return to Model Design Accelerator, it opens to your last work session. You cannot browse to other projects from Model Design Accelerator.

Build the Fact Query Subject

Use the **Explorer Tree** to explore the tables and columns in your data source. Select the measures that you want to include in your reports and add them to the fact query subject in the Model Accelerator workspace. Measures are numeric columns in a database table or view that can be grouped or aggregated. Examples of measures are Production Cost or Quantity. Model Design Accelerator tracks the required keys and automatically includes them in the model. There is no requirement to add keys. Select only the measures you want report authors to see in your package.

Select measures in the **Explorer Tree** and drag them onto the fact query subject in the **Model Accelerator** workspace. You can also make your selections from the **Explorer Diagram**.

All the measures being added must be from the same table in your data source. If you try to add measures from multiple tables, or add query items that are not measures, the **Model Warning** dialog box will appear. This is because Model Design Accelerator supports only a single fact table star schema model.

Build Related Query Subjects

Use objects from the **Explorer Tree** to build the query subjects in the star schema model. Select columns from the **Explorer Tree** and add them to the appropriate query subjects in the Model Accelerator workspace. You can add or remove query subjects or columns from the workspace as required.

To help you build the query subjects, use the **Explorer Diagram** to examine the metadata for the tables and views that you selected from the data source. You can create query subjects from either the **Explorer View** or the **Explorer Diagram**.

From the **Explorer Diagram**, you can select one query subject or many and view the relationships.

Generate Your Model

When your star schema is complete or ready for testing, click **Generate Model** to build the model and enter IBM® Cognos® Framework Manager.

The **Model Advisor** tests are applied to the resulting model. In the **Verify Model Results** tab, review the issues that are identified. There is a description of each issue and a list of objects that are impacted by the issue. For more information on the Model Advisor, see ["Analyze a Model" \(p. 191\)](#).

The star schema model that you created will usually contain only a subset of query items from the data source. When you generate a model, only the metadata that is required to support your star schema model is imported into Framework Manager.

Each time you generate a model, new namespaces are created in the open Framework Manager project. The **Physical View** contains only the tables that were imported to support the star schema model. The **Business View** contains model query subjects with only the columns you selected when building the query subjects. These are the objects you built in Model Design Accelerator. The **Presentation View** contains a collection of shortcuts referencing the model query subjects in the Business View.

From Framework Manager, you can enhance the model by adding calculations, filters, additional languages, and dimensional structures. However, each time you generate a new model, new namespaces are created and your enhancements are not applied to the new namespaces.

You can return to Model Design Accelerator at any time to edit your star schema. You must publish the model in Framework Manager to use it in the IBM® Cognos® studios.

Managing Your Star Schema

One way to design a star schema is by creating a stand-alone model. To do this, create a project, build a star schema in Model Design Accelerator, generate a model and save the model in the IBM® Cognos® Framework Manager project.

You can save and close your star schema model design as often as you wish without generating a model. After generating a model, you can return to Model Design Accelerator by selecting **Run Model Design Accelerator** from the **Tools** menu. You can do one of the following:

- edit the current star schema, if one exists
- create a new star schema using the same imported data source metadata

The Model Design Accelerator metadata is saved in the mda_metadata.xml file in the project folder. You can move the star schema to another project by copying the mda_metadata.xml file into a different project folder.

Each time you generate a model, namespaces are created in the open Framework Manager project. The **Physical View** contains only the tables that were needed to support the star schema model. The **Business View** contains model query subjects with only the columns you selected when building the query subjects. These are the objects you built in Model Design Accelerator. The **Presentation View** contains a series of shortcuts to the model query subjects you created in the Business View.

For each subsequent model generated, a new set of namespaces is created in the Framework Manager project. A number is appended to each namespace name to distinguish it from the previous version. For example, the first model contains a namespace entitled Physical View. The second model will contain Physical View1.

When your star schema design is complete, you can enhance the resulting model in Framework Manager by adding calculations, filters, additional languages, and dimensional structures. If you change the star schema and regenerate a model, your original design and any enhancements are not overwritten.

Chapter 4: Importing Metadata from Data Sources

Before modeling in IBM® Cognos® Framework Manager, you must import metadata. You can import metadata from a variety of data sources, both relational and dimensional. You can import into a new model or into an existing one. Before importing, it is important that you determine that the data source contains the data and metadata that satisfy your reporting needs.

To import metadata, we recommend that you do the following:

- ❑ Define data sources [\(p. 51\)](#).
- ❑ Set data source security [\(p. 51\)](#).
- ❑ Learn about the types of data source connections [\(p. 52\)](#).
- ❑ Learn how to work with data source connections [\(p. 53\)](#).
- ❑ Create a data source connection [\(p. 56\)](#).
- ❑ Import metadata [\(p. 58\)](#).

Data Sources

Before you can create models and import metadata, you must define data sources. A data source connection supplies the information that IBM® Cognos® BI needs to connect to a database.

Each data source can contain one or more physical connections to databases. The data source connection specifies the parameters needed to connect to the database, such as the location of the database and the timeout duration. A connection can include credential information and signons.

You can secure data sources using IBM Cognos authentication. IBM Cognos authentication respects any security that is also defined within the data source. You can create data source signons to isolate the database logon process from the end users. The signon stores the user ID and password required to access the database. You can also deploy data sources.

For more information about data source connections, see the IBM Cognos *Administration and Security Guide*.

Data Source Security

You can define security for data sources using IBM® Cognos® authentication or data source-specific security. Defining IBM Cognos authentication for a data source does not override any database vendor-specific security.

Depending on the data source, one or more of the following types of IBM Cognos authentication are available:

- no authentication

IBM Cognos BI logs on to the data source without providing any signon credentials.

- IBM Cognos service credentials

IBM Cognos BI logs on to the data source using the logon specified for the IBM Cognos service. Users do not require individual database signons. For production environments, however, individual database signons are generally more appropriate.

- external namespace

IBM Cognos BI logs on to the data source with the same credentials used to authenticate to the specified external authentication namespace. The namespace specified must be active, users must be logged on to it prior to accessing the data source, and the credentials used for the namespace authentication must be relevant for the data source authentication.

All data sources also support data source signons defined for the Everyone group or for individual users, groups, or roles. If the data source requires a data source signon, but you don't have access to a signon for this data source, you will be prompted to log on each time you access the data source.

IBM Cognos BI also respects any security defined for the data source. For example, for IBM Cognos cubes, the security may be set at the cube level. For Microsoft® Analysis Services data sources, the security may be set using cube roles.

Types of Data Source Connections

IBM® Cognos® supports many different types of data sources. The data source connection information may vary for each type of data source you use.


For detailed information about connections for specific data sources, see the IBM Cognos *Administration and Security Guide*. If you require additional information about the parameters to connect to your specific data source, see the vendor documentation for the data source you are using.

In a Framework Manager model, catalogs and schemas are properties that are associated with a data source object. If present, these properties qualify any object that is generated in an SQL statement at run time.

When switching the data source against which a model is mapped, qualification levels are often removed. The Relational Database Management System (RDBMS) applies appropriate name space searches as it prepares the SQL statements it receives. Depending on the vendor, the RDBMS may look in up to two places to resolve a reference to an object. An error results if a reference cannot be resolved.

IBM® Cognos® Framework Manager preserves the names of tables and columns as presented by the RDBMS. These names appear in generated Cognos SQL statements. Cognos SQL uses quotes for the names to preserve case and to avoid conflicts with special characters or keywords.

Native Metadata

IBM® Cognos® supports OLAP data sources as well as relational data sources. The term native metadata refers to objects such as models, packages, and queries that are based on an OLAP data source. A namespace that contains native metadata uses this icon  to indicate that it is different from namespaces containing other types of metadata.

OLAP data sources are metadata rich data sources. Explicit modeling for these data sources is not enabled in Framework Manager and the package is published directly to the portal. For more information, see ["Publish a Package Based on an OLAP Data Source"](#) (p. 268).

Levels are created using the generation names in the labels. If you want to alter the way levels are named, you can do this by changing the dimension build settings in the application that generated the cube. For more information, see the vendor documentation.

Relational data sources require dimensional modeling to enable them to work in IBM® Cognos® Analysis Studio and to work with drill capabilities in the other studios. For more information about dimensional modeling, see ["Dimensions"](#) (p. 114).

If you installed IBM Cognos components on UNIX servers, we recommend that you also locate the file-based data source on a UNIX server. You should then use a UNIX path, such as /servername/cubes/Great Outdoors Company.mdc to access the file. For more information, see [\(p. 56\)](#).

Compound packages contain both OLAP and relational metadata.

Working With Data Source Connections

You can customize data source connections to meet the needs of users.

Multiple Data Source Connections

If you have access to more than one data source connection in a data source, you are prompted to select a data source connection when you open an IBM® Cognos® Framework Manager project. You can use multiple data source connections in a single data source to facilitate the migration from one environment to another and maintain the integrity of a project.

For example, you can use multiple data source connections to work with metadata from a test data source. Create a new project, using the GoSales data source connection. Create and modify the objects you want in the project, and test to ensure that the project is modeled the way you want. After you close the session, and reopen the Framework Manager project, you can select the production data source connection. When you publish the package to the IBM Cognos server, your users choose which data source connection they want to use in their report.

Multiple connections to the same data source must be defined in IBM Cognos Connection. If you want to support multiple connections for each data source, clear the data source catalog and schema names, and create a connection for each database in IBM Cognos Connection.

Note: If you are working with multiple cubes containing unlike metadata, we recommend that you use separate data sources for each cube. To be able to expand an OLAP package in the Studios, the internal name of both cubes must be the same. If you want to run saved reports that use different data source connections, the cube name, as well as the dimension, hierarchy, level and attribute names, must be the same in both cubes. If you use a single data source with a separate connection for each cube, the internal names of all the cubes must be the same.

For more information about data source connections, see the IBM Cognos *Administration and Security Guide*.

Isolation Levels

The isolation level specifies how transactions that modify the database are handled. By default, the default object gateway is used. Not all types of databases support each isolation level. Some database vendors use different names for the isolation levels.

Queries that are executed by reports and analysis are intended to be read-only operations. The queries execute with a unit of work at the data source known as a transaction with either a default or administrator-defined isolation level. Report authors should not assume that queries that execute stored procedures commit any data written by the procedure. In some environments, changes made by a procedure may be committed due to features of the database. A stored procedure that is marked for-write in Framework Manager commits changes but can only be used by Event Studio.

If you need specific queries to run with different isolation levels, you must define different database connections.

For OLAP data sources, including SAP BW, the transaction unit of work is read-only.

The following isolation levels are in increasing order of isolation:

- Read Uncommitted

Changes made by other transactions are immediately available to a transaction.

Database type	Equivalent isolation level
Oracle	Not applicable
DB2®	Uncommitted read
Microsoft® SQL Server	Read uncommitted
Sybase Adaptive Server Enterprise	Read uncommitted
Informix®	Dirty read

- Read Committed

A transaction can access only rows committed by other transactions.

Database type	Equivalent isolation level
Oracle	Read committed
DB2	Cursor stability
Microsoft SQL Server	Read committed
Sybase Adaptive Server Enterprise	Read committed
Informix	Committed read

- Cursor Stability

Other transactions cannot update the row in which a transaction is positioned.

Database type	Equivalent isolation level
Oracle	Not applicable
DB2	Not applicable
Microsoft SQL Server	Not applicable
Sybase Adaptive Server Enterprise	Not applicable
Informix	Cursor stability

- Reproducible Read

Rows selected or updated by a transaction cannot be changed by another transaction until the transaction is complete.

Database type	Equivalent isolation level
Oracle	Not applicable
DB2	Read stability
Microsoft SQL Server	Repeatable read
Sybase Adaptive Server Enterprise	Repeatable read
Informix	Repeatable read

- Phantom Protection

A transaction cannot access rows inserted or deleted since the start of the transaction.

Database type	Equivalent isolation level
Oracle	Not applicable
DB2	Not applicable
Microsoft SQL Server	Not applicable
Sybase Adaptive Server Enterprise	Not applicable

Database type	Equivalent isolation level
Informix	Not applicable

- Serializable

A set of transactions executed concurrently produces the same result as if they were performed sequentially.

Database Type	Equivalent isolation level
Oracle	Serializable
DB2	Repeated read
Microsoft SQL Server	Serializable
Sybase Adaptive Server Enterprise	Serializable
Informix	Not applicable

Create a Data Source Connection

A data source defines the physical connection to a database. A data source connection specifies the parameters needed to connect to a database, such as the location of the database and the timeout duration. These parameters form a connection string for the data source.

You can create data sources in the portal or in Framework Manager. Because they are stored on the server, data sources appear in both places, regardless of where they were created. Existing data source connections can be edited only in the portal.

If you are an administrator, you can set up all required data sources before models are created in Framework Manager so that all connections are available in the Framework Manager Metadata wizard.

Data sources are stored in the Cognos® namespace and must have unique names. For example, you cannot use the same name for a data source and a group.

You can include authentication information for the database in the data source connection by creating a signon. Users need not enter database authentication information each time the connection is used because the authentication information is encrypted and stored on the server. The signon produced when you create a data source is available to the Everyone group. Later, you can modify who can use the signon or create more signons.

Before creating data sources, you must have write permissions to the folder where you want to save the data source and to the Cognos namespace. You must also have execute permissions for the **Data Source Connections** secured feature.

Recommendation - Use Network Paths For File-Based Data Sources

If you have a distributed installation with several servers, we recommend that you use network paths for all file-based data sources rather than local paths. This ensures that the data sources can be accessed by the services that require them, regardless of which server requires the data.

When you create a connection to a file-based data source, such as a PowerCube, you enter a path and file name. To point to the file, use a local path, such as C:\cubes\Great Outdoors Company.mdc, or a network path, such as \\servername\cubes\Great Outdoors Company.mdc.

In a distributed installation, where report servers are running on different computers, using a local path requires that the file and path be valid on each computer where a report server is running. Alternatively, if you use a network path to point to a file, each report server points to the same file on the network without having the file available locally. Also, to ensure that the file is always available, we recommend that you store it in a shared directory that can be accessed on your network.

If you installed IBM Cognos Business Intelligence components on UNIX® operating system servers, we recommend that you also locate the file-based data source on a UNIX server. You should then use a UNIX path, such as /servername/cubes/Great Outdoors Company.mdc to access the file.

If you have installed all components on a single computer, you can use local paths, but you must ensure that the services requesting the data have the appropriate access to the data files on the computer.

For Microsoft® Windows® operating system distributed installations, we recommend that you use UNC paths to shared directories for any file based data source, such as PowerCubes or XML files.

Connections to Specific Data Sources

To create a connection to a data source, you should first review the pertinent information in the IBM Cognos *Administration and Security Guide*. For each of the following data source types, you will find detailed information about the required connection parameters and authentication.

- IBM® DB2®
- IBM® Cognos® Cubes
- Oracle Essbase
- IBM® Infosphere™ Warehouse Cubing Services
- IBM® Informix®
- Microsoft® Analysis Services
- Microsoft® SQL Server
- ODBC Connections
- Oracle
- SAP BW
- Sybase Adaptive Server Enterprise
- IBM Cognos TM1

- XML

Depending on the type of database you are connecting to, you may also need to review information about isolation levels ([p. 54](#)).

Steps

1. Click the namespace, folder, or segment you want to import into, and from the **Actions** menu, click **Run Metadata Wizard**.
2. In the **Select Metadata Source** window, click **Data Sources** and then select **Next**.
3. In the **Select Data Source** window, click **New**. This runs the **New Data Source Wizard** that is also available from Cognos Connection.

From the **New Data Source Wizard**, you can access the IBM Cognos *Administration and Security Guide* from the online help system.

4. In the name and description page, type a unique name for the connection and, optionally, a description and screen tip, and then select **Next**.
5. In the connection page, from the **Type** drop-down list, select the type of data source you want to create.

If your data source is not listed, select **Other type**.

6. If necessary, specify an isolation level for your data source. ([p. 54](#))
7. Enter any parameters that make up the connection string, and specify any other settings, such as a signon or a timeout.

In the IBM Cognos *Administration and Security Guide*, you will find detailed information about the required connection parameters and authentication.

8. Select **Test the connection** and then **Test** to test whether parameters are correct.

If prompted, type a user ID and password or select a signon, and then click **OK**. If you are testing an ODBC connection to a User DSN, you must be logged on as the creator of the DSN for the test to succeed.

In the **Status** column, you can see if the connection was successful. If it was unsuccessful, select **Close**, return to the previous steps and verify your connection parameters.

Tip: You can also test the data source connection from the **Data Sources** folder in the **Project Viewer**. Right-click the data source and click **Test**.

Importing Metadata

You can import metadata into a new project or an existing project. Importing metadata is an operation that can be performed many times to extend the project. You can also export your model to a Common Warehouse Metamodel (CWM) "[Export Metadata](#)" ([p. 291](#)).

IBM® Cognos® Framework Manager can use the metadata and data from external data sources to build a project.

To import metadata, you must indicate which sources you want and where they are located. You can import from only one data source at a time. If you want to import from more than one data source, you must perform multiple imports.

You can import metadata from

- [relational databases](#), such as Oracle, IBM® DB2®, and Microsoft® SQL Server
- [SAP BW data sources](#)
- [IBM Cognos models](#)
- [Architect models and Impromptu catalogs](#)
- [IBM Cognos DecisionStream or Data Manager models](#)
- [IBM metadata sources](#)
- [third party metadata sources](#)
- [XML as a data source](#)

For information about the supported data source types, access one of the IBM Cognos Information Centers at <http://publib.boulder.ibm.com/infocenter/cogic/v1r0m0/index.jsp>.

For information about working with data source connections, see [\(p. 53\)](#).

Duplicate Object Names

When you import metadata, you can select how you want the import to handle duplicate object names. You have the option of not importing the object, or importing and creating a unique name. The advantage of importing everything except these duplicate objects is that you can add new database objects to the project without specifying them individually, and without going through synchronization "[Synchronize Projects](#)" ([p. 301](#)). To import metadata that has the same table names, you must create two namespaces and import each table into a different namespace.

When you import SAP BW metadata, IBM® Cognos® Framework Manager assigns a unique name to each object. Therefore, if you rename an object in the model and then reimport it, Framework Manager recognizes that it already exists. To reimport an object with a different unique name, you can create a new namespace and reimport the object into this namespace.

You can also import objects that have the same name ([p. 76](#)).

Import Metadata from a Relational Database

When you import metadata from a relational database, you can import all the metadata or select particular object types such as tables, columns, views, synonyms, stored procedures, and functions. You can also import system objects from a relational source. System stored procedures are not supported. IBM® Cognos® Framework Manager supports only user-defined stored procedures.

The following table shows the database objects that are mapped to Framework Manager objects.

Database object	Framework Manager object
table	query subject
column	query item
view	query subject
synonym	query subject
procedure	query subject
function	project function

Named sets imported from Microsoft® SQL Server and Microsoft® Analysis Server stored as read-only calculations in the Framework Manager model. The calculation has a flag that identifies it as a named set, and a property that contains the dimension name.

Steps

1. Click the namespace, folder, or segment that you want to import into and, from the **Actions** menu, click **Run Metadata Wizard**.
2. From the Select Metadata Source dialog, you can choose the type of metadata you wish to import.

The **Data Sources** option provides a list of data sources defined in the IBM Cognos software.

- Select the specific relational metadata source to import.
- Select a data source connection and click **Next**.
If the data source connection you want is not listed, you must first create it ([p. 56](#)).
- Select the check boxes for the objects you want to import.
- Specify how the import should handle duplicate object names. Choose either to import and create a unique name, or not to import. If you choose to create a unique name, the imported object appears with a number. For example, you see QuerySubject and QuerySubject1 in your project.
- If you want to import system objects, select the **Show System Objects** check box, and then select the system objects that you want to import.
- Specify the criteria to use to create relationships.
- If you want to convert all cardinalities to 1, clear the **Fact detection enabled** check box. Removing *n* cardinalities disables fact detection and the ability to automatically prevent double-counting. For more information, see "[Modeling 1-n Relationships as 1-1 Relationships](#)" ([p. 355](#)).

3. Click **Import**.

Import statistics including a list of objects that could not be imported and a count of objects that were imported are shown.

4. Click **Finish**.

After importing metadata, you must check the imported metadata for the following areas:

- relationships and cardinality ([p. 339](#))
- determinants ([p. 92](#))
- the **Usage** property for query items ([p. 148](#))
- the **Regular Aggregate** property for query items ([p. 148](#))

For more information about checking the metadata, see "[Verifying Imported Metadata](#)" ([p. 339](#)).

Import Metadata from an IBM Cognos Model

You can import metadata from an existing IBM® Cognos® model.

Note: If you import from another IBM® Cognos® Framework Manager project, expression syntax is not adjusted for each language. For example, you create a Framework Manager project using French as the design language and you use French-specific syntax in calculations and filters. You then create a new project using English as the design language and you import the French project into the new project. Expressions defined in the calculations and filters are not valid. You must manually modify the expression after importing the metadata.

Steps

1. Click the namespace, folder, or segment you want to import into and, from the **Actions** menu, click **Run Metadata Wizard**.
2. Click **IBM Cognos Model** and click **Next**.
3. Locate the model (.cpf file) that you want, click **Open**, and click **Next**.
4. Follow the instructions in the **Import** wizard:
 - Select the check boxes for the objects that you want to import.
 - Specify how the import should handle duplicate object names.
Choose either to import and create a unique name, or not to import. If you choose to create a unique name, the imported object appears with a number. For example, you see Query-Subject and QuerySubject1 in your project.
5. Click **Next** and click **Finish**.

Import Metadata from an Architect Model or an Impromptu Catalog

To import metadata from an IBM Cognos Architect® model or an Impromptu catalog, you must first convert it to XML files. Because of differences between IBM Cognos Series 7 and IBM Cognos

BI, after you import the migrated metadata in IBM® Cognos® Framework Manager, additional work is required to test and refine the metadata.

For information about the migration of Series 7, see the IBM Cognos Migration Assistant *User Guide* on one of the IBM Cognos Information Centers at <http://publib.boulder.ibm.com/infocenter/cogic/v1r0m0/index.jsp>.

Steps

1. Ensure that you exported the Architect model or Impromptu catalog.
2. Click the namespace, folder, or segment you want to import into and, from the **Actions** menu, click **Run Metadata Wizard**.
3. Click either **IBM Cognos Architect (.xml)** or **IBM Cognos Impromptu (.xml)** and click **Next**.
4. Locate the Architect or Impromptu XML file that contains the metadata to import.
A message in the **XML Preview** window confirms that you chose a valid XML file.
5. Click **Open**.
6. Select the namespace containing your Series 7 security information.
7. Click **Import**.
A list of created objects appears.
8. If you want to verify the imported metadata, click the **Verify after import** check box.
9. Click **Finish**.

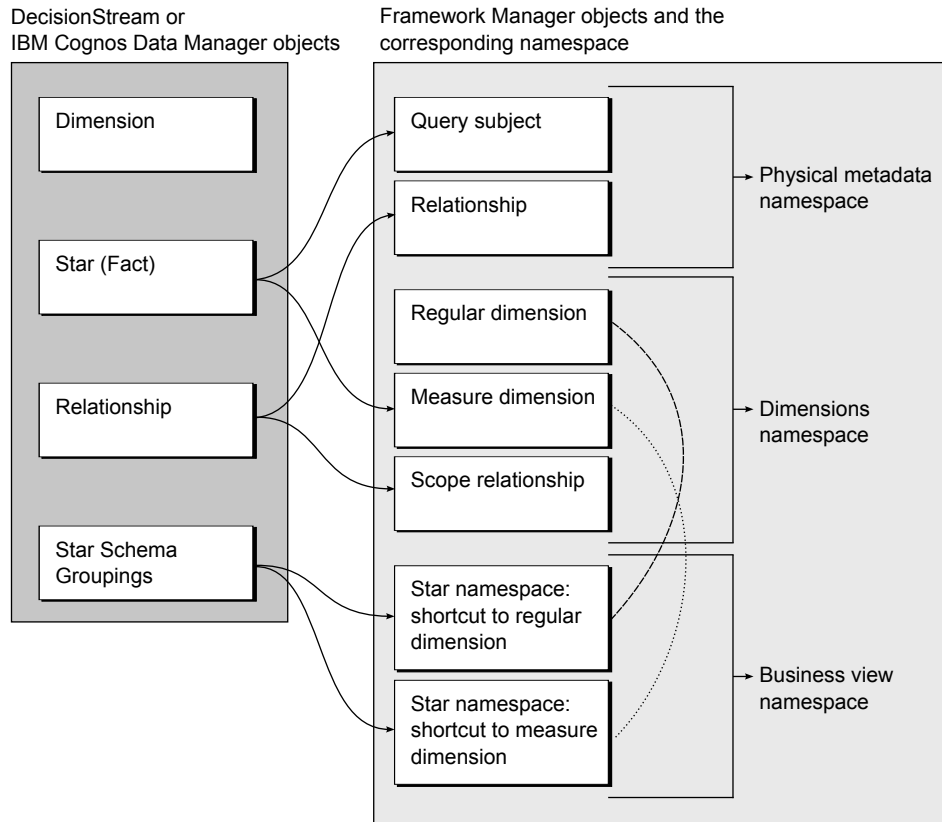
Import Metadata from IBM Cognos DecisionStream or IBM Cognos Data Manager

You can use IBM® Cognos® Framework Manager to import metadata from an XML file created by IBM Cognos DecisionStream or IBM Cognos Data Manager. You can import

- a physical layer residing in the Physical Metadata namespace
This layer contains data source query subjects representing the imported tables. The physical layer contains query subjects and physical relationships between query subjects. These physical relationships are inferred from the relationships defined in the import file. Imported tables become Framework Manager query subjects, and surrogate keys become Framework Manager determinants.
- a dimensional layer residing in the Dimensions namespace
This layer contains regular dimensions, measure dimensions, and scope relationships. The regular dimensions may be conformed or non-conformed. The measure dimension objects correspond to the imported stars. The scope relationships are inferred from the relationships defined in the import file.
- a logical layer residing in the Business View namespace

This layer contains shortcuts to the regular and measure dimensions in the Dimension namespace. The shortcuts are organized as star schema groupings, which are namespaces with the same name as the stars from the import file.

The following diagram shows how objects from DecisionStream and Data Manager are mapped to Framework Manager objects.



Facts

A star maps to a Framework Manager query subject in the Physical Metadata namespace or as a measure dimension in the Dimensions namespace. The following fact attributes are included in the model.

Attribute name	Framework Manager mapping
Table name	Name of the database query subject on which the model query representing the fact is based
Short name	Query subject description
Business name	Custom property
Description	Query subject description
Column name	Query item name

Attribute name	Framework Manager mapping
Column type	Query item data type
Column length	Query item size
Column short name	Custom property
Column business name	Custom property
Column description	Query item description
Column type	Query item usage
Table keys	Determinants in the Physical Metadata namespace

Connections

A connection maps to a Framework Manager data source.

Note: A data source connection is not automatically created in IBM® Cognos® Connection. You must manually create the connection in IBM Cognos Connection as explained in the import procedure steps (p. 62).

The following data source attributes are included in the model.

Attribute name	Framework Manager mapping
Connection short name	Custom property
Connection business name	Data source name
Connection description	Data source description
Connectivity	type.interface property
Connection string	Custom property

Dimension Builds

A dimension build maps to Framework Manager as a top-level namespace.

Hierarchies

A dimension containing hierarchies, levels, and columns maps to a Framework Manager regular dimension containing hierarchies, levels, and query items.

Conformed Stars

Conformed stars map to a Framework Manager namespace that resides in the Business View namespace. It contains shortcuts referencing the dimensions.

The following conformed star attributes are included in the model.

Attribute name	Framework Manager mapping
Star short name	The name of the namespace representing the star
Star business name	The name of the measure dimension representing the fact
Star description	The description of the measure dimension representing the fact
Facts	Shortcuts to a measure dimension
Dimensions	Shortcuts to regular dimensions
Hierarchies	Hierarchies in the regular dimension representing the DecisionStream dimensions

Model Properties

The export file contains the following model properties.

Attribute name	Framework Manager mapping
Schema version	Not mapped
Catalog version	Custom property
Model short name	The name of the namespace representing the top-level model object
Model business name	Custom property
Model description	The description of the namespace representing the top-level model object

Steps

1. Click the namespace, folder, or segment that you want to import into and, from the **Actions** menu, click **Run Metadata Wizard**.
2. Click **IBM Cognos DecisionStream (.xml)** or **IBM Cognos Data Manager (.xml)** and click **Next**.

3. Locate the XML file that contains the metadata to import.
A message in the **XML Preview** window confirms that you chose a valid XML file.
4. Click **Open** and then click **Import**.
A list of created objects appears.
5. If you want to verify the imported metadata, click the **Verify after import** check box.
6. Click **Finish**.
7. Create the data source connection in IBM Cognos Connection. The name you define in IBM Cognos Connection must be the same as the data source name shown in the **Data Sources** folder in Framework Manager.
For information about how to create a data source connection, see [\(p. 56\)](#).

Import from IBM Metadata Sources

You can use IBM® Cognos® Framework Manager to import metadata from IBM data sources such as IBM InfoSphere DataStage. Similar to third party data sources, metadata is extracted using the Meta Integration® Model Bridge (MIMB). For more information, see "[Import Metadata From Third Party Metadata Sources](#)" (p. 69).

IBM Metadata Source Import Options

The options that you see are based on the selected data source. Not all options apply to Framework Manager.

Framework Manager Options

The IBM® Cognos® Framework Manager options available are the same regardless of the type of metadata source that you select. This table shows the options used to create objects in Framework Manager.

Framework Manager options	Description
Logical/Physical representation	<p>Specifies how logical and physical objects are represented.</p> <p>Integrated represents the logical and physical objects as one integrated object.</p> <p>Separated represents the logical and physical objects as two related objects. This is the default.</p> <p>Separated (verbose) represents the logical and physical objects as two distinct objects.</p>
Dimensional/Logical representation	<p>Integrated represents the dimensional object as one integrated object. This is the default.</p> <p>Separated represents the dimensional object as one dimensional and one logical object.</p>

Framework Manager options	Description
Diagram representation	<p>Specifies how diagrams are represented in the project.</p> <p>As Namespaces indicates that diagrams are represented as namespaces.</p> <p>As Packages indicates that diagrams are represented as packages.</p> <p>Both indicates that diagrams are represented as both namespaces and packages. This is the default.</p> <p>None indicates that diagrams are not represented in the project.</p>
Namespace hierarchy	<p>Specifies where the hierarchy of packages is kept.</p> <p>All indicates the hierarchy of packages is kept in all the namespaces. This is the default.</p> <p>None indicates that the hierarchy of packages is not retained.</p> <p>Dimensional level only indicates the hierarchy of packages is kept in the dimensional namespace only</p> <p>Logical level only indicates the hierarchy of packages is kept in the logical namespace only.</p> <p>Physical level only indicates the hierarchy of packages is kept in the physical namespace only.</p>
Logical only classes	<p>Specifies whether the logical only classes are represented.</p> <p>Ignore indicates that logical only classes are ignored.</p> <p>Show indicates that logical only classes are shown along with all their attributes. This is the default.</p>
Logical only attributes	<p>Specify whether the logical only attributes are represented. This option does not affect logical only attributes in a logical only class. See the Logical Only Classes option.)</p> <p>Drop indicates that logical information of a class is not shown if it contains a logical only attribute.</p> <p>Ignore indicates logical only attributes are ignored.</p> <p>Show indicates that logical only attributes are shown. This is the default.</p>

Framework Manager options	Description
Unreferenced dimension attributes	<p>Specifies how unreferenced dimension attributes are represented in the project. An unreferenced dimension attribute is one that does not participate in a level and is not referenced by any other dimension attribute.</p> <p>Ignore All indicates that all unreferenced dimension attributes are ignored.</p> <p>Ignore Join indicates that only unreferenced dimension attributes participating in a join are ignored.</p> <p>Show All indicates that all unreferenced dimension attributes are shown. This is the default.</p> <p>Show Join indicates that only unreferenced dimension attributes participating in a join are shown</p>
Dimensions without dimensional information	<p>Specifies how to represent dimensions that do not contain any dimensional information.</p> <p>Dimension creates a dimension and a default hierarchy and level. This is the default.</p> <p>Model Query creates a modelQuery.</p>
Create cubes	<p>Specifies whether the import can create cubes. A cube is represented as a namespace containing all the information necessary to build the cube.</p>
Populate screentip	<p>Specifies whether the import uses the description field as a screenTip.</p> <p>True allows the content of the description field to be seen as a screentip in IBM Cognos Report Studio, Query Studio or IBM Cognos Business Insight Advanced.</p>
Name of the namespace containing the dimensional information	<p>Specifies the namespace that contains the dimensional information. The default is "Dimensional Model".</p>
Name of the namespace containing the logical information	<p>Specifies the namespace that contains the logical information. The default is "Logical Model".</p>
Name of the namespace containing the physical information	<p>Specifies the namespace that contains the physical information. The default is "Physical Model".</p>
Name of the namespace containing the subject areas	<p>Specifies the namespace that contains the subject areas. The default is "Subject Area".</p>

Framework Manager options	Description
Consistency Check	<p>Specifies the consistency check level.</p> <p>Basic is the recommended consistency check level.</p> <p>Extensive performs a more thorough validation of the model.</p> <p>None indicates that no validation is performed.</p>

Import Metadata From Third Party Metadata Sources

You can use IBM® Cognos® Framework Manager to import metadata from other sources, including relational. Metadata is imported using a metadata bridge.

You can import both relational and other metadata into the same model. We recommend that you start with a new Framework Manager model and import the other metadata before the relational metadata. This avoids conflicts if you import objects that have the same name.

When you import other metadata, data sources are created based on information provided through the import wizard. If at least one physical object in the other source references a database schema or catalog or both, one Framework Manager data source is created with its catalog or schema properties set to the names of the catalog or schema defined in the metadata. A generic data source is created for those physical objects that do not reference a catalog or schema.

If you want to access metadata from another data source, you must perform a physical model import.

Not all data sources contain metadata that is appropriate for business intelligence reporting and not all concepts map to Framework Manager. The metadata import is tailored to Framework Manager and only compatible metadata will be imported.

All metadata bridges deliver a physical layer that provides the basis for further modeling. The richness of the resulting Framework Manager model is directly related to the richness of the metadata source.

Multiple Databases

Other metadata sources can be based on multiple databases. The best way to import these multiple data sources into Framework Manager is to perform multiple imports. For each import, you select the items that correspond to that specific data source.

For example, the first time that you import from another metadata source, you select datasource1 and all the items that correspond to that data source. The next time, you select datasource2 and the items that correspond to that data source. You continue to import until you have imported all the data sources and their corresponding items.

Third Party Specific Import Options

Metadata is extracted from the third party data sources by the Meta Integration® Model Bridge (MIMB). Not all options apply to Framework Manager. For information about supported tools and object mappings, see the Meta Integration Web site.

The options that you see during import are based on the selected data source.

Framework Manager Options

The IBM® Cognos® Framework Manager options available are the same regardless of the type of metadata source that you select. This table shows the options used to create objects in Framework Manager.

Framework Manager options	Description
Logical/Physical representation	<p>Specifies how logical and physical objects are represented.</p> <p>Integrated represents the logical and physical objects as one integrated object.</p> <p>Separated represents the logical and physical objects as two related objects. This is the default.</p> <p>Separated (verbose) represents the logical and physical objects as two distinct objects.</p>
Dimensional/Logical representation	<p>Integrated represents the dimensional object as one integrated object. This is the default.</p> <p>Separated represents the dimensional object as one dimensional and one logical object.</p>
Diagram representation	<p>Specifies how diagrams are represented in the project.</p> <p>As Namespaces indicates that diagrams are represented as namespaces.</p> <p>As Packages indicates that diagrams are represented as packages.</p> <p>Both indicates that diagrams are represented as both namespaces and packages. This is the default.</p> <p>None indicates that diagrams are not represented in the project.</p>

Framework Manager options	Description
Namespace hierarchy	<p>Specifies where the hierarchy of packages is kept.</p> <p>All indicates the hierarchy of packages is kept in all the namespaces. This is the default.</p> <p>None indicates that the hierarchy of packages is not retained.</p> <p>Dimensional level only indicates the hierarchy of packages is kept in the dimensional namespace only</p> <p>Logical level only indicates the hierarchy of packages is kept in the logical namespace only.</p> <p>Physical level only indicates the hierarchy of packages is kept in the physical namespace only.</p>
Logical only classes	<p>Specifies whether the logical only classes are represented.</p> <p>Ignore indicates that logical only classes are ignored.</p> <p>Show indicates that logical only classes are shown along with all their attributes. This is the default.</p>
Logical only attributes	<p>Specify whether the logical only attributes are represented. This option does not affect logical only attributes in a logical only class. See the Logical Only Classes option.)</p> <p>Drop indicates that logical information of a class is not shown if it contains a logical only attribute.</p> <p>Ignore indicates logical only attributes are ignored.</p> <p>Show indicates that logical only attributes are shown. This is the default.</p>
Unreferenced dimension attributes	<p>Specifies how unreferenced dimension attributes are represented in the project. An unreferenced dimension attribute is one that does not participate in a level and is not referenced by any other dimension attribute.</p> <p>Ignore All indicates that all unreferenced dimension attributes are ignored.</p> <p>Ignore Join indicates that only unreferenced dimension attributes participating in a join are ignored.</p> <p>Show All indicates that all unreferenced dimension attributes are shown. This is the default.</p> <p>Show Join indicates that only unreferenced dimension attributes participating in a join are shown</p>

Framework Manager options	Description
Dimensions without dimensional information	<p>Specifies how to represent dimensions that do not contain any dimensional information.</p> <p>Dimension creates a dimension and a default hierarchy and level. This is the default.</p> <p>Model Query creates a modelQuery.</p>
Create cubes	Specifies whether the import can create cubes. A cube is represented as a namespace containing all the information necessary to build the cube.
Populate screentip	<p>Specifies whether the import uses the description field as a screenTip.</p> <p>True allows the content of the description field to be seen as a screentip in IBM Cognos Report Studio, Query Studio or IBM Cognos Business Insight Advanced.</p>
Name of the namespace containing the dimensional information	Specifies the namespace that contains the dimensional information. The default is "Dimensional Model".
Name of the namespace containing the logical information	Specifies the namespace that contains the logical information. The default is "Logical Model".
Name of the namespace containing the physical information	Specifies the namespace that contains the physical information. The default is "Physical Model".
Name of the namespace containing the subject areas	Specifies the namespace that contains the subject areas. The default is "Subject Area".
Consistency Check	<p>Specifies the consistency check level.</p> <p>Basic is the recommended consistency check level.</p> <p>Extensive performs a more thorough validation of the model.</p> <p>None indicates that no validation is performed.</p>

Before you can import metadata, there must be a connection to the data source ([p. 56](#)).

Steps to Import from Third Party Metadata Sources

1. Click the namespace, folder, or segment you want to import into, and from the **Actions** menu, click **Run Metadata Wizard**.
2. Select **Third Party Metadata Sources** and click **Next**.

3. Click the metadata type to import.
4. In the **Third Party Specific Import Options** dialog box, use the **File** option to identify the file that contains the metadata to import.
Click the other options that you want. The options that you see are based on the selected data source.
Note: We recommend that you use the default options. These default options optimize the metadata import. If you change the options, you may see unexpected results. To revert back to the default options, click **Use Defaults**.
5. Click **Next**.
6. In the **Framework Manager Specific Import Options** dialog box, click the options that you want and click **Next**.
7. Follow the instructions in the **Metadata Wizard**:
 - Select a data source connection and click **Next**.
If the data source connection you want is not listed, you must first create it ([p. 56](#)).
 - Select the check boxes for the objects you want to import.
 - Specify how the import should handle duplicate object names. Choose either to import and create a unique name, or not to import. If you choose to create a unique name, the imported object appears with a number. For example, you see QuerySubject and QuerySubject1 in your project.
 - Specify the criteria to use to create relationships and click **Import**.
For more information, see ["Relationships"](#) ([p. 78](#)).

Import statistics including a list of objects that could not be imported and a count of objects that were imported are shown.
8. Click **Finish**.

Modeling After the Import

The source metadata has a logical structure that is compatible with IBM Cognos BI. During the import, this structure is preserved for higher fidelity with the source model.

After the import, only the physical metadata is available.

Do the following:

- ☐ analyze and review the model, see ["Analyze a Model"](#) ([p. 191](#))
- ☐ set the determinants
- ☐ set the **Usage** property

Some bridges require facts to be set manually because this is not available from the metadata source.

- ☐ verify the relationships and cardinality

- ❑ resolve ambiguous relationships, such as multiple valid relationships, reflexive relationships, and recursive relationships

Troubleshooting Metadata from Other Sources

This section describes issues that you can encounter when working with metadata that you have imported into IBM® Cognos® Framework Manager.

Cannot Test a Query Subject from Another Source

You cannot test a query subject that you have imported from another data source.

For example, you test a query subject and get this error message:

QE-DEF-0177 An error occurred while performing operation 'sqlPrepareWithOptions' status='-201'.

UDA-SQL-0196 The table or view "GOSALES1.COGNOS.COM.GOSALES.CONVERSION_RATE" was not found in the dictionary.

To resolve this problem, ensure that the following conditions exist:

- The Framework Manager data source object created by the import, and referenced by the query subject, has an identically named corresponding data source in the IBM Cognos Business Intelligence content store.
- The corresponding content store data source is valid and the connection information is correct.
- The optional schema or catalog properties of the data source object that the import created and the query subject references are correct. For the databases accepting case-sensitive identifiers, ensure that the case is also correct.
- The database object (table or view) represented by the query subject exists in the database with the identical name, and is accessible within the current connection.
- The **Query Type** is valid. An invalid **Query Type** can exist if you import a query subject from SAP BW and change the value of the **Query Type** property for the data source. By default the value of **Query Type** is **multidimensional**. Do not change it.

Relationships Involving Table Views Are Not Imported from an Oracle Designer File

Primary key and foreign key relationships involving at least one table view are not imported from an Oracle Designer file into Framework Manager.

Create the primary key and foreign key relationships manually.

Some Expressions Imported from Other Metadata Sources Are Not Valid

Support for expression parsing was improved in the MIMB in IBM® Cognos® BI. Functions that are equivalent between other metadata sources and IBM Cognos BI are parsed. Exceptions to this include

- functions with no mapping between IBM Cognos BI and the other metadata source
- expressions that use a specialized syntax

Solution

If you have functions with no mapping between IBM Cognos BI and the other metadata source, after importing expressions from other metadata sources, edit these expressions manually in Framework Manager to conform to the equivalent Framework Manager syntax.

If you have expressions that use a specialized syntax, do the following:

1. In Framework Manager, identify all imported query items that represent embedded calculations using references such as \$\$1, \$\$2, and so on.
2. In the **Properties** pane, find the information provided in the **patternMapping** property for each query item.

The **patternMapping** property indicates the mapping between the parameters in the imported calculation and the actual object references in the original model.

3. Double-click a broken query subject.
4. Double-click the embedded calculation corresponding to the query item identified in step 1.
5. If the calculation was assigned a default name ("Calculation..."), replace it with the actual query item name.
6. Replace the imported parameters with the actual Framework Manager object references that these parameters represent.
7. Repeat these steps for each broken query subject.

Import Metadata Using XML as a Data Source

You can import an XML file as a tabular data source in IBM® Cognos® Framework Manager. You can import it locally or from a remote site through a valid URL. In Framework Manager the XML file is used to model metadata and create a package.

The XML file is validated and parsed at run time, when the query is processed by either Report Studio, Query Studio or IBM Cognos Business Insight Advanced. If you add the **VALIDATE=ON** option to the connection string, Framework Manager partially validates the XML file in the `<columnList>` tag that describes the metadata. For information about supported data types, see ["XML Data Types" \(p. 611\)](#).

You must use the `xmldata.xsd` schema to validate the XML file. The schema is located in the `\c10\bin` folder. It is not necessary to specify the location of the schema in the XML file itself.

To use XML as a data source, ensure that

- you do not use Native SQL to access data in an XML file
- you do not access Binary Large Objects (BLOB)
- you use only `sqlColumns()` and `sqlTables()` metadata calls
Other calls return an unsupported function error.
- the XML file is well-formed and valid

Before you can import metadata, there must be a connection to the data source ([p. 56](#)). If the XML data source is on another computer, you must use an account that has permissions to access the data source.

After you create a connection to an XML data source, the data source appears in the list of data sources.

To use XML as a data source, you must understand XML, schemas, and other XML-related technology.

Steps

1. Click the namespace, folder, or segment that you want to import into, and from the **Actions** menu, click **Run Metadata Wizard**.
2. Click the XML data source that you want to import, and click **Next**.

Import Objects with the Same Name

Namespaces are containers like folders. Objects in an IBM® Cognos® Framework Manager project must be uniquely identifiable. If you have two objects that have the same name, they must reside in two separate namespaces.

For example, you have a database that contains financial data. One set of tables represents Forecast and Actual information. Both the Forecast and Actual information have tables named Accounts Payable and Accounts Receivable. To import these tables into Framework Manager and use the same table names in the project, you must create two namespaces. You can name one namespace Forecast, and the other namespace Actual.

Steps

1. Click the model or root namespace and from the **Actions** menu and click **Create, Namespace**.
2. Right-click the namespace, click **Rename**, and type a descriptive name.

Chapter 5: Modeling Relational Metadata

Note: Information on SAP BW metadata is in ["Working with SAP BW Metadata" \(p. 197\)](#).

Note: For information about modeling for use with Dynamic Query Mode, see the *Dynamic Query Guide*.

After importing metadata, you must ensure that it is set up to meet your users' reporting requirements, and provide any additional information that they require. Enhancements you make in IBM® Cognos® Framework Manager do not affect the original data source.

Tip: To verify that the model meets the reporting requirements, you can select objects that will appear in a report and test them. The test results show you the report that your users will see as well as the SQL and messages from the IBM Cognos software, if any. Or you can publish a package at any time and then use the package to create reports.

You can check the project at any time to ensure that the references between the objects it contains are valid [\(p. 251\)](#).

We recommend that you create at least two views: the import view and the business view. Using two, or more, views makes it easier to remap items to a new data source [\(p. 290\)](#).

The import view contains the metadata you imported from the data source. To ensure that the metadata is set up correctly in the import view, do the following:

- ❑ Ensure that the relationships reflect the reporting requirements [\(p. 78\)](#).
- ❑ Optimize and customize the data retrieved by query subjects [\(p. 85\)](#).
- ❑ Optimize and customize the data retrieved by dimensions [\(p. 114\)](#). You may want to store dimensions in a separate dimensional view.
- ❑ Handle support for multilingual metadata [\(p. 131\)](#).
- ❑ Control how data is used and formatted by checking query item properties [\(p. 138\)](#).

The business view provides a layer to the information in the source data so that it is easier for your users to build reports. To enhance the metadata in the business view, do the following:

- ❑ Add business rules, such as calculations and filters, that define the information users can retrieve [\(p. 155\)](#).
- ❑ Organize the model by creating separate views for each user group that reflect the business concepts familiar to your users [\(p. 179\)](#).
- ❑ Create a durable model that can withstand later changes to query item names with no impact on existing reports, and report authors [\(p. 189\)](#).

You can analyze the metadata to ensure that the model is following current modeling guidelines by using the **Model Advisor** [\(p. 191\)](#).

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 both directions to fully understand the relationship.

The different types of relationships are

- one-to-one
One-to-one relationships occur when one instance of data in a query subject relates to exactly one instance of another. For example, each student has one student number.
- one-to-many or zero-to-many
One-to-many or zero-to-many relationships occur when one instance of data in a query subject relates to many instances of another. For example, each teacher has many students.
- many-to-many
Many-to-many relationships occur when many instances of data in a query subject relate to many instances of another. For example, many students have many teachers.

When importing metadata, IBM® Cognos® Framework Manager can create relationships between objects in the model based on the primary and foreign keys in the data source. 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 may 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.

Framework Manager stores relationships in the nearest common parent of the objects that participate in the relationship. The parent can be either a folder or a namespace. If you move one of the participating objects outside the common parent, the relationship moves to the next namespace that is common to both ends of the relationship. If you move a relationship to a different folder or namespace, the participating objects also move to the same folder or namespace.

Tip: Use the **Search** tab (**Tools** pane) to find an object of class Relationship whose name matches a specified pattern. For example, if you search for a relationship whose name contains Order Header, Framework Manager finds all relationships that have Order Header as one end. If you renamed a relationship, a search of this type may not find it.

Cardinality

Relationships exist between two query subjects. 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; this expression usually refers to the primary and foreign keys of the underlying tables.

IBM Cognos software 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

A query that uses multiple facts from different underlying tables is split into separate queries for each underlying fact table. Each single fact query refers to its respective fact table as well as to the dimensional tables related to that fact table. Another query is used to merge these individual queries into one result set. This latter operation is generally referred to as a stitched query. You know that you have a stitched query when you see `coalesce` and a full outer join.

A stitched query also allows IBM Cognos software to properly relate data at different levels of granularity (p. 326).

You must ensure that all relationships and cardinality correctly reflect your users' reporting requirements.

For more information, see ["Cardinality in Generated Queries "](#) (p. 320) and ["Cardinality in the Context of a Query"](#) (p. 321).

Detecting Cardinality from the Data Source

When importing from a relational data source, cardinality is detected based on a set of rules that you specify. The available options are

- use primary and foreign keys
- use matching query item names that represent uniquely indexed columns
- use matching query item names

The most common situation is to use primary and foreign keys as well as matching query items that represent uniquely indexed columns. The information is used to set some properties of query items as well as to generate relationships.

To view the index and key information that was imported, right-click a query subject and click **Edit Definition**. For a query subject, you can change the information in the **Determinants** tab.

Optional relationships, full outer joins, and many-to-many relationships can be imported from your data source. IBM® Cognos® Framework Manager will run them as queries.

Note: All regular dimensions begin as query subjects. If you converted a query subject to a regular dimension, note that determinant information for the query subject is leveraged as a starting point to define the levels of a single hierarchy. We recommend that you review the levels and keys created in the hierarchy of the dimension.

Notation

By default, Framework Manager uses Merise notation. Merise notation marks each end of the relationship with the minimum and maximum cardinality of that end. You can also use Crowsfoot notation, which provides a pictorial representation of the relationship. For information about how to change the notation, see ["Change the Settings for Diagrams"](#) (p. 33).

When you interpret cardinality, you must consider the notation that appears at both ends of the relationship.

Possible end labels are

- 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 this relationship:

- an inner join (1)

An inner join shows all matching rows from both objects.

- an outer join (0)

An outer join 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 and only what matches from the other side.

Your users see a different report depending on whether you use an inner or outer join. For example, your users want a report that lists salespeople and orders. If you use an outer join to connect salespeople and orders, the report shows all salespeople, regardless of whether they have any orders. If you use an inner join, the report shows only the 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 the items match but the relationship in the model has a minimum cardinality of 0, an outer join is always used, although the results are the same with an inner join. For example, the underlying table for one object contains a mandatory (non-NULLable) foreign key for the other. Ensure that the data and cardinalities match.

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

Cardinality in Generated Queries

IBM Cognos software supports both minimum-maximum cardinality and optional cardinality.

In 0:1, 0 is the minimum cardinality, 1 is the maximum cardinality.

In 1:n, 1 is the minimum cardinality, n is the maximum cardinality.

A relationship with cardinality specified as 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 may be specified as 1:1 to 0:n. This indicates that reports will show the requested customer information even though there may not be any sales data present.

Therefore a 1 to n relationship can also be specified as:

- 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 cardinality. For example, Sales Staff (1:1) is joined to Orders (0:n).

Relationship impact:	Each Order has one and only one Sales Staff. Each Sales Staff has zero or more Order (outer join).
----------------------	---

It is important to ensure that the cardinality is correctly captured in the model because it determines the detection of fact query subjects and it is used to avoid double-counting factual data.

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

- Cardinality is applied in the context of a query.
- 1 to n cardinality implies fact data on the n side and implies 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.

Sparse Data

When modeling for analysis or reporting, it is important to consider the nature of the business questions versus the nature of the data source.

A common scenario is that a relationship between a dimension and a fact table in a star schema is optional. This means that not every dimensional member is mandatory in the fact table. OLAP engines compensate for this by inserting an appropriate value when creating the OLAP structure for any dimensional intersection points that do not have data.

For example, an Analysis Studio user wants to create this report:

Country	2005	2006
Canada		1,000,000
Mexico	500,000	750,000
United States	1,000,000	1,250,000

When modeling, it is common to override optional relationships between dimensions and facts for improved performance. However, when performing analysis or reporting on sparse data where you require information about dimensional members that have no facts, outer joins must be enabled to ensure that data is returned for valid dimensional intersection points.

To enable outer joins, we recommend that you do the following:

- Check with your database administrator to ensure that the data source can support full outer joins.
- Import metadata with outer joins enabled.

Modify a Relationship

After you import data (p. 51) or create a relationship (p. 115) in IBM® Cognos® Framework Manager, you can rename the relationship and redefine cardinality.

You can create custom relationship expressions by selecting an operator from the list or by manually changing the expression in the expression editor.

You can also create a complex expression for the relationship (p. 82).

You can view the relationships that already exist for an object by selecting the object and clicking **Launch Context Explorer** from the **Tools** menu.

Steps

1. Click a relationship and, from the **Actions** menu, click **Edit Definition**.
2. To modify existing elements, on the **Relationship Expression** tab, select the query items, cardinalities, and operator you want.

The query items must have the same data type.
3. To create an additional join, on the **Relationship Expression** tab, click **New Link**, and define the new relationship.
4. To test the relationship, on the **Relationship SQL** tab, identify the number of rows you want returned and click **Test**.
5. Click **OK**.


If your metadata is from an OLAP data source, click **Close**.

Create a Complex Expression for a Relationship

You can create complex expressions for relationships by using functions, parameters, and objects from the model.

Steps

1. Click a relationship and, from the **Actions** menu, click **Edit Definition**.
2. On the **Relationship Expression** tab, click the ellipses (...) button next to the **Expression** box.
3. Define the expression.

If you insert session parameters (p. 165) or prompts (p. 149) and you want to specify the values that they represent when you test the expression, click the **options** button .
4. Click **OK**.

Create a Relationship

You create a relationship to join logically related objects that your users want to combine in a single report. This is useful for relationships between objects that were not selected during metadata import, were not joined in the data source, or are from multiple sources.

You can directly create a relationship between the query items.

You can also create a complex expression for the relationship (p. 82).

You can also use IBM® Cognos® Framework Manager to automatically generate relationships (p. 84) between objects based on selected criteria.

You can view the relationships that already exist for an object by selecting the object and clicking **Launch Context Explorer** from the **Tools** menu.

Steps

1. Ctrl+click one or two dimensions, query subjects, or query items.
2. From the **Actions** menu, click **Create, Relationship**.

If this relationship is a valid target for a relationship shortcut, Framework Manager asks if you want to create a shortcut to that relationship. For more information, see "[Create a Relationship Shortcut](#)" (p. 83).

3. Click **OK**.

The **Relationship Definition** dialog box appears. You can use this dialog box to modify the relationship (p. 82).

Create a Relationship Shortcut

A relationship shortcut is a pointer to an existing relationship. You can use relationship shortcuts to reuse the definition of an existing relationship. Any changes to the source relationship are automatically reflected in the shortcut. You can also use relationship shortcuts to resolve ambiguous relationships between query subjects "[Resolving Ambiguous Relationships](#)" (p. 339).

IBM® Cognos® Framework Manager asks whether you want to create a relationship shortcut whenever you create a relationship and both of the following conditions apply:

- At least one end for the new relationship is a shortcut.
- A relationship exists between the original objects.

Steps

1. Ctrl+click the objects that you want to participate in the relationship shortcut.
2. From the **Actions** menu, click **Create, Relationship**.

Framework Manager asks if you want to create a shortcut to that relationship.

3. Click **Yes**.

A list appears of all relationships in which one end is a model object and the other end is either another model object or a shortcut to another model object.

4. To retrieve all relationships in which both ends can be either a model object or a shortcut to a model object, click **Find All**.
5. Click the relationship that you want to be the target of the relationship shortcut.
6. Click **OK**.

Detect and Generate Relationships

You can use IBM® Cognos® Framework Manager to detect and generate relationships between two or more existing objects in your model. This is useful when you import metadata in stages, or when you want to change the criteria that apply to existing relationships, such as whether they include outer joins.

When importing star schema metadata, avoid generating relationships based on matching column or query item names unless you have naming conventions in place. Data warehouses often apply naming standards to columns, such as `surr_key` as the default column name for surrogate keys in dimensions. In this case, generating relationships that are based on matching column names generates inappropriate relationships between all dimension tables.

Steps

1. Ctrl+click two or more objects.
2. From the **Tools** menu, click **Detect Relationships**.
3. Select the rules you want to apply to each pair of tables.

Rule	Result
Use primary and foreign keys	Creates joins that are based on primary key and foreign key relationships. The query item names do not have to match.
Use matching query item names that represent uniquely indexed columns	Creates joins between query items whose names and data types match, if one or both of the underlying columns are uniquely indexed.
Use matching query item names	Creates joins between query items whose names and data types match. This generates as many relationships as possible.

4. Indicate whether you want Framework Manager to detect and generate relationships between
 - the selected objects
 - each selected object and every object in the project that is not selected
 - the selected objects and every other object in the project

5. Identify whether you want Framework Manager to create outer joins or inner joins based on outer joins that exist in the data source.
6. If you want to disable the automatic prevention of double-counting, convert all *n* cardinalities to 1 by clearing the **Fact detection enabled** check box.

For more information, see ["Guidelines for Modeling Metadata" \(p. 319\)](#).

7. Click **OK**.

Query Subjects

Information about query subjects for SAP BW metadata appears in a different topic [\(p. 216\)](#).

A query subject is a set of query items that have an inherent relationship.

You use IBM® Cognos® Framework Manager to modify query subjects to optimize and customize the data that they retrieve. For example, you can add filters or calculations. When you change the definition of a query subject, Framework Manager regenerates the associated query items, ensuring that any changes to query subject properties are reflected in all query items for that query subject.

There are different types of query subjects in Framework Manager:

- data source query subjects [\(p. 85\)](#)
- model query subjects [\(p. 87\)](#)
- stored procedure query subjects [\(p. 88\)](#)

You may also be interested in this topic, ["Query Subjects vs. Dimensions" \(p. 332\)](#).

Data Source Query Subjects

Query subjects that are based on relational metadata are defined by SQL statements that describe how to retrieve data from the data source.

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

For example, you import the Employee detail fact table from the Great Outdoors Warehouse sample database. Framework Manager then 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. For example, 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.

Each data source query subject can reference data from only one data source at a time. However, 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 that control the data that the query retrieves and create query subjects based on arbitrary SQL.

You may have created SQL statements for query subjects without enclosing references to columns and tables in quotes. This works in relational data source environments that do not use a case-sensitive lookup against the metadata tables they parse. The SQL statement for the query subject may be Cognos SQL (p. 109), native SQL (p. 110), or pass-through SQL (p. 111). 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. Cognos SQL statements, however, 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 appear in the SQL statement.

To use multiple data sources for a query subject, use a model query subject that accesses the data source query subjects or other model query subjects.

Create a Data Source Query Subject

Data source query subjects directly reference data in a single data source. IBM® Cognos® Framework Manager automatically creates a data source query subject for each table and view that you import into your model. You can create additional data source query subjects.

You can add any combination of objects to a query subject, such as query items, other query subjects, or dimensions. You can add stand-alone calculations and filters, and you can also embed calculations and filters in the query subject.

You can also create model query subjects, which are based on metadata that exists in your model (p. 87), and stored procedure query subjects, which are generated from the stored procedures in a relational data source (p. 88).

Steps

1. Select the namespace folder and, from the **Actions** menu, click **Create, Query Subject**.
2. In the **Name** box, type a name for the new query subject.
3. Click **Data Source**, and click **OK**.
4. Complete all the steps in the **New Query Subject** wizard.

To ensure that the data source is uniquely identified for a data source query subject, do not exit the wizard before the **Finish** button appears.
5. Click **Finish**.
6. Right-click the query subject you created and click **Edit Definition**.
7. Click the **SQL** tab, and from the **Available database objects** box, drag objects to the **SQL** box.

If your project contains multiple data sources and you want to add objects from different data sources to the query subject, click **Insert Data Source**, select the required data source, and click **OK**.

You can also insert a macro, embed a calculation (p. 155), and embed a filter (p. 158).
8. Choose the action you want:

Goal	Action
Provide control over granularity	Click the Determinants tab (p. 92).
Test the query subject	Click the Test tab (p. 101).
View the SQL	Click the Query Information tab (p. 106).
View the system tables from the data source	Select the Show System Objects check box.

9. Click **OK**.

A warning appears if any modifications invalidated relationships, other query subjects, calculations, or filters.

10. Ensure that the **Usage** and **Regular Aggregate** properties are set correctly (p. 141).

Model Query Subjects

Information about model query subjects for SAP BW metadata appears in a different topic (p. 217)

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.

Usually, model query subjects are created in the business view, not the import view. For information about the business view, see "[Organizing the Model](#)" (p. 179).

Because model query subjects are based on the metadata in your model, they let you

- reuse complex SQL statements that exist in the model
- reference objects from different data sources in the same query subject

If you import a model query subject from another model, the model query subject will not work unless you also import the data source query subjects that the model query subject references.

If you want to edit the SQL, you must convert the model query subject into a data source query subject (p. 106).

Create a Model Query Subject for Relational Metadata

Model query subjects 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.

You can add any combination of objects to a query subject, such as query items, other query subjects, or dimensions. You can add stand-alone calculations and filters, and you can also embed calculations and filters in the query subject.

You can create a new model query subject by merging existing query subjects and query items (p. 96). You can also create data source query subjects, which directly reference data in a single data source (p. 85), and stored procedure query subjects, which are generated from the stored procedures in a relational data source (p. 88).

When you use a model query subject in the IBM Cognos studios, IBM Cognos BI uses the relationships you have defined for the model query subject, not the relationships defined for the underlying data source query subjects. If you want to ensure that the relationships defined for the underlying data source query subjects are used, do not define relationships or determinants for the model query subject.

Steps

1. Select the namespace folder and, from the **Actions** menu, click **Create, Query Subject**.
2. In the **Name** box, type a name for the new query subject.
3. Click **Model**, and click **OK**.
4. Click the **Query Subject Definition** tab.
5. To add items to the model query subject, drag items from the **Available Model Objects** box to the **Query Items and Calculations** box.

You can change the order of items and calculations. However, if the query subject contains a query item folder, you can change the order only in the **Project Viewer**.

6. You can also embed a filter.
If you add calculations (p. 155) or filters (p. 158) to a model query subject, Framework Manager must go to the data source instead of simply accessing the model.
7. If you want to test the query subject, click the **Test** tab (p. 101).
8. Click **OK**.

A warning appears if any modifications invalidated relationships, other query subjects, calculations, or filters.

9. Ensure that the **Usage** and **Regular Aggregate** properties are set correctly (p. 141).

You may be interested in the following related topics:

- embedded calculations (p. 237)
- embedded filters (p. 239)
- testing and setting test options (p. 213)
- modifying the properties for multiple query subjects at the same time (p. 34)

Stored Procedure Query Subjects

Stored procedure query subjects are generated when you import a procedure from a relational data source. IBM® Cognos® 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 may return.

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

Each result set must return the same form, such as the same number, types, and names of columns. Overloaded signatures are supported by IBM Cognos BI, but each procedure must be defined as a uniquely named procedure with a separate query subject for each result set.

Output parameters are not supported.

After you import or create a stored procedure query subject, it appears 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 could be a function. If the test fails, the query subject is a function and must be deleted.

For more information, see ["Create or Modify a Stored Procedure Query Subject" \(p. 90\)](#).

Stored Procedures from Informix Data Sources

If you have stored procedures from Informix Dynamic or Parallel Server data sources, you must edit the parameters. We recommend that you refer to the original source of the stored procedures to ensure that they are mapped correctly.

Informix 7.x and 8.x provide only the name of the stored procedure to Framework Manager. You must provide all parameters, such as the parameter name, data type, mode, size, precision, scale, and value so that a result set can be obtained. Informix 9.x provides metadata for stored procedures and user defined functions with default parameter values. Check all parameters before using them, especially the mode attribute.

Informix functions are imported as stored procedures. After you import them, you must change them to functions by clicking the **f(x)** button in the **Edit Definition** dialog box. This button is enabled only for these functions. Then select the argument that represents the results or use the values obtained from the test results.

Stored Procedures from Composite Information Server

If you have stored procedures from Composite Information server, we recommend that you refer to the original source of the stored procedures to ensure that they are mapped correctly.

Composite functions are imported as stored procedures. After you import them, you must change them to functions by clicking the **f(x)** button in the **Edit Definition** dialog box. This button is enabled only for these functions. Then select the argument that represents the results or use the values obtained from the test results.

Create or Modify a Stored Procedure Query Subject

After you import or create a stored procedure query subject, you can modify it. To avoid inconsistencies, the modified query subject should return the same result set structure as the original stored procedure.

IBM® Cognos® Framework Manager supports only user-defined stored procedures. System stored procedures are not supported.

There are different types of stored procedures:

Type of Stored Procedure	Description
Data Query	<p>Issues a read-only transaction</p> <p>If you have a stored procedure with its type set to Data Query, the stored procedure issues a read-only transaction. When you run the stored procedure in Event Studio, an error message says that the stored procedure wants to update the database. The reason for the error is that the stored procedure contains a passive transaction that is supported by the underlying database. The solution is to click OK so that the stored procedure updates the database. No other action is required.</p>
Data Modification	<p>Writes a record to the data source. Use this type when you want to use the stored procedure in Event Studio.</p> <p>If you want Event Studio users to be able to select a parameter in a task, you must put quotation marks around the parameter.</p> <p>Warning: Testing a data modification stored procedure in the Edit Definition dialog box results in data being written to the data source. You cannot roll back transactions to the data source in Framework Manager. If undesired data is written to the data source as a result of testing the stored procedure, a rollback can be done by the database administrator if the data source is configured to support it. To test the stored procedure without data being written to the data source, click Test from the Tools menu.</p>

You can also create data source query subjects, which directly reference data in a single data source (p. 85), and model query subjects, which are based on metadata that exists in your model (p. 87).

Steps

1. Do the following:

Goal	Action
Create a stored procedure query subject	<p>Select the namespace folder and, from the Actions menu, click Create, Query Subject.</p> <p>In the Name box, type a name for the new query subject.</p> <p>Click Stored Procedure, and click OK.</p> <p>Complete all the steps in the New Query Subject wizard.</p>
Modify a stored procedure query subject	<p>Select the stored procedure query subject that you want to modify.</p> <p>From the Actions menu, click Edit Definition.</p>

- Click the **Definition** tab and choose the action that you want.

Goal	Action
Use a different stored procedure	In the Stored Procedure Name box, type the name of the stored procedure.
Change the type of the stored procedure	From the Type box, select Data Query or Data Modification .
Change which data source the stored procedure is in	<p>Click the ellipsis (...) button next to the Data Source box.</p> <p>When you import a stored procedure, a new data source is created. You can point to the original data source and delete the new one.</p>
Edit an argument	<p>Click the argument and click the ellipsis (...) button.</p> <p>The Syntax box in the Query Subject Definition dialog box shows the correct syntax to use.</p>
Generate the projected query items	Click the Test tab (p. 101).

- Click **OK**.

Framework Manager runs the stored procedure and, if the query subject returns a result set, validates the query subject.

If the stored procedure does not return a result set, the query subject becomes an invalid query subject if saved in the model. If the invalid query subject is included in the published package, the invalid query subject cannot be used in a report.

4. Ensure that the **Usage** and **Regular Aggregate** properties are set correctly for each newly created query item.

For example, a query item may be set as a fact when it is an identifier.

You can update the stored procedure query subject if the data source changes ([p. 104](#)).

Example - Use Prompts with a Stored Procedure

If you define prompts for stored procedure variables, your users can set the variables in reports.

Steps

1. Create a stored procedure query subject that uses the `sp_FIND_ORDER_DATE` stored procedure. The **Query Subject Definition** dialog box appears.

2. On the **Definition** tab, select the `@order_number` argument, and click the ellipsis (...) button.

3. In the **Value** box, type the following macro syntax and then click **OK**:

```
#prompt('Order Number','integer')#
```

Note: Framework Manager removes anything that is outside the number signs when running the macro.

4. If you want to test the prompt for the variable, do the following:

- Click the **Test** tab, and then click **Test Sample**.

The **Prompt Values** dialog box appears.

- In the **Name** column, click **Order Number**.
- In the **Value** field, type **1234** and click **OK**.

One record is returned, showing the date for Order Number 1234.

Framework Manager uses this value for the duration of the current session or until you clear the prompt value.

5. Click **OK**.

Determinants

Determinants reflect granularity by representing subsets or groups of data in a query subject and are used to ensure correct aggregation of this repeated data. Determinants are most closely related to the concept of keys and indexes in the data source and are imported based on unique key and index information in the data source. We recommend 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.

An example of a unique determinant is Day in the Time example below. An example of a non-unique determinant is Month; the key in Month is repeated for the number of days in a particular month. When you define a non-unique determinant, you should specify **Group By**. This indicates to IBM Cognos software that when the keys or attributes associated with that determinant are repeated in the data, it should apply aggregate functions and grouping to avoid double-counting. It is not recommended that you specify determinants that have both **Uniquely Identified** and **Group By** selected or have neither selected.

Year Key	Month Key	Month Name	Day Key	Day Name
2006	200601	January 06	20060101	Sunday, January 1, 2006
2006	200601	January 06	20060102	Monday, January 2, 2006

You can define three determinants for this data set as follows -- two **Group By** determinants (Year and Month) and one unique determinant (Day). The concept is similar but not identical to the concept of levels and hierarchies.

Name of the Determinant	Key	Attributes	Uniquely Identified	Group By
Year	Year Key	None	No	Yes
Month	Month Key	Month Name	No	Yes
Day	Day Key	Day Name Month Key Month Name Year Key	Yes	No

In this case, we use only one key for each determinant because each key contains enough information to identify a group within the data. Often Month is a challenge if the key does not contain enough information to clarify which year the month belongs to. In this case, however, the Month key includes the Year key and so, by itself, is enough to identify months as a sub-grouping of years.

Note: While you can create a determinant that groups months without the context of years, this is a less common choice for reporting because all data for February of all years would be grouped together instead of all data for February 2006 being grouped together.

When to Use Determinants

While determinants can be used to solve a variety of problems related to data granularity, you should always 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.

For example, Time has multiple levels, and it is joined to Inventory on the Month Key and to Sales on the Day Key. For more information, see ["Multiple-fact, Multiple-grain Queries" \(p. 326\)](#).

- There is a need to count or perform other aggregate functions on a key or attribute that is repeated.

For example, Time 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. In SQL, that is `XMIN(Days in the month for Month_Key)`. There is also a `Group by` clause in the Cognos SQL.

There are less common cases when you need to use determinants:

- 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. Override the determinants imported from the data source that conflict with relationships created for reporting.

You cannot use multiple-segment keys when the query subject accesses blob data. With summary queries, blob data must be retrieved separately from the summary portion of the query. To do this, you need a key that uniquely identifies the row and the key must not have multiple segments.

- A join is specified that uses fewer keys than a unique determinant that is specified for a query subject.

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 relationships, there will be a conflict. 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. Modify the determinant information of the query subject if it is not appropriate to use the additional columns in the relationship.

Specify a Determinant

Determinants provide control over granularity for query subjects.

If a query subject has determinants, each query item of the query subject must be included in one of the determinants.

Determinants are processed in the order in which they are specified in the model. You can change the order of the determinants. If a query subject has more than one determinant, the first one that

covers all the requested items is used. Determinants are evaluated in the context of each required join as well as the context of requested items.

Data source query subjects are imported with determinants defined for them. These default determinants are generated based on keys and indexes in the data source.

Constraints

Model query subjects do not have determinants defined for them automatically. If determinants are needed, you must define them manually.

Stored procedure query subjects do not have determinants.

You cannot use determinants with user-entered SQL that was specified in a query defined in Report Studio.

Steps

1. Click the query subject you want and, from the **Actions** menu, click **Edit Definition**.
2. Click the **Determinants** tab.
3. Click **Add** under the **Determinants** box.

The entry **New Determinant** appears in the box. To give this entry a meaningful name, right-click it, and click **Rename**.

4. To define a key, right-click a query item in the **Available items** box and click **Add as Key**.

Tip: You can also drag query items to the **Key** box.

5. To identify which query items should be associated with this determinant, right-click query items in the **Available items** box and click **Add as Attributes**.

Tip: You can also drag query items to the **Attributes** box.

You can have a determinant with no attributes defined for it. Framework Manager uses this type of determinant to indicate which query items are indexed.

6. To specify that the selected determinant should be used as the unique identifier, select the **Uniquely Identified** check box.

Do this only if the data in this item is unique for every row in the underlying data source.

You can specify more than one unique determinant if they are truly unique. At query time, the relationship being used will determine which unique determinant to use.

7. Select the **Group By** check box to indicate that when keys or attributes associated with that determinant are repeated in the data, IBM Cognos BI should apply aggregate functions and grouping to avoid double-counting.
8. If you want to change the order of the determinants, use the arrow buttons.
Determinants are processed in the order in which they are specified in the model.
9. Click **OK**.

For more information, see ["Determinants" \(p. 322\)](#) and ["The SQL Generated by IBM Cognos Software" \(p. 351\)](#).

The Effect of Determinants on SQL

It is important to understand the effect that determinants have on the SQL that is generated. Determinants affect the grouping and aggregation of data, including other query subjects that have relationships with the query subject as well as the query subject itself.

For example, consider the following information. Each Product Line contains many occurrences of Product Type. Each Product Type contains many occurrences of Product. For Product, Product Key is a surrogate key and Product Number is a business key that is used as an attribute. Data joined on Product Key is aggregated correctly when reported by Product Line or Product Type or both.

Determinant	Key	Group By	Uniquely Identified	Attributes
Product line	Product line code	Yes		Product line
Product type	Product type code	Yes		Product type
Product	Product key		Yes	Cost Margin Product name Product number

Create a Model Query Subject Based on Existing Objects

Information about SAP BW model query subjects appears in a different topic ([p. 217](#)).

You can select existing model objects and merge them into a new model query subject. This means that you can reuse existing metadata to quickly create query subjects.

The objects that you can merge include

- relational data source query subjects and their shortcuts
- model query subjects and their shortcuts
- query items, filters, and calculations in model and data source query subjects
- relationships and relationship shortcuts between model and data source query subjects

You can merge any number of the same type of objects into a new query in a single operation. The merge always creates a new model query subject.

The new query subject contains any filters that exist in the original query subject.

Steps

1. Ctrl+click the objects that you want to merge into a single query subject.
2. From the **Actions** menu, click **Merge in New Query Subject**.

View Related Objects




You can explore a visual representation of the objects that are connected to the query subject or dimension that you select in the **Project Viewer**. The **Context Explorer** shows the objects that the selected object is connected to. You can also select a connected object and see its references.

You can hide an object in the **Context Explorer**. You can also change the layout, fit all objects in the **Context Explorer**, zoom in and out, print, preview diagrams before printing, and change the page setup.

You can also use the **Dimension Map** tab to explore dimensions.

Steps

1. Select one or more objects that you want to explore.
2. From the **Tools** menu, click **Launch Context Explorer**.
3. To see the connected objects, click one or more objects and click the appropriate button.

Goal	Button
View the objects that are related to the selected object.	
View the immediate references for the objects.	
View all references for the objects.	

4. If you want to see details about an object, such as its relationships and query items, right-click the object, click **Navigate Diagram**, **Diagram Settings**, and then select the details you want.

Create a Query Set

A query subject can be defined using the `set` operations of union, intersect, or except. You define a query set to merge, compare, or equate similar data from different data sources. Query sets are useful when modeling data from disparate systems.

There are many reasons for creating a query set. A query set may be needed to create a conformed dimension across disparate data sources. Or, you may want to compare the contents of two queries to determine whether the queries contain the same data; this is common in test environments. Or, you may want to compare queries that return nulls. Or, you want to handle a fact-to-fact relationship that is truly a one-to-one relationship. (If it is not truly a one-to-one relationship, create a multiple-grain query ([p. 326](#)).)

A query set can consist of only two query subjects. You can create a query set that merges two other query sets together. A query set can contain

- all the rows of two query subjects (union operation)
For example, your company recently acquired another company and you need a complete list of all customers.
- only the rows that are shared between the query subjects (intersect operation)
For example, you want to find out which staff members are also managers.
- only the rows that exist in the first query subject and not in the second query subject in the query set (except operation)
For example, you want to highlight the differences between where your products were sold this year and ten years ago.

The names of the items in the projection list default to the items assigned to the first query subject in the set operation.

Relationships between the two query subjects in the query set and other query subjects are not included in the query set.

Reports show different results depending on which operator is used. For example, you have two query subjects with the names of various employees.

The first query subject contains these rows:

Row	Value
1	Jane
2	John
3	John
4	Michael
5	Michael

The second query subject contains these rows:

Row	Value
1	Jane
2	John
3	John
4	Patrick

You create a query set. You see different results depending on the operator you use.

Operator	Result	Notes
Union	Jane, John, Michael, Patrick	All items are shown. Values are not duplicated.
Intersect	Jane, John	Items in common are shown. Values are not duplicated.
Except	Michael	Items that are not common are shown. Values are not duplicated. If the second query subject were listed first in the query set, the result would be Patrick.
Union All	Jane, Jane, John, John, John, John, Michael, Michael, Patrick	All items are shown. Values are duplicated.
Intersect All	Jane, John, John	Items in common are shown. Values are duplicated.
Except All	Michael, Michael	Items that are not common are shown. Values are duplicated. If the second query subject were listed first in the query set, the result would be Patrick.

Constraint

Not all data types are supported. Generally, sets are not permitted on `BFILE`, `BLOB`, `CLOB`, `LONG`, and `VARRAY` data types, or on nested table columns.

Steps

1. Select two query subjects that meet these requirements:

- Each query subject must have the same number of columns.
- Columns must be in the same order.
- Columns must have the same or similar data types.

The data types do not need to be exactly the same if those in the second result set can be automatically converted by the data source to data types compatible with those in the first result set.

For example, one query subject contains country data and uses `int` as the data type. Another query subject contains country data and uses `smallint` as the data type. Framework Manager imports these query subjects as `int16` and `int32` and performs a `set` operation.

2. From the **Actions** menu, click **Define Query Set**.
3. Click the **Definition** tab.
4. In the **Name** box, give the query set a name.
5. Review the **Query Subject** boxes to ensure the order that the query subjects will appear in the `Select` clause is correct.

The order could be important if you want a specific set of column names (aliases) that appears in only one of the query subjects. If the order is incorrect, cancel this query set and start again.

For union and intersect, the order of the query subjects does not matter. You can change the order and receive the same answer. For except, the order of the query subjects does matter.

6. Use the **Operator** box to define how the rows of the query subjects are combined.

Option	Description
Union	Retrieves all unique rows from both sets. Duplicates are removed.
Intersect	Retrieves rows that are common between the query subjects.
Except	Retrieves rows that exist in the first query subject and not in the second query subject.

7. To create a Union All, Intersect All, or Except All operation, clear the **Remove Duplicate Row** check box.
8. Choose the action that you want.

Goal	Action
Work with the calculations that are embedded in the query subjects	Click the Calculations tab. You can add or edit the calculations and change the order of the calculations.
Work with the filters that are embedded in the query subjects	Click the Filters tab. You can add or edit the filters, change the order of the filters, and change the usage of filters.
Test the query set	Click the Test tab.

9. Click **OK**.

You may be interested in the following related topics:

- [embedded calculations \(p. 155\)](#)
- [embedded filters \(p. 158\)](#)
- [determinants \(p. 92\)](#)
- [testing the query set or changing the test settings \(p. 101\)](#)

Test a Query Subject or Query Set

You can see the results that an object returns by testing it. You can test when creating an object or later on. The objects you can test are dimensions, query subjects, query sets, hierarchies, levels, calculations, and query items.

You can view the data that will appear in a specific report before publishing a package by selecting and testing the objects that will appear in the report. This makes it easier to debug a model and to verify that the model meets the reporting requirements because you do not need to create and publish packages first.

When you test an object, IBM® Cognos® Framework Manager returns sample data. Formatting is not applied to the sample data. If you must test formatting, you must publish the package and view the objects in the IBM Cognos studios.

You may see different results depending on what you test. For example, if you use the expression editor to test a calculation that is embedded in a query subject, Framework Manager tests only the expression, not the item, so the aggregation setting for the query item is not applied to the test. Testing the entire query subject, which includes the calculation, gives a different result because the aggregation setting is applied. For example, if the aggregation setting is summarize, you can see a smaller number of rows in the test.

If you test a child segment of a segmented model, you may see an error if an object you are testing refers to an object in another child segment and the referenced object is not available to the project you are in. We recommend that you check that the parent model contains all the objects and that this error message does not appear when you test the parent model.

Governor settings may affect the testing results. For more information, see ["Set Governors" \(p. 304\)](#).

You can change existing test settings to customize the results that the test shows. For example, in addition to other settings, you can control the number of rows returned.

Steps When Creating or Modifying the Object

1. Select the object you want to test.
2. From the **Actions** menu, click **Edit Definition**, and click the **Test** or **Query Information** tab.

The **Test Results** box is initially empty until you run the query.

Any result sets that contain binary large objects are shown as [blob].

3. To run the query and bring back all the test results, click **Test Sample**.

- 4. If you want to add a count of the rows, click **Total Rows**.
- 5. If you want to apply the **Regular Aggregate** property of the query item or the **Aggregate Rules** property of a semi-additive measure that is referenced in the expression, select the **Auto Sum** check box.

If you clear this check box, a row is returned for each row in the result set of the query.
- 6. If you want to obtain more information about the query results, click the **Query Information** tab.
- 7. Click **OK**.

Steps to View the Data That Will Appear in a Specific Report

- 1. Select the objects that will appear in the report.
- 2. From the **Tools** menu, click **Test**.
- 3. To run the query and bring back all the test results, click **Test Sample**.
- 4. To view details about any problem that is found, click the **Query Information** tab.

If you do not see the results of the query in the test window, the data from your data source may exceed the value of one of the governors. The query stops at the specified limit, but the test result window does not contain any data. **Tip:** Set each governor to zero.

Change the Test Settings

You can customize the tests by changing the test settings.

Steps

- 1. Select the object that you want.
- 2. From the **Actions** menu, click **Edit Definition** and then click the **Test** tab or the **Query Information** tab.
- 3. Click **Options** and then click the **Test Settings** tab.
- 4. Choose the options that you want.

Goal	Action	Persistence
Retrieve all data and show a specified number of rows	Select the Restrict the maximum number of rows to be returned check box and type the required number of rows. This setting does not improve performance for retrieving data when testing dimensions, query subjects, and query sets.	This setting applies to all dimensions, query subjects, and query sets in the model. This setting is saved and used in your next session with any model.

Goal	Action	Persistence
Specify the level of detail	Drag the Level of Information shown in Results Information slider to the location that represents the amount of detail you require.	This setting is saved and used in your next session with this model.
Temporarily override session parameters	In the Session Parameters box, click Set . The Session Parameters dialog box appears.	The override values are not saved with the model. This setting is for your current session only.
Apply relevant design mode filters	Select the Apply all relevant design mode filters when testing check box. This applies all relevant filters whose usage is set to design mode in another dimension, query subject, or query set.	This setting is saved and used in your next session with any model.
Apply a security filter	In the Security Filters box, click Edit .	This setting is saved and used in your next session with this model.
Change the prompt values	In The Current Prompt Values box, click Prompts . The Model Prompts Manager dialog box appears, which shows all prompts, and their values, that are in the model.	The prompt values are not saved with the model. This setting is for your current session only.

5. Click **OK** two times.

You may be interested in the following related topics:

- setting governors ([p. 304](#))
- security filters ([p. 257](#))
- temporarily overriding session parameters ([p. 165](#))
- changing prompt values ([p. 149](#))
- working with dimensions ([p. 114](#))
- working with query subjects ([p. 85](#))

Validate a Query Subject

Information about validating SAP BW query subjects appears in a different topic (p. 218).

You can validate the definition of the query subject without having to open the **Query Subject Definition** dialog box. This is useful to do when

- new query items were added to a query subject
- the definition of the underlying query subject has changed
- the parameters of a stored procedure were changed

The **Evaluate Object** command completes an exhaustive check of all query subjects and ensures that they can run.

What happens in the evaluation process depends on the type of query subject selected.

Type of query subject	Evaluation process
Relational data source query subject	A request based on the derived items is sent to the relational data source. The list of data source references is updated. The physical attributes, such as data type, are updated as needed.
Model query subject based on relational metadata	A request based on the derived items is sent to the data source. The cached SQL, if available, is updated. The physical attributes, such as data type, are updated as needed.
Stored procedure query subject	A request based on the latest parameters of the stored procedure is sent to the data source. The list of derived query items is updated.

You can also update the query subject (p. 104) if it is from a relational data source, or synchronize the entire project (p. 301).

Steps

1. Select the query subject that you want to evaluate.
2. From the **Tools** menu, click **Evaluate Object**.

If you changed the **Regular Aggregate** property to **unsupported**, the property is reset when you evaluate the query subject. If the property is set to any other value, the property is not changed.

Update Query Subjects

If you are using a relational data source, you can choose to update only the query subjects instead of performing a full project synchronization. You must perform a project synchronization to synchronize changes in another data source.

The query subject is updated based on the definition in the data source. When you update a query subject, new metadata is fetched from the data source and query items are re-synchronized.

You can also evaluate the query subject, if it is from a relational data source ([p. 104](#)).

Constraint

You cannot use the **Update Object** command for model query subjects.

Steps

1. Select one or more query subjects.
2. From the **Tools** menu, click **Update Object**.

Tip: You can instead open the **Query Subject Definition** dialog box and click **OK**.

Convert a Query Subject into a Dimension

You can convert a query subject into a regular dimension or a measure dimension when you want to use features associated with dimensions, such as defining hierarchies and levels. A model query subject becomes a model dimension.

While you can convert data source query subjects to data source dimensions, data source dimensions have limited functionality in comparison to query subjects or model dimensions. It is recommended that you discontinue using data source dimensions (both regular and measure). Create new models following the recommendations in "[Guidelines for Modeling Metadata](#)" ([p. 319](#)) to use query subjects as the relational foundation of the model. Define regular and measure dimensions as model objects based on data source query subjects or model query subjects or both. Guidance on migration for existing users of data source dimensions will be provided in a future release.

If the query subject has determinants specified for it, the keys you specified for each determinant is used to build a hierarchy with a business key for each level. The determinants form one hierarchy. The first string attribute for each determinant is used for the business caption. If the caption is not the attribute you want to use, you must change it manually.

Converting a query subject into a dimension is simply a starting point. We recommend that you examine each dimension that is created this way to ensure that it reflects your requirements.

You can also convert a dimension into a query subject ([p. 130](#)).

Constraints

You cannot use determinants to create separate hierarchies for the dimension. You must create the separate hierarchies for the dimension after converting the query subject.

You cannot convert the following to dimensions:

- query sets
- stored procedure query subjects

- SAP BW query subjects

Steps

1. Select the query subjects that you want to convert.
2. From the **Actions** menu, click **Convert to Regular Dimension** or **Convert to Measure Dimension**.

Convert a Model Query Subject into a Data Source Query Subject

You can convert a model query subject into a data source query subject if you want to edit the SQL.

Constraint

Do not convert the model query subject if you want it to reference multiple data sources.

Steps

1. Select the model query subject that you want to convert.
2. From the **Actions** menu, click **Convert to Data Source Query Subject**.

This command is available only if you have run the query and the **Query Information** tab in the **Edit Definition** dialog box contains SQL.

Edit the SQL

SQL is the industry-standard language for creating, updating, and querying relational database management systems. When you edit the definition of a relational data source query subject, you can use

- Cognos SQL ([p. 109](#))
- native SQL ([p. 110](#))
- pass-through SQL ([p. 111](#))

If you want to edit the SQL of a model query subject, you must copy the SQL for the model query subject from the **Query Information** tab and paste it into a new data source query subject. You can also convert the model query subject into a data source query subject ([p. 106](#)). Do not edit the SQL if you want the model query subject to reference multiple data sources.

Changing the alias of a column regenerates the query item that represents that column. Any modifications that you made to the query item are not retained because IBM® Cognos® Framework Manager considers it a new query item.

You can add comments to the SQL by using `/*` before the comment and `*/` at the end.

Here is an example:

```
select country /* this is a multiline
comment
another line
another line
```

*/

Steps

1. Click the data source query subject that you want to change.
2. From the **Actions** menu, click **Edit Definition**.
3. Click the **SQL** tab, and drag objects into the **SQL** box or type in the SQL you want.
4. Click **OK**.

Change the Type of SQL

When choosing the type of SQL in which to generate a data source query subject, you must weigh the following factors and decide which are most important.

Type	Advantage	Disadvantage
Cognos SQL	Cognos SQL improves query subject performance; for example, by removing unused elements at query time. SQL works on any supported database.	You cannot enter non-standard SQL.
Native SQL	Performance is optimized across all related query subjects. You can use SQL that is specific to your database.	You cannot use SQL that the data source does not support for sub-queries. The SQL may not work on a different database type.
Pass- Through SQL	You can enter any SQL supported by the database.	There is no opportunity for Framework Manager to automatically optimize performance. The SQL may not work on a different data source.

Prerequisites for Changing to Native SQL

If you change an existing query subject to native SQL, you must first ensure that the SQL reflects the rules that apply to the native data source so that your query runs properly. You must do the following:

- Edit existing table names.

Cognos SQL uses a two-part structure to name query subjects. For example, [gosales] . [ProductLine] means that the ProductLine query subject comes from the gosales database.

Therefore, when you switch to native SQL, you must ensure that all table names include the parent elements required by the data source vendor.

For information about naming conventions, see ["Naming Conventions for Objects in a Project" \(p. 38\)](#).

- Ensure that the SQL is valid for subqueries.

IBM® Cognos® Framework Manager processes native SQL query subjects as subqueries. For example, here is a Cognos SQL query subject:

```
Select
P.ProductName, P.Margin From Product P
```

If you change it to native SQL, Framework Manager generates the following statement:

```
Select
oracle_plain.ProductName as Productname,
oracle_plain.Margin as Margin
From
(GOSALES1_OR_92_WE...SELECT
P.PRODUCTNAME, P.MARGIN
FROM
PRODUCT P}
)oracle_plain
```

Therefore, you must ensure that the query subject adheres to additional database restrictions that are imposed on subqueries, such as not using the `With` clause. Pass-through SQL does not have the same restrictions. However, the fact that native SQL is processed as part of a larger query improves its performance.

To test native SQL using a query tool, such as Oracle's SQL*Plus, you must place the SQL in the `From` clause of a `Select` statement. For example, you can use the following syntax in a query tool:

```
Select * from (<Native SQL>) T1
```

- Assign aliases to derived columns.

Assign alias names to any column whose values are calculated. Here is an example:

```
SELECT
Length(Country) as LGTH
FROM Country
```

- Insert double quotation marks around alias names.

Changing the SQL type of a query subject can change the case of alias names. When this happens, any query subject that references the changed query item becomes invalid. To ensure that there is no case change, insert double quotation marks around the alias, such as

```
Select
COUNTRY as "test" from COUNTRY
```

- If a data source query subject contains a macro in the projection list (`Select` clause) of the SQL statement, specify an alias in the SQL that matches the **Column Name** property of the query item.

An error could occur because the macro evaluates to a column name that is different from the **Column Name** property of the corresponding query item. The result is that the system is unable to locate the item in the projection list. Projection lists are static.

Assigning an alias ensures that the name of the item in the projection list remains constant, as the results of evaluating the macro change.

For example, the following query contains a session parameter, `runLocale`, whose value specifies which column the query retrieves:

```
Select
#$ColumnMap{$runLocale}# as
CountryNameAlias
From
[GoSales].Country
```

Note that the number sign (#) is reserved for macros. Framework Manager removes anything that is outside the number signs when running the macro.

Steps

1. Click the query subject that you want to change.
2. From the **Actions** menu, click **Edit Definition**, and then click the **Query Information** tab.
The **Test Results** box is initially empty until you run the query.
3. Click **Options**, and then click the **SQL Settings** tab.
4. Use the **SQL Type** list to change the type of SQL.
If you are changing the type to native SQL, see the checklist above to ensure that the SQL reflects the rules that apply to the native data source.
5. Click **OK**.
6. If you want to see the SQL, click **Test Sample**.
7. If you want to see the actual query, click **Query**.
8. If you want to see the xml that IBM Cognos BI uses, click **Response**.
9. Click **OK**.

Cognos SQL

By default, IBM® Cognos® Framework Manager uses Cognos SQL to create and edit query subjects. Cognos SQL adheres to SQL standards and works with all relational and tabular data sources. Framework Manager generates the most optimized SQL possible. In this way, Cognos SQL is preferable.

Because query subjects in Framework Manager are similar to views in databases, the SQL for each query subject must conform to the SQL standards that apply to views. For example, you must assign aliases to any column that is empty or whose name is not unique. This level of conformance means that Cognos SQL behaves more consistently than vendor-specific SQL, which does not adhere to SQL standards.

In general, using Cognos SQL is preferable because you can create query subjects that

- can contain metadata from multiple data sources
- have fewer database restrictions
- interact more effectively with IBM Cognos applications

Constructs of the SQL Standard

If the data source supports it, you can use the `With` clause with Cognos SQL. The `With` clause is used to generate more readable SQL and to let the data source generate a more optimal plan for data retrieval. The data source can more easily detect the cases where the same tables must be scanned and can then resolve these as an inline view or temporary table.

By default, IBM® Cognos® Framework Manager uses the common table constructor from the SQL standard when the **Use With clause when generating SQL** governor is set.

Use the `With` clause for better query performance if the request is restricted to functionality supported by the underlying data source software. When a request uses functionality that is not supported by the data source, using the `With` clause may cause additional decomposition of the query, which can lead to degraded performance. In this case, not using the `With` clause may generate a better set of queries to the underlying data source.

Here is an example of Cognos SQL using derived tables:

```
SELECT * FROM
(SELECT SNO C1, AVG(QTY) C2, COUNT(*) C3 FROM
SUPPLY
GROUP BY SNO) T1,
(SELECT MAX(QTY) C1 FROM SUPPLY) T2
```

The following shows how Cognos SQL turns the above example into a `With` clause:

```
WITH T1 AS (SELECT SNO C1, AVG(QTY) C2, COUNT(*)
C3 FROM
SUPPLY GROUP BY SNO),
T2 AS (SELECT MAX(QTY) C1 FROM SUPPLY)
SELECT *FROM T1, T2
```

Do not use the `With` clause for recursive processing.

For more information about the `With` clause, see ["Set Governors" \(p. 304\)](#).

Data type checking and SQL validation are continually being improved. Because of this and because not all vendors are completely compliant with the SQL standard, invalid or ambiguous SQL expressions that previously were passed to the data source will no longer be passed down. If you have an expression that returns a data type not specified by the SQL standard, we recommend that you pass the expression to the data source by using the syntax `{expr}`. Your users should use the same technique.

Native SQL

Native SQL is the SQL that the data source uses, such as Oracle SQL. Use Native SQL to pass the SQL statement that you enter to the database. IBM Cognos BI may add statements to what you

enter. You can not use native SQL in a query subject that references more than one data source in the project.

SQL specified in IBM® Cognos® Framework Manager and processed by the database, whether native or pass-through, must be completely self-contained. It must not reference anything outside that SQL, such as database prompts, variables, or native formatting that would normally be supplied by the calling application.

If you are comfortable working with a native SQL version, you may want to use it for query subjects that are based on a single data source. By doing so, you can use keywords that are not available in Cognos SQL, and copy and paste SQL from another application into Framework Manager.

When the query is generated, Framework Manager combines the SQL of each query subject that uses a given data source connection into a single query. This helps improve the performance of the query. However, because the SQL is being generated as a series of subqueries, native SQL queries must adhere to any restrictions that their database vendor places on derived tables.

Here is an example of native SQL that returns a list of employees and managers:

```
SELECT
    lpad(' ', (level-1)* 4) ename
EMP_CHART,
    level, empno, ename, job, mgr
FROM
    emp
CONNECT BY PRIOR
    empno = mgr
AND
    deptno not in (20,30)
START WITH
    mgr IS NULL
ORDER BY
    level, job
```

Pass-through SQL

Use pass-through SQL when the SQL statement that you enter is not valid inside a derived table. 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.

Because each query subject is sent to the data source as a separate statement rather than being optimized by IBM® Cognos® Framework Manager, performance is slower. Therefore, in choosing between native SQL and pass-through SQL, you must decide which is more important: performance or using SQL that is not permitted in a subquery.

Generally, you should use pass-through SQL only if you must create a query subject that contains constructs that are specific to a data source and that cannot be used inside a derived table, such as in a `With` or `OrderBy` clause.

SQL specified in Framework Manager and processed by the database, whether native or pass-through, must be completely self-contained. It must not reference anything outside of that SQL, such as database prompts, variables, or native formatting that would normally be supplied by the calling application.

For example, here is a systems-oriented report that contains the system date:

```
SELECT
    TO_CHAR(SYSDATE, 'DAY, DDTH MONTH YYYY')
FROM
    SYS.DUAL
```

Note that the number sign (#) is reserved for macros and that column names must be unique. Framework Manager removes anything that is outside the number signs when running the macro.

Change How the SQL Is Generated

You can specify how IBM® Cognos® Framework Manager generates the SQL that retrieves data from relational data sources for data source query subjects (p. 85) or model query subjects (p. 87).

The **SQL Generation** type of a query subject can be set to either **As View** or **Minimized**. By default, it is set to **Minimized**.

When the generation type is set to **Minimized**, the generated SQL contains only the minimal set of tables and joins needed to obtain values for the selected query items.

When the generation type is set to **As View**, Framework Manager generates queries that contain the full SQL statement that defined the query subject. Use **As View** when you want to ensure that the query is run as a block. The SQL is treated as a view. For example, you want the query to return the same number of rows each time that it is run.

Using minimized SQL improves performance, resulting in a query that runs significantly faster. Generating minimized SQL is especially beneficial for query subjects that represent dimension tables. By using a single model query subject to model a dimension, you can benefit from small SQL queries that run significantly faster.

For example, the **SQL Generation Type** of the following query subject is **As View**. Note that this query subject contains a nested select statement.

```
select
    New_Query_Subject.COUNTRYCODE as COUNTRYCODE,
    New_Query_Subject.EUROINUSESINCE as
EUROINUSESINCE
from
    (select
        CONVERSIONRATE.COUNTRYCODE as COUNTRYCODE,
        COUNTRY.EUROINUSESINCE as EUROINUSESINCE
    from
        "2 - GOSales1 - OLE-DB".GOSALES1.dbo.CONVERSIONRATE
        "2 - GOSales1 - OLE-DB".GOSALES1.dbo.COUNTRY COUNTRY
    where
        (COUNTRY.SALESCOUNTRYCODE = CONVERSIONRATE.COUNTRYCODE)
```



```
) New_Query_Subject
```

If you change the **SQL Generation Type** to **Minimized**, Framework Manager generates the following simplified SQL:

```
select
  CONVERSIONRATE.COUNTRYCODE as COUNTRYCODE,
  COUNTRY.EUROINUSESINCE as EUROINUSESINCE
from
  "2 - GOSales1 - OLE-DB".GOSALES1.dbo.CONVERSIONRATE
  CONVERSIONRATE,
  "2 - GOSales1 - OLE-DB".GOSALES1.dbo.COUNTRY COUNTRY
where
  (COUNTRY.SALESCOUNTRYCODE = CONVERSIONRATE.COUNTRYCODE)
```

Minimized SQL works best when the returned result sets of each query item are equivalent. If there are records in one column that do not correspond to records in another column, the result of the minimized query produces additional rows. You can avoid this by setting the **SQL Generation Type** to **As View**.

For example, if there are Product Types that are not used by any of the Products and these Product Types all have a common Product Line, a Product Line is reported for which there are Product Types, but for which there are no related Products.

Steps

1. Click the query subject that you want to change.
2. From the **Actions** menu, click **Edit Definition**, and then click the **Query Information** tab.
The **Test Results** box is initially empty until you run the query.
3. Click **Options**, and then click the **SQL Settings** tab.
4. Set **Generate SQL** to **As View** or **Minimized**.
5. Click **OK**.
6. If you want to see the SQL, click **Test Sample**.
7. If you want to see the actual query, click **Query**.
8. If you want to see the xml that IBM Cognos BI uses, click **Response**.
9. Click **OK**.

Model Query Subjects and SQL Types

A model query subject that is based on another model query subject may use the logic of the parent query subject instead of its own logic. If the child model query subject uses the **Minimized** SQL type, it does not use the logic of the parent. If the child model query subject uses the **As View** SQL type, it uses the logic of the parent.

For example, you create a model query subject named Returned Products, which shows all return reasons for all products. When you run Returned Products, you see a list of over 700 items. You

then create another model query subject based on Returned Products that is named Return Reasons. This model query subject contains only the Return Reason query item. If the SQL type is set to **Minimized**, the Return Reasons query subject shows five return reasons when it is run. If the SQL type is set to **As View**, the Return Reasons query subject uses the logic of the Returned Products query subject and shows over 700 items.

Dimensions

Information about dimensions based on SAP BW metadata appears in a different topic ([p. 204](#)).

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 with in IBM® Cognos® Framework Manager are regular dimensions and measure dimensions. In SAP BW, measure dimensions are called key figures.

For example, in a project for sales analysis, you include these dimensions:

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

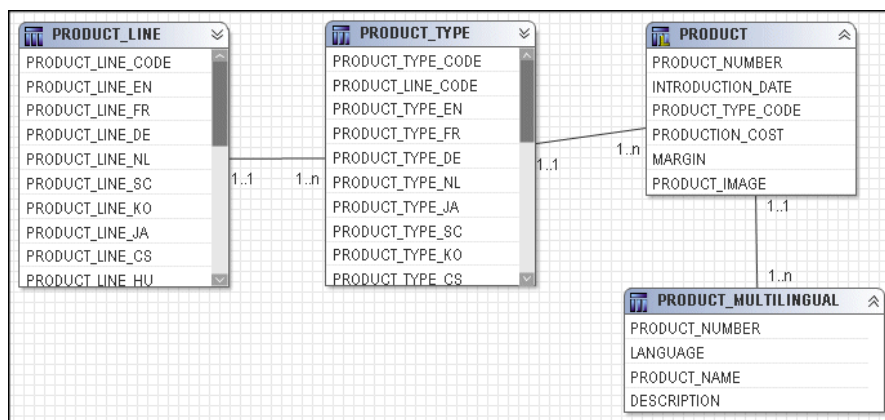
You must use regular and measure dimensions to enable analysis on your relational data source. In most data sources, measure dimensions are likely to be shared by more than one regular dimension. Regular dimensions are often called shared dimensions. A measure dimension and regular dimensions organized into a cluster is often referred to as a star schema group but can also be referred to as a functional or subject area group.

You may also be interested in this topic, "[Query Subjects vs. Dimensions](#)" ([p. 332](#)).

Normalized Data Sources

Normalized or snowflaked data sources often have several tables that describe a single business concept. For example, a normalized representation of Product may include four tables related by 1 . . n relationships. Each Product Line has one or more Product Types. Each Product Type has one

or more Products. Products have names and descriptions in multiple languages so they exist in the Product Multilingual lookup table.



You may be interested in the following related topics:

- creating regular dimensions ([p. 115](#))
- creating measure dimensions ([p. 124](#))

Create a Regular Dimension

Information about modifying a regular dimension for SAP BW metadata appears in a different topic ([p. 205](#)).

A regular dimension contains descriptive and business key information and organizes the information in a hierarchy, from the highest level of granularity to the lowest. It usually has multiple levels and each level requires a key and a caption. If you do not have a single key for your level, it is recommended that you create one in a calculation.

Model regular dimensions are based on data source or model query subjects that are already defined in the model. You must define a business key and a string type caption for each level. When you verify the model, the absence of business keys and caption information is detected. Instead of joining model regular dimensions to measure dimensions, create joins on the underlying query subjects and create a scope relationship between the regular dimension and the measure dimension.

When creating regular dimensions, you must understand the dimensionality of the data. You must be able to answer the following questions:

- What are the levels in your dimension?
- What is the order and combination of levels that form hierarchies?
- What are the relationships between the levels?
- What uniquely identifies a level?
- Which data elements are associated at each level?
- Do you have more than one level of granularity, such as some data is recorded monthly and some is recorded daily?
- Are foreign keys defined in the data source?

You can specify multiple hierarchies on regular dimensions in IBM® Cognos® Framework Manager. Multiple hierarchies for a regular dimension behave as views of the same query. However, you can use only one hierarchy at a time in a query. For example, you cannot use one hierarchy in the rows of a crosstab report and another hierarchy from the same dimension in the columns. If you need both hierarchies in the same report, you must create two dimensions, one for each hierarchy. For more information, see ["Modeling Dimensions with Multiple Hierarchies" \(p. 346\)](#).

In addition to creating regular dimensions, you can also merge dimensions into a single dimension or convert query subjects to dimensions.

Multiple-fact querying is enabled with conformed dimensions.

While you can use data source dimensions, they have limited functionality in comparison to query subjects or model dimensions. It is recommended that you discontinue using data source dimensions (both regular and measure). Create new models following the recommendations in ["Guidelines for Modeling Metadata" \(p. 319\)](#) to use query subjects as the relational foundation of the model. Define regular and measure dimensions as model objects based on data source query subjects or model query subjects or both. Guidance on migration for existing users of data source dimensions will be provided in a future release.

Steps

1. Select a namespace or folder where you want to place the dimension.
2. From the **Actions** menu, click **Create, Regular Dimension**, and then click the **Dimension** tab.
3. Click **Add Hierarchy** and then drag one or more objects from the **Available items** box to the **Hierarchies** box.

You can define multiple hierarchies for a dimension. The first hierarchy is used as the default, or primary, hierarchy.

You can also create an alternate hierarchy by copying a level. Click a level and drag it to the right border of the dimension. You can only copy a level within the same dimension.

4. Click **Add Level** and then drag one or more objects from the **Available items** box into the new level.

You can also create copies of levels in the **Dimension Definition** dialog box or in the **Dimension Map** tab. Click the level and drag it to another position in the hierarchy. All attributes of the level are also copied. You can only copy a level within the same dimension.

5. If you want to use a different item in a level, drag it from the **Available items** box to the **Select a level in the hierarchy control to see the query items** box.

You are prompted to specify its role.

By default, Framework Manager adds the name of the namespace.

Tip: To have a multiple-part key such as first name plus last name, create a new attribute that combines the items, and then specify that the new attribute is the business key.

6. If you want to indicate that the keys of the levels above the current level are not necessary to identify the members in this level, select the item and select the **Unique Level** check box. This indicates that key values belonging to the level should be considered unique regardless of context.

In some circumstances, the **Unique Level** setting is used by IBM Cognos software to optimize SQL queries.

Note: The **Unique Level** check box does not affect the generation and handling of MUN identifiers for the members in this level. All MUNs are fully-qualified.

7. Choose the additional tasks that you want to perform:

- Specify roles ([p. 122](#)).
- Embed calculations by clicking **Add** and then defining the expression ([p. 155](#)).


To change a calculation that has been embedded in the dimension, in the **Dimension Map** tab, click **Attributes**, right-click the query item, and click **Edit Expression**.

- Embed filters ([p. 158](#)).
- Specify the sort characteristics of levels ([p. 121](#)).
- Test the dimension ([p. 128](#)).
- Edit the SQL and change various options ([p. 109](#)).

8. Click **OK**.

9. To change the default hierarchy for a dimension with multiple hierarchies, do the following:

- In the **Properties** pane, click the ellipsis (...) button in the **Default Hierarchy** box.
- Select a different hierarchy, and click **OK**.

You can also use the **Dimension Map** tab to create a regular dimension. Click the **regular dimension** button .

Hierarchies for a Regular Dimension

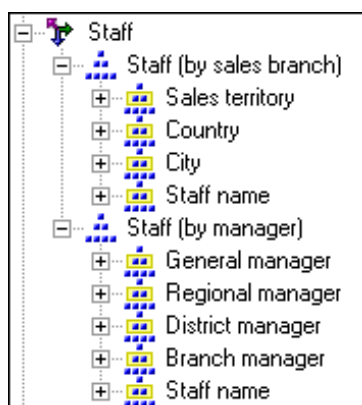
Information about hierarchies for SAP BW metadata appears in different topics ([p. 205](#)) and ([p. 199](#)).

A hierarchy is an ordered list of levels or a collection of items. Each query item in a hierarchy must have a unique name.

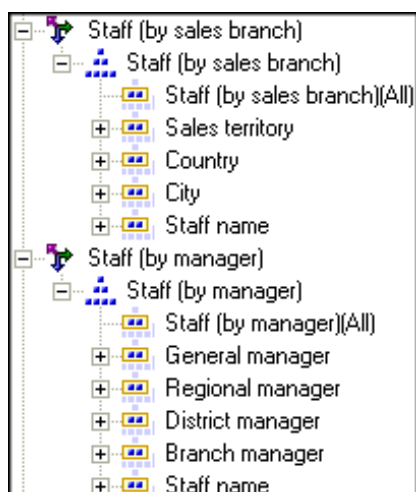
You can specify multiple hierarchies on regular dimensions in IBM® Cognos® Framework Manager. Multiple hierarchies for a regular dimension behave as views of the same query. The first hierarchy is the primary or default hierarchy.

You can use only one hierarchy at a time in a query. For example, you cannot use one hierarchy in the rows of a crosstab report and another hierarchy from the same dimension in the columns. If you need both hierarchies in the same report, you must create two dimensions, one for each hierarchy. For more information, see "[Modeling Dimensions with Multiple Hierarchies](#)" ([p. 346](#)).

For example, sales staff can be viewed by manager or by sales branch and can be modeled as a single dimension with two hierarchies.



If you need both hierarchies in the same report query, such as on opposing axes, you must create a regular dimension for each hierarchy. For example, here is sales staff as two dimensions.



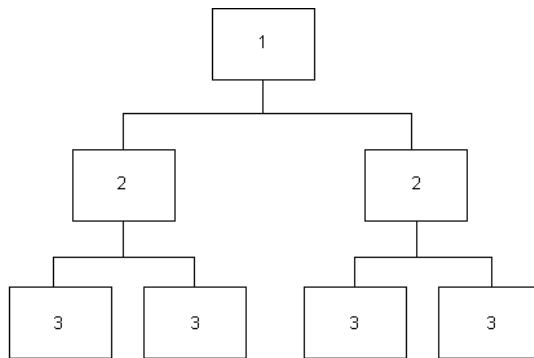
Tip: To change the default hierarchy for a dimension with multiple hierarchies, in the **Properties** pane, click the ellipsis (...) button in the **Default Hierarchy** box, and select a different hierarchy.

If a hierarchy in a dimension contains a large number of members, running a query in one of the IBM Cognos studios may be slow because the IBM Cognos engine is generating one large query for a locally-built cube. To resolve this issue, set the **Wide Member Tree** property in the **Properties** pane to **true**. The engine will then generate multiple smaller queries for the locally-built cube.

Balanced Hierarchy

Each path in a balanced hierarchy descends to the same depth.

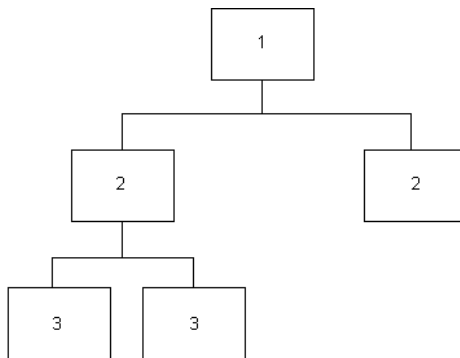
For example, in the following diagram, the highest level is Product Line(Level 1); Level 2 is Product Type; Level 3 is Products.



Unbalanced Hierarchy

The branches in an unbalanced hierarchy descend to different levels.

For example, in the following diagram, the highest level in an organization is the CEO (Level 1); Level 2 is the vice-presidents and the CEO's executive assistant. The executive assistant does not have subordinates, unlike the vice-presidents.



An unbalanced hierarchy can also be ragged. In a ragged-unbalanced hierarchy, there are gaps in the levels and the levels descend to different depths.

Ragged and Network Hierarchies

For relational metadata, we recommend that ragged hierarchies and network hierarchies be flattened in the data source.

Levels for a Regular Dimension

The simplest definition of a level consists of a business key and a caption, each of these referring to one query item. An instance (or row) of a level is defined as a member of that level. It is identified by a member unique name, which contains the values of the business keys of the current and higher levels. For example, `[gosales].[Products].[ProductsOrg].[Product]->[All Products].[1].[1].[2]` identifies a member that is on the fourth level, `Product`, of the hierarchy `ProductsOrg` of the dimension `[Products]` that is in the namespace `[gosales]`. The caption for this product is `TrailChef Canteen`, which is the name shown in the metadata tree and on the report.

The first level of the hierarchy is automatically defined as the All level. It contains a single root member, which represents the top level of the hierarchy. For example, the All level for the Time dimension is named `Time (All)`. You cannot delete or move the All level. You can change its name, description, and screen tip.

If you do not specify the levels of the hierarchy correctly, incorrect aggregation could occur.

Member Unique Names

The member unique name (MUN) is how the member is found in the data source, much like using business keys to find records in a table.

The member unique name is used in the expression for a member data item that is used in a report, a reference to members in filters and expressions, and used in drill-through between OLAP data sources. The member keys in the MUN for the different OLAP data sources must match.

If a member unique name changes, members that are directly referenced in expressions, filters, or reports are no longer found because the MUN is contained in the report specification. Member unique names can change for a variety of reasons:

- Changes to the hierarchy and level structures may change the level unique name.
- The business key values have changed and this changed the member key path.
- The application changed during design or over time.
- The cube has category codes that are unpredictably unique.
- The production environment has more members than the test environment.
- The member no longer exists in the data source.

To avoid these problems, we recommend the following practices:

- Use unique codes and keys within a dimension for the member keys.
- Use unique conformed values for similar dimensions between the target and source environments when enabling drill through.
- When using a dimensionally modeled relational model for drill-through, ensure the root business key conforms with the root members of the other data sources.
- Ensure that the business keys and dimension metadata structure are the same between the production and test environments.
- Do not change the business keys in IBM® Cognos® Framework Manager after going into production.
- Resolve the non-unique keys within a dimension in the data source. Tildes are not recommended in the category codes.
- If you have a Transformer cube that is built with non-unique source values, do not use the Clean House feature in Transformer because it will most likely change the category codes. We recommend that you keep a backup copy of your MDL file.

Keys for Levels

A key is a query item that uniquely identifies members in a level. For example, Product Number uniquely identifies a product while City, State, and Country are all needed to uniquely identify a

city. The key may or may not be contained in a level. Foreign keys are used to relate the measure dimension to its regular dimensions.

Each level needs an item that is defined as a key.

If a model dimension contains a query item whose data type is `BLOB`, create a query subject that has determinants and then create a model dimension that is based on the model query subject.

Sort Members of a Level

For dimensionally modeled relational metadata, you can specify sort characteristics on a dimension. You can also specify sorting on individual levels within the dimension. When you sort individual levels within a dimension, you can ensure that the order in which the data is delivered to the model is appropriate. The order in which the data is delivered to the model can be particularly important when reporting data over relative time periods.

The default sort order is alphabetical, in ascending order, depending on the level caption. When you specify a sort order, the data is sorted on another column such as the business key.

Steps

1. In the **Project Viewer** pane, select a dimension or level.
2. From the **Actions** menu, click **Edit Definition**.
3. Click the **Member Sort** tab.
4. Select the sorting option to apply.

Sorting Option	Description
Metadata (as shown in the member tree)	Used only to specify how members are sorted when displayed in the metadata tree. The sort of the metadata tree cannot be changed by report authors.
Data - Only as Default Report Sort	Used as the default data sort for members in a report. If no data sort is specified, data is retrieved in the order that it was entered in the database. Report authors can override this value and apply a different sort to the members displayed in a report.

Sorting Option	Description
Data - Always (OLAP compatible)	<p>Used to provide member relative functions (MRFs) with a consistent order of the members. This setting should only be used if MRFs are required. Otherwise, the sorting results in unnecessary overhead. Report authors cannot change the order of members as delivered to the MRFs. However, authors can apply a different sort to the members displayed in the report.</p> <p>If no sort is specified and MRFs are used, the report author will receive an error when MRFs are processed.</p> <p>When this option is set, the members of the level are also sorted in the metadata tree even if the Metadata option is not selected. The sort of the metadata tree cannot be changed by the report author.</p>

Tips

- To apply the default sort order to all child levels within the levels in the dimension that do not have a sort option defined, click **Detect**.
 - To remove sort options from all child levels within the levels in the dimension, click **Clear All**.
5. In the **Select a Level to Assign Sort Properties** box, click a level.
 6. In the **Available Data Items** box, click an item to sort and click the right arrow to add it to the **Level Sort Properties** box. Use the up and down arrows to change the order of items.
Tip: To change the sort order to ascending or descending, click **Sort Order**.
 7. To specify how null values are sorted in reports, click the **Nulls** box beside the item and then click **First**, **Last**, or **Unspecified**.
First places the null values at the beginning, and Last places the null values at the bottom. Unspecified uses the setting defined in the data source.
 8. Click **OK**.

Roles

Information about roles for SAP BW metadata appears in a different topic ([p. 210](#)).

Roles define what appears in the member tree in the IBM Cognos studios. Use roles to organize and manage metadata and to determine how to present data to your users.

You can also create expressions that refer to roles instead of query items. You must use the `roleValue` function to refer to a particular role. For example, you want to query against a specific role in a hierarchy but the query item playing that role is different at each level of the hierarchy. A single query can span the different query items at each level. You can also use the `roleValue` function when you know the role but not the underlying query item.

You can assign multiple roles to one query item, but the same role cannot be assigned to different query items in the same level.

Default roles are pre-defined for all parent-child hierarchies and for all levels in level-based hierarchies. Most of these roles are not visible in the IBM Cognos studios.

The roles that are reserved by IBM Cognos BI start with an underscore. The name for a custom role cannot start with an underscore.

Default Roles

The default roles include the following:

- `_businessKey`

Represents the key for the level. The level can be defined as unique if the business key of the level is sufficient to identify each set of data for a level.

The `_businessKey` role can be assigned to only one attribute in a level.

The **Root Business Key** property shows the value of the business key for the root member. The root member is an artificial level created for dimensionally modeled relational models. To enable drill-through on conforming dimensions, you must set the **Root Business Key** property.

- `_memberCaption`

Presents the caption for a member that will be shown in the IBM Cognos studios.

The `_memberCaption` role is necessary to leverage member functions and to enable dragging and dropping levels in the IBM Cognos studios.

Ensure that the data type is set to string for the item that will be assigned the `_memberCaption` role.

- `_memberDescription`

Returns the description for a member within a dimension.

Custom Roles

By default, attributes are included with no role. You can assign attributes to existing roles or you can create custom roles. Each role that you create must have a unique name.

You can translate the custom roles in the model.

Specify Roles

Roles define what appears in the member tree in the IBM Cognos studios. Use roles to organize and manage metadata and to determine how to present data to your users.

Steps

1. Click the dimension whose roles you want to define.
2. From the **Actions** menu, click **Edit Definition**.
3. Click the **Dimension** tab.
4. In the **Hierarchies** box, click the level you want.

5. In the **Select a level in the hierarchy control to see the query items** box, click a query item.
6. Under **Role**, click the ellipsis (...) button.
7. Do one of the following:
 - To use a role defined by Framework Manager, click the **Default Roles** tab, and select a role.
 - To create a role, click the **Custom Roles** tab, and click **Add**.
8. Click **Close**.
9. Click **OK**.

You can also use the **Dimension Map** tab to define roles. Click **Attributes**, right-click the query item, and click **Edit Roles**.

Create a Measure Dimension

Information about modifying a key figures dimension for SAP BW metadata appears in a different topic ([p. 212](#)).

A measure dimension is a collection of facts. You can create a measure dimension for one or more query subjects that have a valid relationship between them.

Model measure dimensions should be composed of only quantitative items. Because, by design, model measure dimensions do not contain keys on which to join, it is not possible to create joins to model measure dimensions. Instead of joining model measure dimensions to regular dimensions, create joins on the underlying query subjects. Then either manually create a scope relationship between them or detect scope if both dimensions are in the same namespace.

Only measures are visible in the model measure dimension. Query items, such as keys, are hidden.

While you can use data source dimensions, they have limited functionality in comparison to query subjects or model dimensions. It is recommended that you discontinue using data source dimensions (both regular and measure). Create new models following the recommendations in "[Guidelines for Modeling Metadata](#)" ([p. 319](#)) to use query subjects as the relational foundation of the model. Define regular and measure dimensions as model objects based on data source query subjects or model query subjects or both. Guidance on migration for existing users of data source dimensions will be provided in a future release.

You can add value by embedding calculations based on existing business rules, such as Profit Margin.

You can change the order of measures, query items, and calculations.

Constraints

If the measure dimension contains a folder, you can change the order only in the **Project Viewer**.

You cannot define hierarchies or levels for a measure dimension.

Steps


1. Click a namespace where you want to place the measure dimension.

- From the **Actions** menu, click **Create, Measure Dimension**.
- Click the **Measure Dimension** tab.
- Drag measures from the **Model Objects** box to the **Measures** box.
- Perform the actions that you want.

Goal	Action
Embed a calculation	Click Add . You can also right-click a measure and click Add or Edit .
Embed a filter	Click the Filters tab.
Test the measure dimension	Click the Test tab or the Query Information tab.
Convert a measure into a query item	Right-click the measure and click Convert to Query Item .

Note: If you test the measure dimension by using the **Query Information** tab, IBM Cognos BI validates the measure dimension. If you test the measure dimension by using the **Test** tab, IBM Cognos BI executes the measure dimension. The SQL for validate is slightly different than the SQL for execute. To generate definitive SQL for the measure dimension, use the **Test** tab.

- Click **OK**.

You can also use the **Dimension Map** tab to create a measure dimension. Click the **measure dimension** button .

You may be interested in the following related topics:

- multiple measure dimensions that are related to dimensions at different levels ([p. 326](#))
- embedded calculations ([p. 155](#))
- embedded filters ([p. 158](#))
- testing the measure dimension ([p. 128](#))

Convert a Measure into a Query Item

If you have created a measure dimension and want to join it to regular dimensions, you need to create joins. Joins need keys and keys are query items, not measures. The measure that you want to use as a key must be converted into a query item.

You can also convert a query item into a measure ([p. 154](#)).

Steps

- Double-click the measure dimension that contains the measure.

2. Click the **Measure Dimension** tab.
3. Right-click the measure and click **Convert to Query Item**.
4. Click **OK**.

Scope Relationships

Scope relationships are necessary to define which dimensions and measures are used together for dimensionally modeled relational models.

Scope relationships exist only between measure dimensions and regular dimensions to define the level at which the measures are available for reporting. They are not the same as joins and do not impact the `Where` clause. There are no conditions or criteria set in a scope relationship to govern how a query is formed, it specifies only if a dimension can be queried with a specified fact. The absence of a scope relationship results in an error at runtime.

If you set the scope relationship for the measure dimension, the same settings apply to all measures in the measure dimension. If data is reported at a different level for the measures in the measure dimension, you can set scope on a measure. You can specify the lowest level that the data can be reported on.

When you create a measure dimension, IBM® Cognos® Framework Manager creates a scope relationship between the measure dimension and each existing regular dimension. Framework Manager looks for a join path between the measure dimension and the regular dimensions, starting with the lowest level of detail. If there are many join paths available, the scope relationship that Framework Manager creates may not be the one that you intended. In this case, you must edit the scope relationship.

A scope relationship is automatically generated when you drag a dimension into the dimension map or when you move a query subject into the dimension namespace and convert it to a regular dimension.


Note: Shortcuts to scope relationships are not supported.

Define Scope Relationships


Scope relationships exist only between measure dimensions and regular dimensions to define the level at which the measures are available for reporting.


Steps


1. Click the **Dimension Map** tab.

Tip: To view scope relationships highlighted with a background color, click the **show scope** button .

2. Click one or more measure dimensions.
3. Click the level of the dimension that you want to set the scope to.

Tip: If you want Framework Manager to define the scope relationship, select the measure dimension and the regular dimension, and click the **determine scope** button .

4. Click the **set scope** button .

If you want to remove the scope, select the hierarchy or dimension and click the **remove scope** button .

If you select a hierarchy, you can remove the scope from a specific hierarchy without affecting the scope set in other hierarchies of the dimension.

If you select the dimension, all scope from all hierarchies is removed. The scope relationship between the measure dimension and the regular dimension is also removed.

Create a Regular Dimension Based on Existing Objects

You can create a new regular dimension by merging existing objects. These objects can be dimensions, query subjects, or query items.

Steps

1. Select the objects that you want in a dimension.
2. From the **Actions** menu, click **Merge in New Regular Dimension**.

View Related Objects

Information about viewing related objects for SAP BW dimensions appears in a different topic ([p. 213](#)).




You can explore a visual representation of the objects that are connected to the query subject or dimension that you select in the **Project Viewer**. The **Context Explorer** shows the objects that the selected object is connected to. You can also select a connected object and see its references.

You can hide an object in the **Context Explorer**. You can also change the layout, fit all objects in the **Context Explorer**, zoom in and out, print, preview diagrams before printing, and change the page setup.

You can also use the **Dimension Map** tab to explore dimensions.

Steps

1. Select one or more objects that you want to explore.
2. From the **Tools** menu, click **Launch Context Explorer**.
3. To see the connected objects, click one or more objects and click the appropriate button.

Goal	Button
View the objects that are related to the selected object.	
View the immediate references for the objects.	
View all references for the objects.	

4. If you want to see details about an object, such as its relationships and query items, right-click the object, click **Navigate Diagram**, **Diagram Settings**, and then select the details you want.

Test a Dimension

Information about testing SAP BW metadata appears in a different topic ([p. 213](#)).

Testing a regular dimension returns the attributes associated with the hierarchy defined as the default.

You can see the results that an object returns by testing it. You can test when creating an object or later on. The objects you can test are dimensions, query subjects, query sets, hierarchies, levels, calculations, and query items.

You can view the data that will appear in a specific report before publishing a package by selecting and testing the objects that will appear in the report. This makes it easier to debug a model and to verify that the model meets the reporting requirements because you do not need to create and publish packages first.

When you test an object, IBM® Cognos® Framework Manager returns sample data. Formatting is not applied to the sample data. If you must test formatting, you must publish the package and view the objects in the IBM Cognos studios.

You may see different results depending on what you test. For example, if you use the expression editor to test a calculation that is embedded in a query subject, Framework Manager tests only the expression, not the item, so the aggregation setting for the query item is not applied to the test. Testing the entire query subject, which includes the calculation, gives a different result because the aggregation setting is applied. For example, if the aggregation setting is summarize, you can see a smaller number of rows in the test.

If you test a child segment of a segmented model, you may see an error if an object you are testing refers to an object in another child segment and the referenced object is not available to the project you are in. We recommend that you check that the parent model contains all the objects and that this error message does not appear when you test the parent model.

Governor settings may affect the testing results. For more information, see "[Set Governors](#)" ([p. 304](#)).

You can change existing test settings to customize the results that the test shows. For example, in addition to other settings, you can control the number of rows returned.

Steps When Creating or Modifying the Object

1. Select the object you want to test.
2. From the **Actions** menu, click **Edit Definition**, and click the **Test** or **Query Information** tab.
The **Test Results** box is initially empty until you run the query.
Any result sets that contain binary large objects are shown as [blob].
3. To run the query and bring back all the test results, click **Test Sample**.
4. If you want to add a count of the rows, click **Total Rows**.

5. If you want to apply the **Regular Aggregate** property of the query item or the **Aggregate Rules** property of a semi-additive measure that is referenced in the expression, select the **Auto Sum** check box.

If you clear this check box, a row is returned for each row in the result set of the query.

6. If you want to obtain more information about the query results, click the **Query Information** tab.
7. Click **OK**.

Steps to View the Data That Will Appear in a Specific Report

1. Select the objects that will appear in the report.
2. From the **Tools** menu, click **Test**.
3. To run the query and bring back all the test results, click **Test Sample**.
4. To view details about any problem that is found, click the **Query Information** tab.

If you do not see the results of the query in the test window, the data from your data source may exceed the value of one of the governors. The query stops at the specified limit, but the test result window does not contain any data. **Tip:** Set each governor to zero.

When you test a measure dimension, the SQL uses aggregates not the measures.

Change the Test Settings

You can customize the tests by changing the test settings.

Steps

1. Select the object that you want.
2. From the **Actions** menu, click **Edit Definition** and then click the **Test** tab or the **Query Information** tab.
3. Click **Options** and then click the **Test Settings** tab.
4. Choose the options that you want.

Goal	Action	Persistence
Retrieve all data and show a specified number of rows	<p>Select the Restrict the maximum number of rows to be returned check box and type the required number of rows.</p> <p>This setting does not improve performance for retrieving data when testing dimensions, query subjects, and query sets.</p>	<p>This setting applies to all dimensions, query subjects, and query sets in the model.</p> <p>This setting is saved and used in your next session with any model.</p>

Goal	Action	Persistence
Specify the level of detail	Drag the Level of Information shown in Results Information slider to the location that represents the amount of detail you require.	This setting is saved and used in your next session with this model.
Temporarily override session parameters	In the Session Parameters box, click Set . The Session Parameters dialog box appears.	The override values are not saved with the model. This setting is for your current session only.
Apply relevant design mode filters	Select the Apply all relevant design mode filters when testing check box. This applies all relevant filters whose usage is set to design mode in another dimension, query subject, or query set.	This setting is saved and used in your next session with any model.
Apply a security filter	In the Security Filters box, click Edit .	This setting is saved and used in your next session with this model.
Change the prompt values	In The Current Prompt Values box, click Prompts . The Model Prompts Manager dialog box appears, which shows all prompts, and their values, that are in the model.	The prompt values are not saved with the model. This setting is for your current session only.

5. Click **OK** two times.

You may be interested in the following related topics:

- working with dimensions ([p. 114](#))
- working with query subjects ([p. 85](#))

Convert a Regular Dimension into a Query Subject

You can convert a regular dimension into a model query subject or a data source query subject.

You can also convert a query subject into a dimension ([p. 105](#)).

Constraint

If a dimension has multiple hierarchies, only the default hierarchy is included when you convert the dimension to a query subject.

Steps

1. Click the regular dimension.
2. From the **Actions** menu, click **Convert to Query Subject**.

Multilingual Metadata

For models that are published in multiple languages, you can view and modify model objects in the different languages.

We recommend that you handle multilingual support in the import view for a variety of reasons. You can reduce the number of query items contained in each dimension and query subject. With fewer dimensions, query subjects, and query items, the model is more manageable. You can simplify maintenance by doing all multilingual work in one place instead of in different business views. This ensures consistency because the languages are set up correctly for all modelers to use. This is particularly important for segmented models.

To support multilingual metadata, do the following:

- ❑ Import metadata from multilingual data sources ([p. 133](#)).
- ❑ Define the languages the model supports ([p. 134](#)).
- ❑ Define one or more parameter maps that translate the locale used when the report is run into the language values in the data source ([p. 163](#)).
- ❑ Use a macro to dynamically substitute language values from the language lookup table using the runLocale session parameter as the key ([p. 168](#)).
- ❑ Export multilingual properties in translation tables, which translators use to enter the correct text for each language ([p. 135](#)).
- ❑ Import the table that contains the translated property values ([p. 136](#)).
- ❑ Publish the metadata in the languages you specify ([p. 251](#)).

For information about how to enable multilingual modeling, see "[Example - Create a Multilingual Project for Relational Metadata](#)" ([p. 136](#)).

Setting Up a Multilingual Reporting Environment

You can create reports that show data in more than one language and use different regional settings. This means that you can create a single report that can be used by report consumers anywhere in the world.

The samples databases provided with IBM® Cognos® 8 store a selection of text fields, such as names and descriptions, in more than 25 languages to demonstrate a multilingual reporting environment.

For information about how data is stored in the samples databases and how the samples databases are set up to use multilingual data, see the *Administration and Security Guide*.

Here is the process for creating a multilingual reporting environment:

- ❑ Use multilingual metadata.

The data source administrator can store multilingual data in either individual tables, rows, or columns.

For more information about configuring your database for multilingual reporting, see the *Administration and Security Guide*.

- ❑ Create a multilingual model.

Modelers use Framework Manager to add multilingual metadata to the model from any data source type except OLAP. They add multilingual metadata by defining which languages the model supports, translating text strings in the model for things such as object names and descriptions, and defining which languages are exported in each package. If the data source contains multilingual data, modelers can define queries that retrieve data in the default language for the report user.

For more information, see the Framework Manager *User Guide*.

- ❑ Create multilingual maps.

Administrators and modelers use a Microsoft® Windows® operating system utility named Map Manager to import maps and update labels for maps in Report Studio. For map features such as country and city names, administrators and modelers can define alternative names to provide multilingual versions of text that appears on the map.

For more information, see the Map Manager *Installation and User Guide*.

- ❑ Create a multilingual report.

The report author uses Report Studio to create a report that can be viewed in different languages. For example, you can specify that text, such as the title, appears in German when the report is opened by a German user. You can also add translations for text objects, and create other language-dependent objects.

For more information, see the Report Studio *User Guide*.

- ❑ Specify the language in which a report is viewed.

You can use IBM Cognos Connection to do the following:

- Define multilingual properties, such as a name, screen tip, and description, for each entry in the portal.
- Specify the default language to be used when a report is run.
Tip: You can specify the default language on the run options page, in the report properties, or in your preferences.
- Specify a language, other than the default, to be used when a report is run.

For more information, see the IBM Cognos Connection *User Guide*.

The data then appears in the language and with the regional settings specified in

- the user's Web browser options
- the run options
- the IBM Cognos Connection preferences

Any text that users or authors add appears in the language in which they typed it.

Modeling with Multilingual Data Sources

To enable a project to work with multiple languages, you must set up data sources to support multiple languages.

Multilingual Relational Data Sources

For relational data sources, you can support multiple languages by using one or more of the following:

- Language-specific database tables

The data source should contain the same tables for each supported language. For example, if the Product table supports English, French, and German, the data source has tables named Product_en, Product_fr, and Product_de.

- Language-specific columns

A database table should contain the same columns for each supported language. For example, if the Product table supports English, French, and German, the table has columns for ProductName_en, ProductName_fr, and ProductName_de.

- Language-specific rows

A database table should contain an additional column to identify the language of each row of data, such as a column named LANG.

These solutions can make the multilingual data sources large and difficult to manage.

You can model a single relational query subject to represent all possible data source languages by using parameter maps and session parameters in the query subject definition. For more information, see ["Creating Prompts with Query Macros"](#) (p. 168) and ["Multilingual Metadata"](#) (p. 131).

Note: Expression syntax is specific to the design language of the model. If you import objects from a model designed in another language, you may have to adjust the expression syntax.

Multilingual SAP BW Data Sources

For SAP BW metadata, you do not need to use parameters to support multilingual reporting. Since SAP BW automatically provides data in the language that matches the logon settings for the current user. If there is no metadata for the current user's language, Framework Manager retrieves data in the default language.

Using a Macro to Model Multilingual Data

You can model multilingual data that is stored in multiple tables, columns, or rows for each supported language. You can use macros with parameter maps and session parameters to create dimensions or query subjects that retrieve data in the preferred language of the person viewing the report.

The location of a parameter in the query subject definition depends on the location of multilingual data in the data source. You must have a parameter map in the macro.

Data source location	Parameter location	Example
A column with a language key in another column	Select list	<pre>Select PRODUCT_TYPE.PRODUCT_TYPE_CODE, PRODUCT_TYPE.PRODUCT_LINE_CODE, PRODUCT_TYPE.PRODUCT_TYPE_ # \$Language_lookup { \$runLocale } # as Product_type from [gosales].PRODUCT_TYPE PRODUCT_TYPE</pre>
Rows whose language is identified by a special column, such as LANG	Filter	<pre>Select PRODUCT.PRODUCT_NAME, PRODUCT_MULTILINGUAL. PRODUCT_NUMBER from [gosales].PRODUCT, [gosales].PRODUCT_MULTILINGUAL Where PRODUCT.PRODUCT_NUMBER = PRODUCT_MULTILINGUAL. PRODUCT_NUMBER and (PRODUCT_MULTILINGUAL."LANGUAGE" = # sq (\$Language_lookup { \$runLocale }) #</pre>

Add a Language to a Project

You can add a language to a project at any time. For example, you do this if the values for a language were not translated earlier.

When you add a language to a project, IBM® Cognos® Framework Manager generates a new property value for every multilingual property of each object in the project. A multilingual property is any text property that appears in a report, such as **Name**, **Description**, and **Screen Tip**.

The new values that Framework Manager assigns to these text properties are a combination of the original property value preceded by the language code. For example, if a dimension is named Country, and you add the Dutch language, Framework Manager inserts a name whose value is (nl)Country.

Each project contains two types of language definitions:

- design language
This is the language in which the model was originally created. This value is stored in the model and cannot be changed. It serves as the default language value.

- active language

This is the language in which model content is currently shown. When you open a model, the active language is set to the language in the model that most closely matches the region and language settings of the computer. You can change this value at any time for your current session only. In future sessions, the model continues to open in the design language.

Steps

1. From the **Project** menu, click **Languages, Define Languages**.
2. In the **Available languages** box, select each language you want to add and click the arrow button to move it to the **Project languages** box.

Tip: To remove a language, select it in the **Project languages** box and click the arrow button to move it to the **Available languages** box.

3. If you want to change the active language, in the **Project languages** box, click a language and click **Set as Active**.
4. Click **OK**.
At the prompt, accept the changes you made to the project.
5. Click **OK**.
6. If you want to view multilingual property values in the **Properties** pane, click the **Languages** tab.

Export a Translation Table

You can generate and export a translation table to simplify the task of translating model objects. The translation table contains a list of all the text strings defined for multilingual properties, such as **Name**, **Description**, and **Screen Tip**. Translators can then use an external application, such as Microsoft® Excel, to type the required information in the table.

You can export a translation table as either a comma-separated value file (.csv) or Unicode text file (.txt). You must export the translation table as a Unicode text file if it either contains a non-Latin language or will be imported by a computer with a language setting that is different from your own computer.

Steps

1. Select the objects you want to export.
2. From the **Project** menu, click **Languages, Export Translation File**.
3. In the **Project Languages** box, click the languages you want to export, and click the arrow button to move them into the **Languages to be exported** box.

You must export the design language of the model that will use the translation table. For example, if the translation table will be used in a model that uses French as the design language, you must export French.

IBM® Cognos® Framework Manager exports the appropriate locale code for each language you select. If you do not select all the languages to be translated, you must manually enter the language codes in the first row of each new language column in the translation table.

4. In the **Model objects to be exported** box, select whether you want to export all model objects, or export only preselected objects and their children.
5. Enter the location and name of the translation table.
6. Click **OK**.

Import a Translation Table

You can add text property values for each language defined in your model by importing translated data from a file. The imported file must be a translation table that was used by translators to enter the required translated values.

The translation table must contain the design language of the model that will use the translation table. The translation table can contain a subset of the languages defined for the project.

Steps

1. From the **Project** menu, click **Languages, Import Translation File**.
2. In the **Project Languages** box, click the languages in the translation table, and click the arrow buttons to move them to the **Translate from** and **Translate into** box.

You must select the design language for this model.

3. In the **Apply translation to** box, select whether you want to apply the translation to all model objects, or only to preselected objects and their children.
4. Enter the location and name of the translation file.
5. Click **OK**.

Example - Create a Multilingual Project for Relational Metadata

You want to create a model that can be used by English, French, and German report authors. You also want the IBM Cognos studios to automatically show metadata in the language required by the report author.

In the `go_data_warehouse` sample, you need to do the following:

- Translate the metadata.
- Use macros to create a multilingual project.

You can modify the dimensions and query subjects to show multilingual content by using the `Language_lookup` parameter map and the `runLocale` session parameter.

Steps

1. Open the `go_data_warehouse` project.

2. Ensure that English, French, and German are supported languages:
 - From the **Project** menu, click **Languages, Define Languages**.
 - Ensure that the **Project languages** pane contains English, French, and German.
 - In the **Project Viewer** pane, click a query item and, in the **Properties** pane, click the **Languages** tab.
For the name, description, and tool tip text, you see one entry for each language.
3. Export all the languages and objects in the project to a comma-separated value file (.csv) named GOSLDW-ML.csv.
 - From the **Project** menu, click **Languages, Export Translation File**.
 - In the **Project Languages** box, Ctrl+click **English, French, and German**, and click the top arrow to move them to the **Languages to be exported** box.
 - In the **Export languages to this file** box, enter the location of GOSLDW-ML.csv.
4. Open the GOSLDW-ML.csv file in Microsoft® Excel, and translate the strings.
Note that each column represents a given language, and the file contains only the text strings that exist in the model.
5. In Framework Manager, import the translated file:
 - From the **Project** menu, click **Languages, Import Translation File**.
 - In the **Project Languages** box, move **French and German** into the **Translate into** box.
 - In the **Import translation table from this file** box, enter the location of GOSLDW-ML.csv.
6. In the **Project Viewer**, double-click the **Language_lookup** parameter map.
Note that the keys match the possible values for the runLocale session parameter, which identifies the language of the current user. These keys are mapped to the language values that are defined in the go_data_warehouse database.
7. Ensure that the parameter map contains the following information.

Key	Value
de	DE
en	EN
fr	FR

8. Select an object that contains multilingual columns, such as the Order method dimension, and, from the **Actions** menu, click **Edit Definition**.

The language identifier that was appended to the query item name is replaced by a parameter map and the `runLocale` session parameter:

```
Select
  ORDER_METHOD_DIMENSION.ORDER_METHOD_KEY,
  ORDER_METHOD_DIMENSION.ORDER_METHOD_CODE,
  ORDER_METHOD_DIMENSION.ORDER_METHOD_# $Language_lookup{$runLocale}
# AS ORDER_METHOD

from

  [go_data_warehouse].ORDER_METHOD_DIMENSION
```

9. To preview the results, click the **Test** tab and then click **Test Sample**.
10. From the **Project** menu, click **Session Parameters** and change the `runLocale` session parameter value to `fr`.
11. Test the Order method dimension again to view the results.

Query Items

Information about query items for SAP BW metadata appears in a different topic ([p. 219](#)).

A query item is the smallest object in the model that can be placed in a report. It represents a single instance of something, such as the date that a product was introduced.

For relational metadata, you can modify the properties of query items by

- setting **Usage** and **Regular Aggregate** properties to reflect the intended use of the query item ([p. 141](#))
- formatting query items to control how data appears in a report ([p. 148](#))
- identifying a column as a prompt, and controlling how your users see the prompt information ([p. 149](#))

You can also modify the properties for multiple query items at the same time ([p. 34](#)).

Because reports can contain different query items from one or more objects in the model, query item properties control many aspects of the final report. When you create a model dimension or model query subject, the query items inherit the properties of the data source query items on which they are based.

The properties for query items or measures include the following.

Query item property	Description
Name	The name of the query item or measure.
Description	A description of the query item or measure.
Last Changed	The date that the query item or measure was last changed. The property is automatically updated with the current date time.

Query item property	Description
Last Changed By	The user who last changed the query item or measure. This property is automatically updated when the item is changed. The value is the current logon username.
Model Comments	Used to add internal comments about the model. The information is used on the Analyze Publish Impact dialog and in the Model Report . Comments are not accessible to package users.
Screen Tip	A description that can appear in the published package for your users.
Expression	Used to create embedded calculations that provide your users with calculated values that they regularly use. This property is for measures only. Note: The Expression property is not used by SAP BW.
External Name	The name that appears in the data source.
Is Hidden	Whether to hide or show the query item or measure in the published package. Even when Is Hidden is set to True and the query item or measure is invisible to your users, it is always present in the published package because the query item or measure may be needed by other objects in the model. You do not see the query item or measure in the Package Publish wizard. For example, a calculation may make use of a hidden query item.
Usage	The intended use for the data represented by the query item. This property is for query items only.
Format	How information appears in a report.
Currency	Which currency is used. This property cannot be changed in the Property pane. Use the Format property to change the currency.
Data Type	The data type that was set in the data source. Because this property is set in the data source, it is read-only in Framework Manager.

Query item property	Description
Precision	<p>The total number of digits.</p> <p>Because this property is set in the data source, it is read-only in Framework Manager.</p>
Scale	<p>How many digits are represented in the scale.</p> <p>For example, you can show numbers in thousands so that 100,000 means 100,000,000.</p> <p>Because this property is set in the data source, it is read-only in Framework Manager.</p>
Size	<p>The size of the query item or measure.</p> <p>Because this property is set in the data source, it is read-only in Framework Manager.</p>
Is Nullable	<p>Whether the query item or measure can contain a null value.</p> <p>Because this property is set in the data source, it is read-only in Framework Manager.</p>
Display Type	<p>How the query item is shown.</p> <p>The column value can appear in the IBM Cognos studios as a picture, as a link, or as a value.</p> <p>The default is value.</p> <p>This property is for query items only.</p>
MIME Type	<p>The format that the column value uses.</p> <p>For example, if Display Type is set to picture, MIME Type could be jpeg.</p> <p>This property is for query items only.</p> <p>Note: The MIME Type property is not used by SAP BW.</p>
Prompt Info	<p>Prompt behavior.</p>
Regular Aggregate	<p>The type of aggregation that is associated with the query item, measure, or calculation in the published package.</p>
Aggregate Rules	<p>For dimensionally modeled relational metadata, the rules for semi-additive aggregation.</p> <p>For SAP BW metadata, the Aggregate Rules property is read-only.</p>

Query item property	Description
Allocation Rule	Specifies the type of allocation defined for the measure. A value of default specifies that constant allocation is used in list queries and once-only allocation is used in crosstab queries. A value of constant specifies that constant allocation is used in all queries.
Is Unsortable	Whether the values of this query item can be sorted. This property is for query items that contain large objects such as BLOBs.

You can rename a query item in the **Calculation Definition** dialog box. Renaming the query item updates references to this query item.

You may be interested in the following related topics:

- the **Usage**, **Regular Aggregate**, and **Aggregate Rules** properties ([p. 141](#))
- formatting the query item ([p. 148](#))
- prompts ([p. 149](#))
- creating a model report([p. 272](#))

Modifying How Query Items Are Aggregated

Information about aggregation of SAP BW query items appears in a different topic ([p. 222](#)).

You can change how some query items and measures are aggregated in reports. IBM® Cognos® Framework Manager applies aggregate rules when your users create a report that summarizes a query item or measure.

When you import metadata, Framework Manager assigns values to the **Usage** and **Regular Aggregate** properties for query items and measures depending on the type of object that the query item or measure is in. The **Usage** property identifies the intended use for the data represented by the query item ([p. 144](#)). The **Regular Aggregate** property identifies the type of aggregation that is applied to the query item or measure ([p. 145](#)). Your users can override the values of the **Regular Aggregate** property. For semi-additive measures, you can specify additional aggregate rules by modifying the **Aggregate Rules** property ([p. 146](#)).

When modifying the **Regular Aggregate** property, you can select values that are not available through importing, such as average and maximum. You must understand what the data represents to know which aggregate rule is required. For example, if you aggregate a part number, the only aggregate values that apply are count, count distinct, maximum, and minimum.

Rules for Setting Properties for Dimensions

IBM® Cognos® Framework Manager uses the following rules to set the **Usage** and **Regular Aggregate** properties.

Object	Usage property	Regular Aggregate property
Query item in a regular dimension	Attribute	Unsupported
Query item in a measure dimension	Identifier	Count
Measure in a measure dimension	Fact	Automatic if the measure is a calculation Sum if the measure is not a calculation

If the measure is semi-additive, use the **Aggregate Rules** property to define rules for semi-additive aggregation ([p. 146](#)).

Rules for Setting Properties for Query Subjects

You can change the **Usage** and **Regular Aggregate** properties for all types of query subjects. The settings for these properties are based on characteristics such as data type and participation in keys and relationships.

For model query subjects, IBM® Cognos® Framework Manager uses the settings of the underlying query subjects. If the source query subject does not use these properties, the rules for data source and stored procedure query subjects are applied.

For data source or stored procedure query subjects, Framework Manager uses the following rules to set the **Usage** and **Regular Aggregate** properties when importing the query subjects.

Object	Usage property	Regular Aggregate property
Query item is part of a key in a determinant	Identifier	Count
Query item participates in a relationship	Identifier	Count
Query item is data type date or time	Identifier	Count
Query item is data type numeric or time interval	Fact	Automatic if the item is a calculation Sum if the item is not a calculation
None of the above applies	Attribute	Unsupported

Rules for Setting Properties for Calculations

IBM® Cognos® Framework Manager uses a number of rules for setting the **Usage** and **Regular Aggregate** properties for calculations.

Rules for Interpreting Calculated Aggregations

The calculated aggregation type is supported only for the following:

- stand-alone calculations
- calculations that are embedded within measure dimensions and are based on measures from the same measure dimension

The calculated aggregation type is not supported for calculations that are embedded within query subjects.

Framework Manager uses the following rules to interpret the calculated aggregation type in the **Regular Aggregate** property:

- Standard aggregation functions (average, count, maximum, minimum, standard deviation, sum, variance) and references to model query subjects are aggregated first. The remaining operations are then applied to the aggregation result.

For example, to divide debt by credit for each row, the SQL looks like this:

```
Select
customer, debt, credit, debt/credit as
percent_debt from x
```

To aggregate for all customers, the SQL looks like this:

```
Select sum(debt), sum(credit), sum(debt)/sum(credit)
as percent_debt from (Select customer, debt, credit from x)
```

- If the query item in the calculation is a fact and the aggregation type for the query item is average, count, maximum, minimum, or sum, the aggregation type of the query item is used.
- If the query item in the calculation has no aggregation type set, the aggregation type minimum is applied in the query. It is not possible in SQL to have an aggregation setting of none.
- Aggregate functions are interpreted as if they are applied to a value in a single row when these functions are used in the detail context. For example, a Report Studio report has the Auto Group and Summarize property set to false.
- Aggregation of a query item is based on the aggregated expression derived from the item definition.

For example, you want to total this stand-alone calculation:

```
[namespace].[Company].[debt]
/ [namespace].[Company].[credit]
```

The calculation is aggregated with this expression:

```
Total([namespace].[Company].[debt]) / Total([namespace].[Company].[credit])
```

- Scalar aggregates, also known as running, ranking, and moving aggregates, are calculated for report granularity unless the **For** clause is explicitly specified.
- Granularity of aggregate functions is set by grouping for determinants or by keys of corresponding levels in the cube.

For example, `Rank([namespace].[Company].[debt])` is interpreted as `Rank([namespace].[Company].[debt] for Report)`.

Rules to Determine the Automatic Aggregation Type

For stand-alone and embedded calculations, IBM® Cognos® Framework Manager uses one of these rules to determine the aggregation type.

Calculation	Aggregation type
Is based on an expression containing an aggregate function such as average, maximum, minimum, or sum	Calculated
Has an if-then-else operation and the if condition references fact items	Calculated
References a calculation using any type except unsupported	Calculated
Has an aggregation type other than unsupported	Calculated
Is based on an expression that references a model query subject whose usage is set to fact and whose aggregation type is set to average, count, maximum, minimum, or sum but the query item expression does not use an aggregate function	Summarize See below for the rules for summarize.
None of these rules apply	Unsupported

Rules to Determine the Summarize Aggregation Type

For stand-alone and embedded calculations, IBM® Cognos® Framework Manager uses one of these rules to determine the aggregation type.

Calculation	Aggregation type
Is a fact containing of only a reference to a query item whose aggregation type is average, count, maximum, minimum, or sum	Uses the aggregation type of the query item
Numeric or an interval type	Sum
Time, datetime, or date type	Maximum
None of these rules apply	Count

Usage Property

The **Usage** property identifies the intended use for the data represented by each query item. During importing, the **Usage** property is set according to the type of data that the query items represent in the data source.

You need to verify that this property is set correctly. For example, if you import a numeric column that participates in a relationship, the **Usage** property is set to **identifier**. You can change the property.

For relational query items, the value of the **Usage** property depends on the type of database object the query item is based on.

Usage property	Database object	Description
Identifier	key, index, date, datetime	Represents a column that is used to group or summarize the data in a Fact column with which it has a relationship. Also represents an indexed column. Also represents a column that is of the date or time type.
Fact	numeric, timeinterval	Represents a column that contains numeric data that can be grouped or summarized, such as Product Cost.
Attribute	string	Represents a column that is neither an Identifier or Fact, such as Description.

Regular Aggregate Property

The **Regular Aggregate** property identifies the type of aggregation for the query item or calculation when you publish it. Your users can either use this default setting to perform calculations on groups of data, or apply a different type of aggregation.

For example, if the **Regular Aggregate** property value of the Quantity query item is sum, and it is grouped by Product Name in a report, the Quantity column in the report shows the total quantity of each product.

The following aggregation types are supported for relational data sources:

- automatic
- average
- calculated
- count
- count distinct
- maximum
- minimum
- sum

Define Aggregate Rules for Semi-Additive Measures

For measures of dimensionally modeled relational metadata, you can define an aggregate rule for each related dimension. These rules are in addition to the **Regular Aggregate** property and are used to specify how semi-additive measures are to be aggregated with respect to information from that dimension. A semi-additive measure is one that is to be summed for some dimensions, but should not be summed across some other dimensions. For the dimensions over which the measure is not additive, a different aggregation rule must be specified.

If a measure expression contains an aggregation function, the aggregation rule for that measure is ignored.

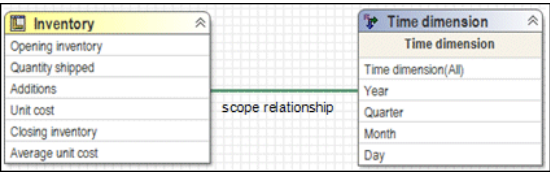
Aggregate rules are applied in this order:

- the **Regular Aggregate** property is applied to dimensions that are included in the query but are not referenced in the **Aggregate Rules** dialog box
- the aggregation that is specified in the **Aggregate Rules** dialog box is applied to their specified dimensions, in the order that you specified the rules
- the report-level aggregation that is specified in the query

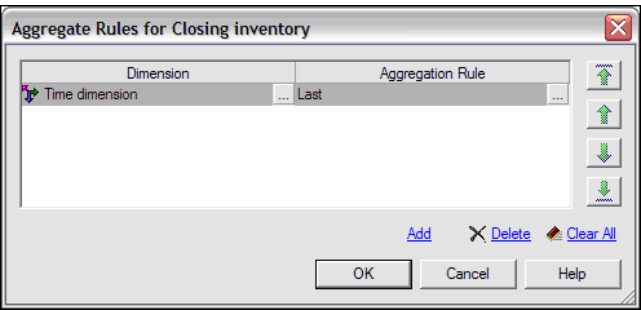
For example, inventory is recorded in the data warehouse with opening and closing balances at the month grain. If you need to report on inventory for different periods of time, you apply an aggregate that takes the value from the appropriate time slice within the period. For an opening balance, the value is the last balance of the previous month. For a closing balance, the value is the last balance of the current month. In this example, the inventory measure has a **Regular Aggregate** property of total and an aggregate rule for the time dimension with a value of last.

You can have only one aggregate rule for each dimension.

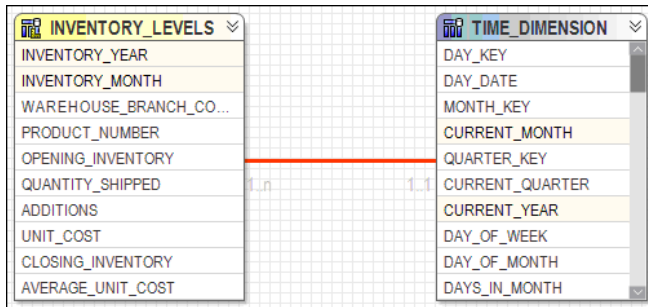
Note: There must be a direct relationship between the underlying query subjects from which the measure and dimension are constructed. For example, consider the Inventory measure dimension and the Time dimension:



You want to define an aggregate rule for Closing inventory that generates the last closing inventory recorded for any given period of time:



For IBM Cognos to apply the aggregate rule of last on Closing inventory based on the Time dimension, there must be a direct relationship between the underlying query subjects used to construct the Time dimension and the Closing inventory measure. This is in addition to the scope relationship defined between the Inventory measure dimension and the Time dimension:



If a relationship does not exist, running a query that contains the measure Closing inventory generates the following error:

RQP-DEF-0564 Unable to determine fact grain of semi-additive dimension '[Dimensional view].[Time dimension]' for measure '[Dimensional view].[Inventory].[Closing inventory]'. Verify that a relationship is defined between the underlying query subjects from which the dimensions are constructed.

Semi-additive measures are not supported for sparse data. Sparse data is multidimensional data in which many of the combinations of dimension values do not have related fact data.

For SAP BW metadata, the **Aggregate Rules** property value is read-only. The **Semi-Aggregate** property is used instead.

Steps

1. Click the measure you want to work with.
2. In the **Properties** pane, click the ellipsis (...) button in the **Aggregate Rules** property.
3. To add a dimension for this measure, click **Add** and select the dimension.

You can include a shortcut to a dimension if there is a scope relationship between the measure dimension and the regular dimension.

4. To specify the aggregate operation, click the ellipsis (...) button in the **Aggregation Rules** box.

The following operations are supported:

- sum
- minimum
- maximum
- average
- first
- last

5. If you want to change the order of the aggregate rules, use the arrow buttons.

6. Click **OK**.

After you set an aggregate rule, the icon for the measure changes.

Semi-Aggregate Property

For relational metadata, the **Semi-Aggregate** property value is set to unsupported and is read-only.

If the value is set to **unsupported** in IBM® Cognos® Framework Manager, the semi-aggregate behavior is ignored in the IBM Cognos studios.

The **Semi-Aggregate** property will not be supported in future releases. Instead, use the **Aggregate Rules** property for semi-additive measures.

Determine Usage and Aggregation Rules

When you use the **Determine Usage** and **Determine Aggregation Rules** commands in IBM® Cognos® Framework Manager, you are resetting the values of the **Usage** and **Regular Aggregate** back to their default values. This overwrites all changes you have made to these properties.

When generating aggregation values, Framework Manager assigns a value that is based on the **Usage** property value and the type of object it is.

Usage property value	Regular Aggregate property value
Identifier	Count
Attribute	Unsupported
Fact	Sum

Steps

1. In the **Project Viewer** pane, click one or more dimensions or query subjects.
2. In the **Properties** pane, click the **Properties** tab.
3. Change the **Usage** property to **unknown**.
4. Change the **Regular Aggregate** property to **unsupported**.
5. From the **Tools** menu, click **Determine Usage**.
6. From the **Tools** menu, click **Determine Aggregation Rules**.

Format Query Items

Information about formatting query items based on SAP BW metadata appears in a different topic ([p. 225](#)).

You can specify how query item values appear in reports. Use the **Format** property to choose a format type, such as text, date, and currency. Each format type contains properties that further specify how the data appears.

For example, you can assign the **Currency** format type to a numeric query item, and then use the **No. of Decimal Places** property in the **Data Format** dialog box to specify how many decimal places appear in reports.

Some characters are language-sensitive and appear properly only when your locale supports the applicable font. For example, for Japanese currency symbols to appear correctly, your locale must be set to Japanese.

If IBM® Cognos® Framework Manager does not show the currency you require, you must ensure that you install the appropriate language packs to support the currency symbols. For example, to have the Indian currency symbol (rupee) appear, you must run an operating system or install a language pack that can show this symbol. The Japanese operating system or Japanese language is one that can show the Indian currency symbol.

You can define properties for several query items at the same time. However, if the query items have different format types, all properties that were previously specified are overridden and the default values from the data source are used. If the original format types of the selected query items are the same, all the properties for the selected query items are set identically.

For example, to use the same decimal separator for two query items and to keep the number of decimals different, each query item must be changed individually. If both are selected and changed at the same time, all properties including the number of decimals are set identically for both query items.

Steps

1. In the **Project Viewer** pane, click the query item you want to format.
2. In the **Properties** tab of the **Properties** pane, click the **Format** property.
3. Select the appropriate **Format type**.
4. In the **Properties** box, select or type the appropriate property value.
5. Click **OK**.

Define a Prompt Control

Information about prompt controls for SAP BW metadata appears in a different topic ([p. 226](#)).

Prompts help your users quickly find the information they need in a report. Prompts are generally defined in reports. However, you can change the behavior of prompts in the studios by modifying the definition of dimensions or query subjects in the model.

This is useful for query items, such as ProductTypeCode, whose values are not shown in a report but are useful for filtering data. In general, it is better to define type-in prompts in the reports to make use of the additional prompt features. However, your users cannot modify some variables. For these variables, you can use IBM® Cognos® Framework Manager instead of the reports to define type-in prompts.

The Prompt Info properties set in Framework Manager give you the ability to control default filtering and prompting. The properties are used by:

- Query Studio to create a filter expression and set the use and display items in a prompt and prompt query
- the Build Prompt Page tool in Report Studio to create a filter expression and set the use and display items in a prompt and prompt query
- generated prompts in Report Studio to set the use and display items in the prompt and prompt query

The syntax for using a prompt as a value is

?<PromptName>?

You can use prompts in

- parameter maps
- session parameters
- stored procedure arguments
- expressions, including filters, calculations, and relationships

For example, a stored procedure returns all rows with a specific product number. Instead of using the product number as the argument for the stored procedure, you can use a prompt, such as

?Product_Number?.

For examples of prompts, see ["Creating Prompts with Query Macros" \(p. 168\)](#) and ["Example - Use Prompts with a Stored Procedure" \(p. 92\)](#).

For more information about creating prompts, see the Report Studio *User Guide*.

Steps

1. Click the query item.
2. In the **Properties** pane, click the **Properties** tab.
3. Click the plus sign (+) next to the **Prompt Info** property.
This is a compound query item property.
4. Modify the following properties to reflect the behavior you require.

Goal	Property
Set the type of prompt control that is generated when the report is run.	Prompt Type
Set the generated prompt as part of a series of generated cascading prompts.	Cascade On Item Reference

Goal	Property
Specifies which query item is displayed to the report user in the prompt.	Display Item Reference
The values in the prompt are data values of the query item.	
Each value in the prompt is associated with a value in the query item specified in the Use Item Reference property.	
Specifies which query item is passed from the prompt to the filter.	Use Item Reference
Each value is associated with a value in the query item specified in the Display Item Reference property.	
Specifies which query item is used in the filter expression to retrieve data.	Filter Item Reference

Prompt Type Property

The **Prompt Type** property sets the type of prompt control that is generated when the report is run, such as an edit box or a pull-down list.

The default value for this property is **Server Determined**.

Note: Prompt types set on attributes are now processed. The report user will see the prompt that matches the prompt type on the attribute. Because prompt types on attributes were not processed in the previous release, some differences may occur.

Value	Prompt Control
Server Determined	The type of prompt control is based on information in the server, such as the data type of the query item.
Edit Box	A simple text box. If the data type of the column is date or dateTime, this value generates a date or date-time control as well as the text box.
Select Date	A date control with a calendar interface.
Select Date/Time	A date-time control with a calendar interface. For SAP BW metadata, this value is not relevant.
Select Interval	A date-time interval control. For SAP BW metadata, this value is not relevant.

Value	Prompt Control
Select Time	<p>A time control that filters data based on the selected time period. For example, if you define a Select Time prompt for Order Time, the user can use the time control to show all orders placed after 1:00, or all the orders placed between 10:00 and 11:00.</p> <p>If you are referring to a time member, you must use the exact values only.</p> <p>If you are using a range, the end points of the range must correspond to values in the data source.</p>
Select Value	A drop-down list.
Select with Search	<p>A list control so that users can search for values.</p> <p>For SAP BW metadata, this value is not relevant.</p>
Select with Tree	A tree prompt control for prompts that are based on a hierarchy node.

Cascade On Item Reference Property

The **Cascade On Item Reference** property indicates that the generated prompt is part of a series of generated cascading prompts. The query item that you reference in this property is the parent item in the cascade. The system prompts the user for the cascade item before prompting them for the current query item.

For example, if you want to prompt for Product Line and then Product within the selected line, set the **Cascade On Item Reference** property of the Product query item to Product Line.

Display Item Reference and Use Item Reference Properties

The **Display Item Reference** property specifies which query item is displayed to the user in the prompt. The **Use Item Reference** property specifies which query item is passed from the prompt to the filter. Each value in the list of display items is associated with a value of the query item specified in the **Use Item Reference** property.

For example, you want the prompt to display Country Name while using Country Code to retrieve data. Set the **Display Item Reference** property to Country Name and the **Use Item Reference** property to Country Code. The prompt for Country Name makes it easy for the report user to select required values. However, using the Country Code in the filter is more efficient for data retrieval.

These properties are used by

- Query Studio to create a filter expression and set the use and display items in a prompt and prompt query
- the Build Prompt Page tool in Report Studio to set the use and display items in a prompt and prompt query

- generated prompts in Report Studio to set the use and display items in the prompt and prompt query

Note: The values of the **Use Item Reference** and **Filter Item Reference** properties must be compatible. Otherwise, the report user may receive unexpected results. For more information, see the **Filter Item Reference** property (p. 153).

Default: If no values are set, the properties default to the name of the query item.

These properties are used only for data driven prompt controls whose **Prompt Type** property is set to either **Select Value** or **Select with Search**.

Filter Item Reference Property

The **Filter Item Reference** property identifies the query item used when Report Studio or Query Studio generates a filter. This property can help create more efficient queries by ensuring that a filter uses an indexed numeric column rather than a non-indexed string column.

For example, a report author wants to create a filter for the Country Name query item. You set the **Filter Item Reference** property to use Country Code instead of Country Name for any filter that uses the Country Name query item.

In another example, a report author wants to create a filter for the Country Code query item that appears in the Orders table. You want that filter to use the Country Code in the Country table because there are fewer rows to read in the Country table so you set the **Filter Item Reference** in the model to `Country.Country Code`.

This property is used by:

- Query Studio to create a filter expression
- the Build Prompt Page tool in Report Studio to create a filter expression

Default: If no value is set, the property defaults to the name of the query item.

Using the Filter Item Reference and Use Item Reference Properties

The values of the **Filter Item Reference** and **Use Item Reference** properties must be compatible. The value of the **Use Item Reference** property must be a type that is expected by the **Filter Item Reference** property. Otherwise, the report user may receive unexpected results. This may occur when a report user creates a filter without creating a prompt page.

In an example model, the **Use Item Reference** property is set to Employee Number and the **Filter Item Reference** property is Employee Name. In Report Studio, a report author creates the following filter without creating a prompt page:

```
Reference.EmployeeName in ?parm1?
```

Report Studio automatically generates prompts when you create a filter without creating a prompt page. Because the prompt is generated, Report Studio uses the Prompt Info properties from the Employee Name query item in the Framework Manager model.

The **Use Item Reference** indicates that the values being passed to the filter are employee numbers. The **Filter Item Reference** is filtering data based on Employee Name. The filter is as follows:

```
Reference].[Employee Name] in ("1", "2").
```

Since there are no Employee Name values of "1" or "2", the report will be blank.

Using Filter Item Reference for Dimensionally Modeled Relational Metadata

For dimensionally modeled relational metadata, **Prompt Info** is specified on the attribute with the role of `_memberCaption`, instead of the level. Although set on the attribute, the **Prompt Info** properties are processed as if they were on the level. By default, when the level is included in a report, users are prompted to enter MUNs in the level's prompt. To enter caption values instead, set the attribute's **Filter Item Reference** property to itself. When the prompted filter is applied, the filtered values will be based on the attribute values.

For example, the level Product Line has an attribute of Product Line with a role of `_memberCaption`. If the **Filter Item Reference** property value is set to Product Line, report users are prompted to enter Product Line values. If the **Filter Item Reference** property value is left blank, users are prompted to enter MUNs.

Note: Do not use the **Filter Item Reference** property with the Select with Tree prompt type. Because a Select with Tree prompt can only filter on a level or hierarchy, setting the **Filter Item Reference** property will cause an error.

Testing a Prompt

When you test a model object that references a prompt, IBM® Cognos® Framework Manager asks you to enter the prompt value. Framework Manager uses this value for either the duration of the session, or until you clear the prompt value.

You can change the session value of prompt values through the **Options** dialog box. This dialog box is available when you modify a dimension or query subject, or define a calculation, filter, query set, or complex relationship. You can change the prompt value at the time that you are testing the expression that references that value.

If you select the **Always prompt for values when testing** check box in the **Prompt** dialog box, Framework Manager prompts you for a value every time you test the object. When updating the object or performing a count, Framework Manager uses the existing prompt value, if one exists.

A prompt on a query item in a model query subject is associated only with that query item. A prompt on a query item in a data source query subject is associated with the entire query subject and therefore, the prompt appears when you test any query item in the query subject.

Convert a Query Item into a Measure

You can convert a query item in a measure dimension back into a measure.

You can also convert a measure to a query item ([p. 125](#)).

Steps

1. Double-click the measure dimension that contains the query item.
2. Click the **Measure Dimension** tab.
3. Right-click the query item and click **Convert to Measure**.
4. Click **OK**.

Adding Business Rules

Information about business rules for SAP BW metadata appears in a different topic ([p. 237](#)).

You can add business rules to the dimensions and query subjects in your model to refine the data retrieved and ensure that the correct information is available for your users.

Creating business rules and storing them in the model instead of in reports has many advantages. You save time because you and your users do not have to re-create the business rules every time they are needed. The business rules ensure consistency because your users all use the same definitions. For example, Low Margin means the same thing throughout the organization. They are easy to update because you maintain the business rules centrally so that all reports are updated automatically as the rules evolve. For example, if the definition for Low Margin changes, all reports that use the Low Margin calculation are updated automatically. The business rules enhance security. For example, you can use filters to limit the data that your users can see.

You can

- ❑ add calculations so that your users can include calculated data in their reports ([p. 155](#)).
- ❑ create and apply filters so that you limit the data that a query subject retrieves ([p. 158](#)).
- ❑ add prompts that will automatically appear whenever a dimension or query subject is used in a report; report consumers are then prompted to filter data ([p. 149](#)).
- ❑ use session parameters ([p. 165](#)) and parameter maps ([p. 163](#)) in macros ([p. 168](#)) to dynamically resolve expressions.
- ❑ create a security filter to control the data that is shown to your users when they set up their reports ([p. 257](#)).

Create a Calculation

Information about calculations for SAP BW metadata appears in a different topic ([p. 237](#)).

You can create calculations to provide your users with calculated values that they regularly use. Calculations can use query items, parameters, variables, calculated members, expressions, and expression components, such as functions.

Punctuation characters, such as the question mark (?), must be in 7-bit ASCII character code. If you type a punctuation character from a multi-byte enabled keyboard, ensure that you type the 7-bit ASCII representation of the character. For example, type Alt+063 for the question mark.

Avoid using characters that are used for expression operators in the name of the calculation. Syntax errors may occur when the expression is evaluated. For example, a calculation named Margin * 10 causes errors when used in an expression such as [Margin * 10] < 20.

In expressions, an operator or function may require operands to be of a particular dimensional type. When an operand is not of the required type, one or more coercion rules may be applied to coerce the operand to the appropriate type. Because coercion rules are not applied to expressions in model query subjects, ensure that those expressions are valid without relying on coercion rules. For more information about coercion rules, see the IBM® Cognos® Report Studio *User Guide*.

If you insert an imported user-defined function in the calculation, ensure that the function name does not repeat vendor-specific names. For example, if the user-defined function name is CHAR you will receive an error when testing the function in the **Calculation Definition** dialog box because this name is considered identical as **char** in Microsoft® SQL Server. For information about function names used in your database, see the database product documentation.

At query time, IBM® Cognos® Framework Manager returns a null value for any calculation that contains a divisor whose value is zero. Framework Manager cannot detect zero-division errors in functions such as average and mod, because the division operator is not explicit.

Framework Manager supports stand-alone calculations and embedded calculations.

- Use a stand-alone calculation when you want to reuse the expression.

You can apply a stand-alone calculation to one or more dimensions or query subjects to provide calculated data to a report, or include it in a package to make it available to your users. By moving a stand-alone calculation or a shortcut to it into a folder, you can better organize the model objects.

You cannot use stand-alone calculations in Analysis Studio. Instead, use an embedded calculation.

- Use an embedded calculation when you want to use a calculation with only one dimension or query subject.

You can create an embedded calculation when modifying a relational data source query subject, model query subject, or dimension.

If you start with an embedded calculation, you can later convert it into a stand-alone expression that you can apply to other dimensions or query subjects. **Tip:** Right-click the calculation expression in the **Calculations** tab and click **Convert to Stand-Alone Calculation**.

When you embed a calculation, the data source query subject must have a relationship to any query subject referenced by the expression. This relationship is necessary even if the expression references a model query subject based on the same table as the data source query subject in which you are embedding the expression.

To create a calculation on an unrelated query subject, do one of the following:

- Ensure that there is a join path between the new query subject and the one that contains the calculation.
- Base the embedded calculation on a query item that is based on the data source query subject you want.
- Convert the calculation to a stand-alone calculation, so that it is not part of the query subject.
- Create a stand-alone calculation that references the embedded object.


Steps to Create a Calculation


1. Do one of the following:

- To create a stand-alone calculation, click the namespace or folder and, from the **Actions** menu, click **Create, Calculation**.
- To create an embedded calculation for a measure dimension, double-click the dimension. On the **Measure Dimension** tab, click **Add**.
- To create an embedded calculation for a regular dimension, double-click the dimension. On the **Dimension** tab, select a hierarchy and click **Add**.
- To create an embedded calculation for a data source query subject, double-click the data source query subject. On the **Calculations** tab, click **Add**.
- To create an embedded calculation for a model query subject, double-click the model query subject. On the **Query Subject Definition** tab, click **Add**.

The **Calculation Definition** dialog will appear.

2. In the **Name** box, type a name for the calculation.
3. Define the expression.

Goal	Action
Add items	On the Model tab, click a query item, filter, or calculation and click the arrow.
Add functions	On the Functions tab, choose a component and click the arrow.
Add parameters	On the Parameters tab, click a parameter and click the arrow.
Retrieve all data and show a specified number of rows	<p>Click the options button, select the Restrict the maximum number of rows to be returned check box, and type the required number of rows to be returned.</p>  <p>This setting does not improve performance for retrieving data when testing dimensions, query subjects, and query sets.</p>
Override session parameters	Click the options button, click Set , enter a value in the Override Value field, and click OK .
Override prompt values	<p>Click the options button, and then click Prompts.</p> <p>The Model Prompts Manager dialog box appears, which shows all prompts, and their values, that are in the model.</p>

4. To test the calculation, click the **test** button .

You can test only calculations that contain query items. If a calculation contains a function, for example `_add_days`, the **Test Sample** button is not available.

Tip: If there is an invalid expression in the calculation, review the **Tips** box in the expression editor for more information.

5. Click **OK**.
6. Modify the **Data Type** property to identify the type of data the calculation returns.

The IBM Cognos studio uses this information to format the data that the calculation returns.

You may be interested in the following related topics:

- functions ([p. 381](#))
- overriding session parameters ([p. 165](#))
- testing ([p. 128](#))

Create a Filter

Information about filters for SAP BW metadata appears in a different topic ([p. 239](#)).

A filter is an expression that specifies the conditions that rows or instances must meet to be retrieved for the dimension, query subject, calculation, or report to which the filter is applied. A filter returns a boolean value so that you can limit the rows returned by a dimension or query subject.

For example, you can use the `in_range` function to create a filter that retrieves data for products introduced in a specific time frame. The syntax for this example looks like this:

```
[gosales_goretailers].[Products].[Introduction  
date]  
in_range {Feb 14, 1999 : July 14, 2007}
```

Note: When using a date or time function, you must use a 24-hour clock. IBM® Cognos® Framework Manager does not support "a.m." or "p.m." in expressions. For example, use 20:00 to signify 8 p.m.

You can restrict the data represented by dimensions or query subjects in a project by creating a security filter. The security filter controls the data that your users can see when they set up their reports.

You can also apply governors to restrict the data that the queries in a package retrieve.

Framework Manager supports stand-alone filters and embedded filters.

- Use a stand-alone filter when you want to reuse the expression.

You can add a stand-alone filter to one or more dimensions or query subjects to limit the data that the query retrieves when the filtered dimension or query subject is used in a report, or you can include it in a package to make it available to your users. By moving a stand-alone filter or a shortcut to it into a folder, you can better organize the model objects.

- Use an embedded filter when you want to use a filter with only one dimension or query subject.

You can create an embedded filter when modifying a dimension, relational data source query subject, or model query subject.

If you start with an embedded filter, you can later convert it into a stand-alone expression that you can apply to other dimensions or query subjects. **Tip:** Right-click the filter expression in the **Filters** tab and click **Convert to Stand-alone Filter**.

When you embed a filter, the data source query subject must have a relationship to any query subject referenced by the expression. This relationship is necessary even if the expression references a model query subject based on the same table as the data source query subject in which you are embedding the expression.

To create a filter on an unrelated query subject, do one of the following:


- Ensure that there is a join path between the new query subject and the one that contains the filter.
- Base the embedded filter on a query item that is based on the data source query subject you want.
- Convert the calculation to a stand-alone filter, so that it is not part of the query subject.
- Create a stand-alone filter that references the embedded object.

Steps

1. Do one of the following:
 - If you want to create a stand-alone filter, click the namespace or folder and, from the **Actions** menu, click **Create, Filter**.
 - If you want to create an embedded filter, double-click the dimension or query subject that will contain the filter, click the **Filters** tab, and then click **Add**.
2. In the **Name** box, type a name for the filter.
3. Define the expression.

Tip: If there is an invalid expression in the filter, review the **Tips** box in the expression editor for more information.

Goal	Action
Add query items and filters	On the Model tab, drag the objects you want to the Expression Definition box.
Add functions	On the Functions tab, drag the functions to the Expression Definition box.
Add parameters	On the Parameters tab, drag the parameters to the Expression Definition box.

Goal	Action
Retrieve all data and show a specified number of rows	<p>Click the options button, select the Restrict the maximum number of rows to be returned check box, and type the required number of rows to be returned.</p>  <p>This setting does not improve performance for retrieving data when testing dimensions, query subjects, and query sets.</p>
Override session parameters	Click the options button, click Set , enter a value in the Override Value field, and click OK .
Override prompt values	<p>Click the options button, and then click Prompts.</p> <p>The Model Prompts Manager dialog box appears, which shows all prompts, and their values, that are in the model.</p>

4. Click **OK**.

You can also apply governors to restrict the data that the queries in a package retrieve ([p. 304](#)).

You may be interested in the following related topics:

- security filters ([p. 257](#))
- functions ([p. 381](#))
- parameters ([p. 163](#))
- session parameters ([p. 165](#))
- testing ([p. 101](#))
- mandatory and optional prompts ([p. 171](#))

Apply a Filter

Information about filters for SAP BW metadata appears in a different topic ([p. 242](#)).

To apply a filter, you must modify the dimension, data source query subject, or model query subject. The query subject must either contain the query items that the filter references, or have a relationship path to the query subjects that contain the query items.

You can embed a stand-alone filter in dimensions or query subjects, but if you want a different usage for each embedded filter, you must create different versions of the stand-alone filter. Otherwise, your users could be required to fill in a prompt that you thought was optional if there is any instance

where the usage is set to mandatory. For information about mandatory and optional prompts, see [\(p. 171\)](#).

For example, in query subject A, you embed a stand-alone filter and define it as optional. In query subject B, you define it as mandatory. When your users create a report that uses both query subjects, they are required to choose values in both filters, even the one defined as optional. All instances of the filter are considered to be mandatory when used in the same query. The solution is to create different versions of the filter, each with its own name.

Steps

- 1. Create a filter.
- 2. Select the filter and, from the **Actions** menu, click **Edit Definition**.
- 3. Click the **Filters** tab, and drag the filter you created to the **Filters** box.
- 4. Select a usage value for the filter.

Usage Value	Description
Always	<p>Use this usage value to ensure specified data is filtered out of all reports. For example, your company may have obsolete information that it stores but does not want to report on.</p> <p>Always is the default usage value.</p>
Design Mode Only	<p>Retrieves a small subset of the data for the sample report. Use this usage value when you do not need to see all the data, for example when testing a query subject.</p> <p>To apply design mode filters in Framework Manager, select the Apply all relevant design mode filters when testing option. This option is available on the Test Settings tab.</p> <p>Your users may need the design mode filter in Query Studio when they want to focus on designing the layout and format of a report and not retrieve all the data as they work. To access the design mode filter in Query Studio, run the report with limited data.</p>

Usage Value	Description
Optional	<p>Specifies that the filter is optional. The user is prompted to filter data and can leave the prompt blank. If the prompt is blank, Framework Manager ignores the filter and retrieves all data for the dimension or query subject.</p> <p>The ? ? syntax is required for optional prompts.</p> <p>Use this usage value if your users want to control when the filter is applied. For example, you want to see on one country sometimes and see the data for all countries at other times. An optional filter for country looks like this:</p> <pre>([GeoNamespace].[Countries].[CountryName] = ?WhichCountry?)</pre>

5. If you want to view the SQL, click the **Query Information** tab.
6. Click **OK**.

Example - Show the Currency Name for Each Country

You want to create a query that shows the currency name for a specific country. To do this, you create a filter that returns data for a specific country code, and apply the filter to a model query subject that retrieves the currency name for each country.

Note: The following example uses a relational data source.

Steps

1. Open the go_sales sample model. It is located in *c10_location/webcontent/samples/Models/go_sales/go_sales.cpf*
2. Create a filter to limit the retrieval of data to only those country codes in the conversion rate table whose value is 2:
 - Click the **Filters** folder and, from the **Actions** menu, click **Create, Filter**, and name the new filter **ConversionRateCountryCode**.
 - Click the **Model** tab.
 - In the **Available Components** box, open the **Database view** folder and then open the **GoSales** folder.
 - Add the **Country Code** query item from **Conversion Rate** query subject to the **Expression definition** box, and type `= '2'` at the end of the expression.
 - Click **OK**.
3. Create a model query subject named **ISO Code**.
 - In the **Available Model Objects** box, open the **Database view** folder.

- Add Country query item and the ISO 3-letter code query item from the Country query subject to the **Query Items and Calculations** box.
4. Apply the ConversionRateCountryCode filter:
 - Click the **Filters** tab.
 - Open the Filters folder and drag ConversionRateCountryCode to the **Filters** box.
 5. Click the **Query Information** tab.
The generated SQL contains the filter even though it does not affect the result set.
 6. Change the usage of the ConversionRateCountryCode filter to **Optional**:
 - Click the **Filters** tab.
 - Click the ellipsis (...) button under **Usage** for the ConversionRateCountryCode filter, and click **Optional**.
 7. If you want to view the SQL, click the **Query Information** tab.
 8. Click **OK**.

Create a Parameter Map

Information about parameter maps for SAP BW metadata appears in a different topic ([p. 243](#)).

Use parameters to create conditional query subjects that allow for substitutions when the report is run. 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. You can manually enter the keys and values, import them from a file, or base them on existing query items in the model.

You can also export parameter maps to a file. To modify the parameter map, you can export the map values to a file, do additions or modifications and then import it back into IBM® Cognos® Framework Manager. This is especially useful for manipulating large, complex parameter maps.

All parameter map keys must be unique so that Framework Manager 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.

The value of a parameter can be another parameter. However, you must enclose the entire value in number signs (#). 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 percentage sign (%) instead of a dollar sign (\$).

We recommend that you assign an alias to a query item that uses a parameter map as part of its name and to add the multilingual names to the object in the **Language** tab (**Properties** pane).

Constraint

We recommend that you do not base a parameter map on a query item or table with a large result set, such as 50,000 rows. Each time you use the parameter map in an expression or in SQL,

Framework Manager executes this large query. Performance is then slowed. Parameter maps are recommended for smaller lookup tables.

Steps to Manually Create a Parameter Map

1. Click the **Parameter Maps** folder and, from the **Actions** menu, click **Create, Parameter Map**.
2. In the **Name** box, type a name for the new parameter map.
3. Click **Manually enter the parameter keys, and/or import them from a file** and click **Next**.
4. Do one of the following:
 - To manually enter values, click **New Key**, type a key, and press Tab to enter a value for that key.
 - To import keys and values, click **Import File** and identify the location of the appropriate .csv or .txt file. For a .txt file to be used for import, the values must be separated by tabs and the file must be saved as UTF8 or Unicode format. ANSI text files are not supported.

Note: If you are going to use a parameter in a data source query subject, the value must use English-specific punctuation. This means that you must use a period (.) to represent a decimal and a comma (,) to separate lists of values.

5. Modify existing parameters as required.

Goal	Action
Assign a default value	<p>In the Default Value box, type a value.</p> <p>If the key used in an expression is not mapped, the default value is used.</p> <p>Setting a default value is optional. However, if no default is provided, an unmapped key could produce an error.</p>
Remove a parameter	Select a row and click Delete .
Modify a parameter	Select the row you want to modify, click the Edit button, and type a value.
Clear all keys and values	Click Clear Map .

6. Click **Finish**.

Steps to Base a Parameter Map on Existing Query Items

1. Click the **Parameter Maps** folder and, from the **Actions** menu, click **Create, Parameter Map**.
2. In the **Name** box, type a name for the new parameter map.

3. Click **Base the parameter map on existing Query Items** and click **Next**.
4. Click the query item to use as the key, and then click the query item to use as the value.
Both query items must be from the same query subject.
5. Click **Next**.
6. In the **Default Value** box, type a value.
If the key used in an expression is not mapped, the default value is used.
Setting a default value is optional. However, if no default is provided, an unmapped key could produce an error.
7. Click **Finish**.

You may be interested in the following related topics:

- using parameters ([p. 167](#))
- using a parameter map to specify the language value ([p. 165](#))

Example - Specifying a Language Value for Relational Metadata

An international company stores its product information in English and French. With the use of a parameter map and macros, employees can retrieve data that matches the information they require.

Create a `Language_lookup` parameter map that contains the following:

Key	Value
en	EN
fr	FR

When you examine the SQL for the Product Line query subject, you see the following:

```
Select
    PRODUCT_LINE.PRODUCT_LINE_CODE,
    #'PRODUCT_LINE.PRODUCT_LINE_' + $Language_lookup{$runLocale}#
as Product_Line
from
    [gosales].PRODUCT_LINE PRODUCT_LINE
```

The `runLocale` macro returns a locale setting that is used by the `Language_lookup` macro to specify a language value.

Create a Session Parameter

Information about session parameters for SAP BW metadata appears in a different topic ([p. 245](#)).

A session parameter is a variable that IBM® Cognos® Framework Manager associates with a session. For example, user ID and preferred language are both session parameters. Because session parameters are key and value pairs, you can think of each session parameter as an entry in a parameter map

named Session Parameters. 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 Content Manager. By default, the following session parameters appear 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. For more information, see the section "[Add a Language to a Project](#)" (p. 134).

- **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 IBM 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 see other session parameters, such as `account.personalInfo.email` or `account.personalInfo.surname`.

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, as well as static values.

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

The rules governing the use of parameters include the following:

- All possible return values must have the same data type.

- Only one value can be defined.

Steps

1. From the **Project** menu, click **Session Parameters**.
2. Click **New Key** and type a session parameter key and value.
3. Choose how to handle the override value.
 - To avoid having to set the override value every time you edit the project, set the session parameter as a value.
 - To avoid having to remove the project setting each time before you publish it, set the session parameter as a session override.
4. Modify existing parameters as required.

Goal	Action
Change the parameter value	Click the row that contains the value you want to change, click Edit , and type a value.
Assign a default value	In the Default Value box, type a value. Framework Manager uses the default value if a key has an invalid value.
Remove a parameter	Click a row and click the Delete button. You cannot delete an environment session parameter.
Clear an override value	Click a row and click Clear Override .

5. Click **OK**.

Using Parameters with Relational Data Source Query Subjects

Model objects do not reflect changes to the data source objects on which they are based. Therefore, when you add a parameter to a data source query subject, consider whether you want to create a model object that references the parameter. If so, you must assign an alias to the parameterized object in the data source query subject. This ensures that model query subjects, filters, or calculations that reference the object return the correct results when the parameter value changes.

For example, the following SQL defines a data source query subject that contains a session parameter named `runLocale`. The `runLocale` parameter value specifies which column the query retrieves. The alias behaves like a shortcut so that when a model object references `CountryNameAlias`, IBM® Cognos® Framework Manager retrieves the value to which the alias is assigned.

```
Select
  #${ColumnMap}${runLocale}# as
  CountryNameAlias
```

```
From
[GoSales].Country
```

Creating Prompts with Query Macros

Macros are fragments of code that you can insert anywhere in the `Select` statement that defines a query subject. You can include references to session parameters, parameter maps, and parameter map entries. Parameter values are set when you run the query.

For example, you can use the language session parameter to show only the data that matches the language setting for the current user.

Macros can be used in these different ways:

- They can be inserted in the SQL.
An example is `Select * from Country where Country.Name = #myMap{$runLocale}#`
- They can supply an argument to a stored procedure query subject.
If a value is not hard-coded for the argument, the stored procedure query subject can be used to return different data.
- They can be inserted in expressions, such as calculations and filters.
An example is a filter `[gosales].[Sales staff].[Staff name] = #UserLookUpMap{$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**.
An example using the **Content Manager Datasource** property is `#$DataSourceMap{$UserId}#`
- They can be used as a parameter wizard.
Parameters can reference other parameters. An example is `Map1, Key = en-us, Value = #myMap{$UserId}#`
- They can be used in the **Session Parameter** dialog box.
An example is `MySessionParameter, value = #myMap{$UserGroup}#`

You can replace the following query subject elements with a parameter.

Element	Example
Query items identified in the <code>Select</code> list	<code>#'Product_name_' + \$languageCode#</code>
Tables identified in the <code>From</code> clause	<code>Product_#\$language#</code>
Where clause	<code>Product_lang = #sq(\$languageCode)#</code>

Element	Example
Name of the data source, schema, or source property	<code>#\$data_source#.\$\$schema#.Products</code>

Syntax

Use the following syntax to reference session parameter and parameter values.

Object	Syntax	Example
Session key	<code>\$session_key</code>	<code>#\$my_account#</code>
Parameter map key	<code>\$map{<key>}</code>	<code>#\$map_one{ 'abc' }#</code>
Parameter map entry whose key is defined by a session parameter	<code>\$map{\$session_key}</code>	<code>#\$map_one{\$my_account}#</code>

You can add the following elements to further define the macro expression.

Symbol	Purpose
Single quotation marks <code>'</code>	<p>Delineates a literal string that has a single quotation mark as part of the string.</p> <p>If the single quotation mark appears in a string, such as a query item, the string must be enclosed in a single quotation mark on both sides of the string and the single quotation mark must be doubled. For example, <code>ab'c</code> is written as <code>'ab' 'c'</code></p> <p>If the single quotation mark appears in a macro, the string must be enclosed in square brackets. For example, <code>ab'c</code> is written as <code>[ab'c]</code></p> <p>If the single quotation mark appears in a prompt, there is no need to enclose the string.</p> <p>To escape a single quotation mark in an expression, use <code>&apos;</code></p>
Square brackets <code>[]</code>	Encloses model objects, such as a namespace or query subject and macro names that contain restricted characters, such as a number sign, hyphen, or space.

Symbol	Purpose
Curly brackets, also known as braces { }	<p>Calls a function that is unknown to the parser, such as <code>dateadd</code> in DB2®, and whose first argument is a keyword.</p> <p>Example:</p> <pre>dateadd ({month},2,<date expression>)</pre>
+ operator	Concatenates two strings, such as <code>'abc' + 'xyz'</code>
Single quote function (sq)	<p>Surrounds the result of a string expression with single quotation marks. Existing single quotation marks are double-escaped so that they do not interfere with the quotation structure. You can use this function to build clauses to test against literal parameter-driven values.</p> <p>Here is an example:</p> <pre>#sq(\$my_sp)#</pre> <p>If a session parameter (<code>my_sp</code>) has the value <code>ab'cc</code>, the result is</p> <pre>'ab"cc'</pre>
Double quote function (dq)	<p>Surrounds the result of a string expression with double quotation marks. You can use this function to refer to table and column names with non-identifier characters, such as a blank space or a percent sign (%).</p> <p>Here is an example:</p> <pre>#dq ('Column' + \$runLocale)#</pre> <p>If <code>runLocale=en-us</code>, the result is</p> <pre>"Column en-us"</pre>
Square bracket function (sb)	<p>Inserts a set of square brackets into the argument to build object references in a model query and model expressions, such as filters and calculations.</p> <p>Here is an example:</p> <pre>#sb ('my item in ' + \$runLocale)#</pre> <p>If <code>runLocale=en-us</code>, the result is</p> <pre>[my item in en-us]</pre>

For information about functions, see ["Using the Expression Editor" \(p. 381\)](#).

Create a Macro

Macros are fragments of code that you can insert anywhere in the `SELECT` statement that defines a query subject. You can include references to session parameters, parameter maps, and parameter map entries. Parameter values are set when you run the query.

Constraints

When you reference a parameter, you must do the following:

- use a number sign (#) at the beginning and end of each set of one or more parameters.
Everything between the number signs is treated as a macro expression, which is processed at runtime. Framework Manager removes anything that is outside the number signs.
- precede each parameter map entry with a dollar sign (\$)
- use a name that starts with an alpha character (a..z, A..Z)


Do not insert macros between existing quotation marks or square brackets because IBM® Cognos® Framework Manager does not execute anything within these elements.

Steps

1. Select the data source query subject you want to modify.
2. From the **Actions** menu, click **Edit Definition**.
3. On the **SQL** tab, click **Insert Macro** to start the Macro Editor.
4. In the **Available components** box, click the parameter maps, session parameters, or functions you want to use, and drag them to the **Macro definition** box.

Ensure that you type the macro expression between the number signs. If you enter text before or after the number signs, when you click OK, Framework Manager deletes this text.

5. Insert single or double quotation mark functions.

Tip: Click the arrow next to these buttons  for a menu of choices for placing the quotation marks.

6. If you want to edit a parameter map or session parameter, click it in the **Macro definition** box.

The **Parameter Map** or **Session Parameters** dialog box appears. You can set override values for session parameters, add new items, or change values.

7. Check the macro in the **Information** box.

If a macro is incorrect, an error message appears.

Tip: To clear a macro, click the **clear all** button .

8. Click **OK**.

Mandatory and Optional Prompts

You can create mandatory and optional prompts in IBM® Cognos® Framework Manager models by using query macros. You can use two prompt macro functions, `prompt` and `promptmany`, to

create single value prompts and multiple value prompts. You can insert a prompt macro anywhere in the SQL statement that defines the query subject.

If you want to use a prompt macro in an expression such as a calculation, you must specify the data type when using an overloaded operator, such as a plus sign (+). You can use the plus sign to concatenate two items and to add two items.

If you want to define a filter on a dimension level and have the filter use the `prompt` or `promptmany` macro, you must provide the data type as `memberuniquename` and a default value. For information about applying filters, see [\(p. 160\)](#).

Here is an example:

```
members( [MS_gosales].[New Dimension].[PRODUCTLINE].[PRODUCTLINE]
) in ( set( #promptmany('what', 'memberuniquename', '[MS_gosales].[PROD1].
[PRODUCTLINE].[PRODUCTLINE]->[all].[1] '
) # ) )
```

Here is an example of a mandatory prompt:

```
select
  COUNTRY_MULTILINGUAL.COUNTRY_CODE as COUNTRY_CODE,
  COUNTRY_MULTILINGUAL.COUNTRY as COUNTRY,
  COUNTRY_MULTILINGUAL."LANGUAGE" as LANGUAGE1,
  COUNTRY_MULTILINGUAL.CURRENCY_NAME as CURRENCY_NAME
from
  gosales.gosales.dbo.COUNTRY_MULTILINGUAL COUNTRY_MULTILINGUAL
where COUNTRY_MULTILINGUAL.COUNTRY = #prompt('CountryName') #
```

When default values are specified in the syntax of macro prompts, you may see an error. Use prompt syntax such as `where Country = ?Enter Country?`.

The `prompt` and `promptmany` functions have the following parameters. All argument values must be specified as strings.

Name

This mandatory parameter is the name of the prompt. Name can also refer to the name of a parameter on a user-created prompt page, in which case the user-created prompt page appears when the report is run instead of the default prompt page that the macro would generate.

Datatype

This optional parameter is the prompt value data type. The default value is string. Prompt values are validated. In the case of strings, the provided value is enclosed in single quotation marks and embedded single quotation marks are doubled.

Values include the following:

- boolean
- date
- datetime
- decimal
- double
- float

- int
- integer
- interval
- long
- memberuniquename

Memberuniquename is not an actual data type. This value must be used when the data type of the parameter is member unique name (MUN).

- numeric
- real
- short
- string
- time
- timeinterval
- timestamp
- token

Token is not an actual data type. It is a way to pass SQL. A token does not pass values.

- xsddate
- xsddatetime
- xsddecimal
- xsddouble
- xsdduration
- xsdfloat
- xsdint
- xsdlong
- xsdshort
- xsdstring
- xsdtime

DefaultText

This optional parameter is the text to be used by default. If a value is specified, the prompt is optional.

If you use a space and no values are provided in the **Prompt Value** dialog box, a `Where` clause is usually not generated.

If you use text and no values are provided in the **Prompt Value** dialog box, a `Where` clause is usually generated using the default value.

Ensure that the text you provide results in a valid SQL statement.

Note: If the data type is `memberuniquename`, a value for the `DefaultText` parameter must be provided. For example:

```
(#prompt('WhichLevel', 'memberuniquename', '[goSalesAgain].[PRODUCT1].[PRODUCT].[PRODUCT(All)]->[all]')#)
```

Text

This optional parameter is text that precedes any user-provided values, such as `'and column1 = '`.

QueryItem

This parameter is optional. The prompt engine can take advantage of the **Prompt Info** properties of the query item. Descriptive information can be shown, although the prompt value is a code.

TextFollowing

This optional parameter is the closing parenthesis that is used most often for the `promptmany` function. This parameter is also useful when the prompt is optional and is followed by hardcoded filters in the SQL statement.

Examples - Select a Country Prompt

When a report is run, you want your users to be prompted to choose the country for which they want to see data. The following code examples describe how you can use macros to create different prompts.

Mandatory Prompt with No Data Type Specified

Note the following:

- The `Datatype` argument is not specified. Therefore, it is a string, which is correct in this case.
- The `DefaultText` argument is not specified. Therefore, it is a mandatory prompt.

```
select
  COUNTRY_MULTILINGUAL.COUNTRY_CODE as COUNTRY_CODE,
  COUNTRY_MULTILINGUAL.COUNTRY as COUNTRY,
  COUNTRY_MULTILINGUAL."LANGUAGE" as LANGUAGE1,
  COUNTRY_MULTILINGUAL.CURRENCY_NAME as CURRENCY_NAME
from
  gosales.gosales.dbo.COUNTRY_MULTILINGUAL COUNTRY_MULTILINGUAL
where COUNTRY_MULTILINGUAL.COUNTRY = #prompt('CountryName')#
```

Mandatory Prompt with the Data Type Specified

Note the following:

- This prompt requires a valid integer value as response.

- The `DefaultText` argument is not specified. Therefore, it is a mandatory prompt.

```
select
  COUNTRY_MULTILINGUAL.COUNTRY_CODE as COUNTRY_CODE,
  COUNTRY_MULTILINGUAL.COUNTRY as COUNTRY,
  COUNTRY_MULTILINGUAL."LANGUAGE" as LANGUAGE1,
  COUNTRY_MULTILINGUAL.CURRENCY_NAME as CURRENCY_NAME
from
  gosales.gosales.dbo.COUNTRY_MULTILINGUAL COUNTRY_MULTILINGUAL
where COUNTRY_MULTILINGUAL.COUNTRY_CODE >
  #prompt('Starting CountryCode',
    'integer',
    '',
    '',
    '[gosales].[COUNTRY_MULTILINGUAL].[COUNTRY_CODE]') #
```

Optional Prompt and Mandatory Filter with the Data Type and Default Value Specified

Note the following:

- This prompt allows the user to supply a valid integer response.
- The `DefaultText` argument is specified. Therefore, the user may omit entering a value, in which case the value 10 is used. This makes it an optional prompt, but not an optional filter.

Example 1:

```
select
  COUNTRY_MULTILINGUAL.COUNTRY_CODE as COUNTRY_CODE,
  COUNTRY_MULTILINGUAL.COUNTRY as COUNTRY,
  COUNTRY_MULTILINGUAL."LANGUAGE" as LANGUAGE1,
  COUNTRY_MULTILINGUAL.CURRENCY_NAME as CURRENCY_NAME
from
  gosales.gosales.dbo.COUNTRY_MULTILINGUAL COUNTRY_MULTILINGUAL
where COUNTRY_MULTILINGUAL.COUNTRY_CODE >
  #prompt('Starting CountryCode',
    'integer',
    '10'
  ) #
```

Example 2:

```
[gosales].[COUNTRY].[COUNTRY] = #prompt('countryPrompt','string','''Canada''')#

Result 2:
[gosales].[COUNTRY].[COUNTRY] = 'Canada'
```

Note the following:

- The `defaultText` parameter must be specified such that is literally valid in the context of the macro, because no formatting takes place on this value. See details below.
- The default string `Canada` in Example 2 is specified as a string using single quotes, in which the embedded single quotes are doubled up, thus 3 quotes. This results in the string being properly displayed within single quotes in the expression.
- As a general rule for the string datatype, the `defaultText` should always be specified as in the previous note, except in the context of a stored procedure parameter.
- For the `defaultText` of types `date` or `datetime`, a special format should be used in the context of SQL. Examples of these formats are `'DATE '2001-12-25''` and `'DATETIME '2001-12-25`

12:00:00'''. In all other contexts, you use the `date/datetime` without the keyword and escaped single quotes (e.g., `'2001-12-25'`).

Prompt That Appends Text to the Value

Note the following:

- The `DefaultText` argument is specified as a space character. In this case, the generated text is just the space character, which eliminates the `Where` clause from the query.
- The `Text` argument is specified, which is written into the generated SQL before the user-provided prompt value.

```
select
  COUNTRY_MULTILINGUAL.COUNTRY_CODE as COUNTRY_CODE,
  COUNTRY_MULTILINGUAL.COUNTRY as COUNTRY,
  COUNTRY_MULTILINGUAL."LANGUAGE" as LANGUAGE1,
  COUNTRY_MULTILINGUAL.CURRENCY_NAME as CURRENCY_NAME
from
  gosales.gosales.dbo.COUNTRY_MULTILINGUAL COUNTRY_MULTILINGUAL
#prompt('Starting CountryCode',
  'integer',
  ' ',          // < = = this is a space
  'where COUNTRY_MULTILINGUAL.COUNTRY_CODE >'
)#
```

Syntax Substitution

Note the following:

- The `Datatype` argument is set to `token`, which means that the user-provided value is entered without any checking on the provided value.

Because checking is not performed on the value, the expression editor may indicate that the expression is not valid. When a valid user-provided value is supplied or if you provide a valid default value, expression editor will interpret the expression as valid.

- `Token` should be used only if there is a list of pick-values for the user.
- The `DefaultText` argument is specified. Therefore, this is an optional prompt and `group by COUNTRY` is used in the generated SQL.

```
Select
  COUNTRY_MULTILINGUAL.COUNTRY_CODE as COUNTRY_CODE,
  COUNTRY_MULTILINGUAL.COUNTRY as COUNTRY,
  COUNTRY_MULTILINGUAL."LANGUAGE" as LANGUAGE1,
  COUNTRY_MULTILINGUAL.CURRENCY_NAME as CURRENCY_NAME
from
  gosales.gosales.dbo.COUNTRY_MULTILINGUAL COUNTRY_MULTILINGUAL
#prompt('Sort column',
  'token',
  'group by COUNTRY',
  'group by '
)#
```

Examples - Create a Prompt That Uses a Parameter Map

When a report is run, you want your users to select a language for the data in the report. The following examples describe several ways you can do this.

Prompt That Uses a Session Variable

Note the following:

- The name of the prompt is specified using a lookup in the parameter map named PromptLabels. The key value is the session variable \$language.
- The where clause is using a parameterized column.

```
select
  ORDER_METHOD.ORDER_METHOD_CODE as ORDER_METHOD_CODE,
  ORDER_METHOD.ORDER_METHOD_#$language#
  as ORDER_METHOD_EN
from
  gosales.gosales.dbo.ORDER_METHOD ORDER_METHOD
#prompt ($PromptLabels{$language},
  ' ',
  ' ',
  'where ORDER_METHOD.ORDER_METHOD_' + $language + '
>'
) #
```

A Parameter Map That Nests Prompts

Note the following:

- In the model, there is a parameter map DynPromptLabels with # \$PromptLabels{\$language} #
- Part of the prompt information is run from a parameter map instead of being coded directly inside the SQL.
- The whole macro containing the prompt can be a value in a parameter map.

```
select
  ORDER_METHOD.ORDER_METHOD_CODE as ORDER_METHOD_CODE,
  ORDER_METHOD.ORDER_METHOD_#$language#
  as ORDER_METHOD_EN
from
  gosales.gosales.dbo.ORDER_METHOD ORDER_METHOD
#prompt ($DynPromptLabels{'ex9'},
  ' ',
  ' ',
  'where ORDER_METHOD.ORDER_METHOD_' + $language + '
>'
) #
```

Examples - Create a Multiple Value Prompt

When a report is run, you want your users to select one or more values. The following examples describe several ways you can do this.

Prompt with a Required Minimum

Note the following:

- The user must enter at least a single value.
- This resembles the first example on prompting for a country ([p. 174](#)).

```
select
  COUNTRY_MULTILINGUAL.COUNTRY_CODE as COUNTRY_CODE,
  COUNTRY_MULTILINGUAL.COUNTRY as COUNTRY,
```

```

COUNTRY_MULTILINGUAL."LANGUAGE" as LANGUAGE1,
COUNTRY_MULTILINGUAL.CURRENCY_NAME as CURRENCY_NAME
from
  gosales.gosales.dbo.COUNTRY_MULTILINGUAL COUNTRY_MULTILINGUAL
where COUNTRY_MULTILINGUAL.COUNTRY IN (#promptmany('CountryName')#)

```

Prompt with a Required Minimum with the Data Type Specified

Note the following:

- This resembles the second example on prompting for a country (p. 174).

```

select
  COUNTRY_MULTILINGUAL.COUNTRY_CODE as COUNTRY_CODE,
  COUNTRY_MULTILINGUAL.COUNTRY as COUNTRY,
  COUNTRY_MULTILINGUAL."LANGUAGE" as LANGUAGE1,
  COUNTRY_MULTILINGUAL.CURRENCY_NAME as CURRENCY_NAME
from
  gosales.gosales.dbo.COUNTRY_MULTILINGUAL COUNTRY_MULTILINGUAL
where COUNTRY_MULTILINGUAL.COUNTRY_CODE IN (
  #promptmany('Selected CountryCodes',
    'integer',
    '',
    '',
    '[gosales].[COUNTRY_MULTILINGUAL].[COUNTRY_CODE]')#
)

```

Optional Prompt with the Data Type and Default Value Specified

Note the following:

- The In clause and both parentheses are part of the SQL statement.

```

select
  COUNTRY_MULTILINGUAL.COUNTRY_CODE as COUNTRY_CODE,
  COUNTRY_MULTILINGUAL.COUNTRY as COUNTRY,
  COUNTRY_MULTILINGUAL."LANGUAGE" as LANGUAGE1,
  COUNTRY_MULTILINGUAL.CURRENCY_NAME as CURRENCY_NAME
from
  gosales.gosales.dbo.COUNTRY_MULTILINGUAL COUNTRY_MULTILINGUAL
where COUNTRY_MULTILINGUAL.COUNTRY_CODE IN (
  #promptmany('Selected CountryCodes',
    'integer',
    '10'
  )#
)

```

Prompt That Adds Text Before the Syntax

Note the following:

- This example uses the TextFollowing argument.

```

select
  COUNTRY_MULTILINGUAL.COUNTRY_CODE as COUNTRY_CODE,
  COUNTRY_MULTILINGUAL.COUNTRY as COUNTRY,
  COUNTRY_MULTILINGUAL."LANGUAGE" as LANGUAGE1,
  COUNTRY_MULTILINGUAL.CURRENCY_NAME as CURRENCY_NAME
from
  gosales.gosales.dbo.COUNTRY_MULTILINGUAL COUNTRY_MULTILINGUAL
#promptmany('Selected CountryCodes',
  'integer',
  ' ', // < = = this
  is a space
)

```

```

        'where COUNTRY_MULTILINGUAL.COUNTRY_CODE IN
    ( ' ,
      ' ' ,
      ' ) '
) #

```

Optional Prompt That Adds Text Before the Syntax

```

Select
  COUNTRY_MULTILINGUAL.COUNTRY_CODE as COUNTRY_CODE,
  COUNTRY_MULTILINGUAL.COUNTRY as COUNTRY,
  COUNTRY_MULTILINGUAL."LANGUAGE" as LANGUAGE1,
  COUNTRY_MULTILINGUAL.CURRENCY_NAME as CURRENCY_NAME
from
  gosales.gosales.dbo.COUNTRY_MULTILINGUAL COUNTRY_MULTILINGUAL,
  gosales.gosales.dbo.COUNTRY XX
where COUNTRY_MULTILINGUAL.COUNTRY_CODE = XX.COUNTRY_CODE
#promptmany('Selected CountryCodes',
  'integer',
  ' ' ,
  ' and COUNTRY_MULTILINGUAL.COUNTRY_CODE IN ( ' ,
    ' ' ,
    ' ) '
) #

```

Organizing the Model

Information about organizing an SAP BW model appears in a different topic ([p. 246](#)).

When you organize the model, you make it easier for your users to find and understand the data in the model. You also make the model easier for you to manage and maintain. We recommend that you create several views, or layers, in the model:

- Keep the metadata from the data source in a separate namespace or folder.

In IBM® Cognos® Framework Manager, this is called the import view.

- Create one or more optional namespaces or folders for resolving complexities that affect querying using query subjects or dimensional objects.

To use IBM Cognos Analysis Studio or any OLAP-style queries, there must be a namespace or folder in the model that represents the metadata with dimensional objects.

- Create one or more namespaces or folders for the augmented business view of the metadata that contains shortcuts to dimensions or query subjects.

In Framework Manager, these are called the business view. Use business concepts to model the business view. One model can contain many business views, each suited to a different user group. You publish the business views.

Security can be defined in any of the views. It depends on your business requirements. For example, if you need to keep everyone from viewing an object, you add security to the object in the import view. Typically security is applied in the business view.

To organize the model, you can do the following:

- ❑ create star schema groups ([p. 180](#))
- ❑ include metadata in several folders by using shortcuts ([p. 184](#))

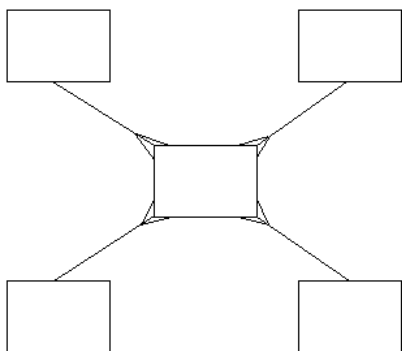
- ☐ create namespaces or folders ([p. 187](#))
- ☐ create query item folders ([p. 188](#))
- ☐ create measure folders ([p. 188](#))

Create a Star Schema Group

Use the **Star Schema Grouping** wizard to quickly create groups of shortcuts that will provide context for your users regarding which objects belong together. This makes the model more intuitive for your users. Star schema groups can also facilitate multiple-fact reporting by allowing the repetition of shared dimensions in different groups. This helps your users to see what different groups have in common and how they can do cross-functional, or multiple-fact, reporting. For more information, see "[Multiple-fact, Multiple-grain Queries](#)" ([p. 326](#)).

Star schema groups also provide context for queries with multiple join paths. By creating star schema groups in the business view of the model, you can clarify which join path to select when many are available. This is particularly useful for fact-less queries.

In a star schema design, numeric, transactional data is contained in a central fact table with related dimension tables radiating out from the fact table.



Star schema groups can contain the selected dimensions, query subjects, or shortcuts. The objects in a star schema group cannot reference, or be referenced by, any object outside the group. Therefore, Framework Manager automatically creates a shortcut for any object that is referenced by an object outside the group.

For example, in a project for sales analysis, you include these dimensions:

- dates of sales (Time)
- locations of sales (Region)
- product details (Product)
- customer information (Customer)

You include quantity in the fact table.

Steps

1. Select one or more dimensions or query subjects.
2. From the **Tools** menu, click **Create Star Schema Grouping**.

3. If you want to exclude an object from the group, in the **Available objects** box, clear the check box next to the object.
4. Do one of the following:
 - To add shortcuts to the group, click **Create a shortcut for all selected objects**.
 - To move the objects to the group, click **Create shortcuts only for objects that are used outside the star schema**.
5. To move the selected objects to a separate namespace, ensure that the **Create a new namespace for this grouping** check box is selected and type the name in the **Namespace name** box.
6. Click **OK**.
7. If there are multiple relationships, also known as role-playing dimensions, create relationship shortcuts for them (p. 83), or create individual dimensions or query subjects if you must rename them.

A Star Schema Group Based on One Dimension or Query Subject

Generally, you select a single object when it is a fact that has a relationship to every dimension or query subject that you want in the star schema group. When you create a star schema group that is based on one object, the following occurs:

- Framework Manager shows a list of objects with which it has relationships so that you can quickly select the objects that you want in the group.
- The name of the group is based on the name of the fact table.
- The new group is created under the same parent as the selected object.

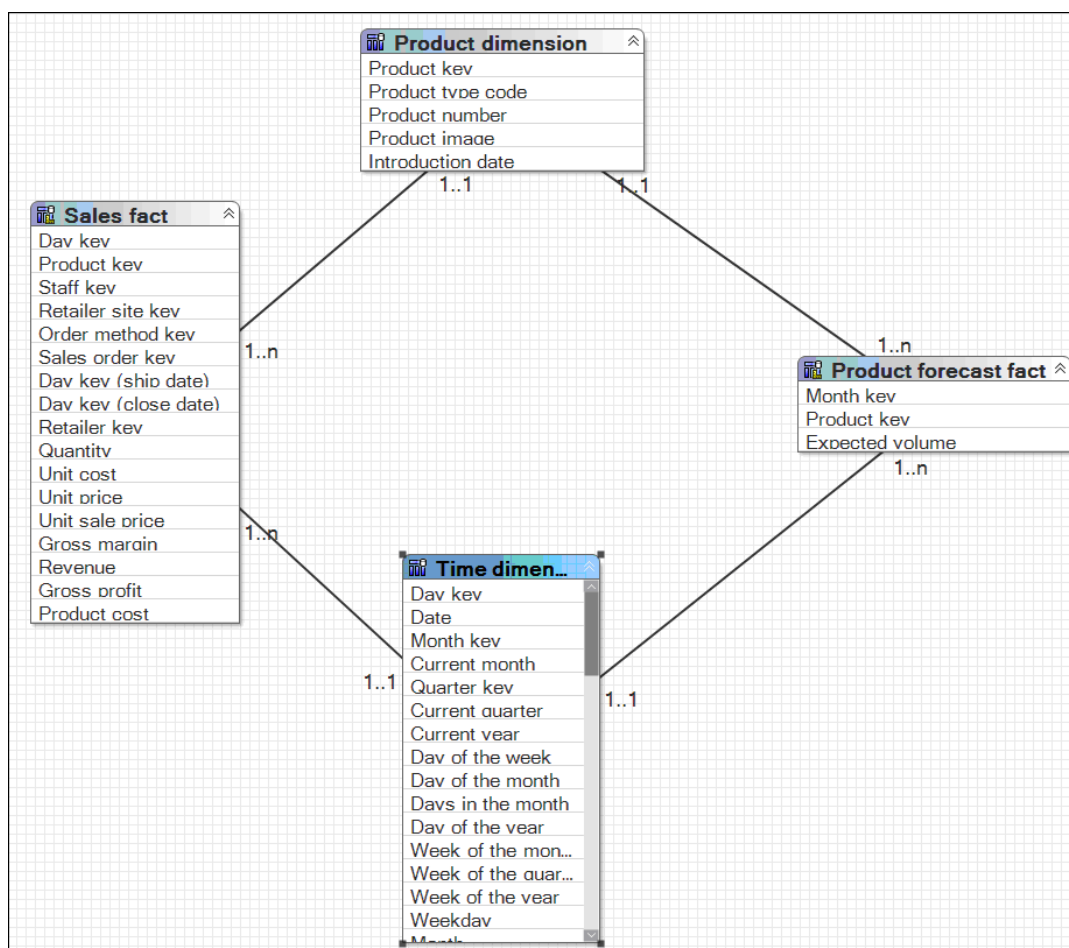
A Star Schema Group Based on Multiple Dimensions or Query Subjects

Selecting multiple dimensions or query subjects is useful if you want to group dimensions or query subjects that do not already have relationships defined. The new group is placed under the nearest common parent of the dimensions or query subjects.

Resolve Multiple Conformed Star Schemas

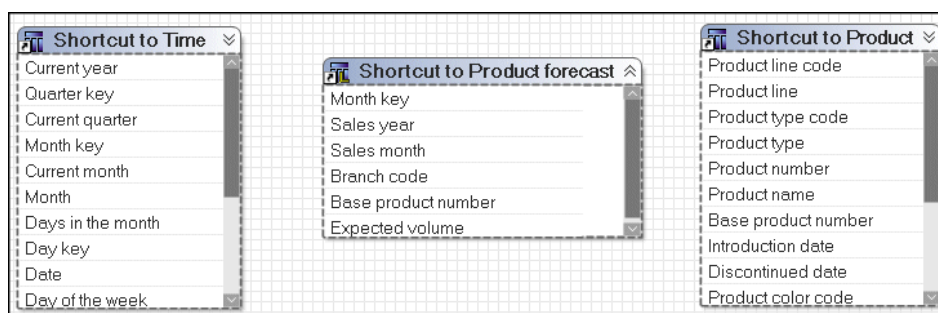
You will likely see dimensional query subjects that are joined to more than one fact query subject. Join ambiguity is an issue when you report using items from multiple dimensions or dimensional query subjects without including any items from the measure dimension or fact query subject. This is called a fact-less query.

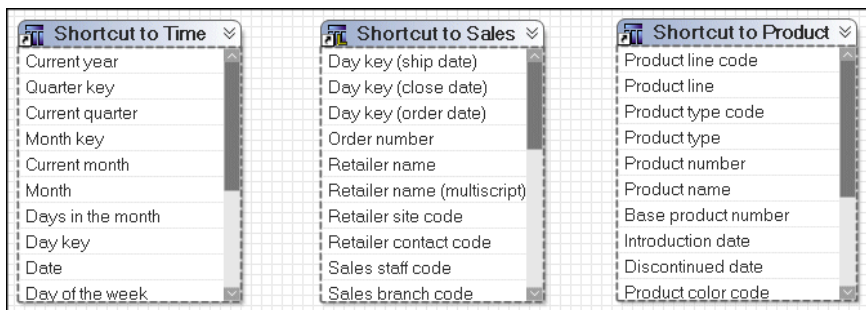
For example, Product and Time dimensions are related to the Product forecast and Sales facts.



Using these relationships, how do you write a report that uses only items from Product and Time? The business question could be which products were forecasted for sale in 2005 or which products were actually sold in 2005. Although this query involves only Product and Time, these dimensions are related through multiple facts. There is no way to guess which business question is being asked. You must set the context for the fact-less query.

In this example, we recommend that you create two namespaces, one containing shortcuts to Product, Time, and Product forecast, and another containing Product, Time, and Sales.





When you do this for all star schemas, you resolve join ambiguity by placing shortcuts to the fact and all dimensions in a single namespace. The shortcuts for conformed dimensions in each namespace are identical and are references to the original object. **Note:** The exact same rule is applied to regular dimensions and measure dimensions.

With a namespace for each star schema, it is now clear to your users which items to use. To create a report on which products were actually sold in 2005, they use Product and Year from the Sales Namespace. The only relationship that is relevant in this context is the relationship between Product, Time, and Sales, and it is used to return the data.

Steps

1. Select one of the measure dimensions.
2. From the **Tools** menu, click **Create Star Schema Grouping**.
3. If you want to exclude an object from the group, in the **Available objects** box, clear the check box next to the object.
4. Click **Create a shortcut for all selected objects**.
5. To move the selected objects to a separate namespace, ensure that the **Create a new namespace for this grouping** check box is selected and type the name in the **Namespace name** box.
6. Click **OK**.
7. Repeat these steps for the other measure dimension.

Model a Snowflaked Dimension as a Star Dimension

A snowflaked dimension removes low-level cardinality attributes from the dimension tables and places them in secondary dimensions that are linked back to the original dimensions by artificial keys.

To model a snowflaked dimension as a star dimension, do the following:

- ☐ Select the query subjects that are required to access the data, and put them in a new namespace (p. 187).
- ☐ Ensure that all relationships are correct (p. 78).
- ☐ Handle multilingual metadata (p. 131).
- ☐ Create a model dimension for each snowflaked dimension:
 - Select the query subjects you need.

- From the **Actions** menu, click **Merge in New Regular Dimension**.
 - Rename the new model dimension. By default, its name is composed of the concatenated names of the original objects.
- ❑ If you require multiple levels, do one of the following:
- Use the dimension map to define hierarchies and levels for the dimension ([p. 115](#)).
 - Specify determinants if the levels are stored in a single query subject ([p. 92](#)).

Use Shortcuts

Information about shortcuts for SAP BW metadata appears in a different topic ([p. 247](#)).

A shortcut is a pointer to an object, such as a relationship, a dimension, a query subject, or a folder. We recommend that you use shortcuts in the business view when there is an overlap between user groups and you want to include the metadata in more than one folder. With shortcuts, you can have multiple references to an object.

For example, you create folders named Orders, Products, and Customers. If you want both Orders and Customers to contain the same dimension, you must create a shortcut to the dimension and add it to both folders.

Note: Two shortcuts to namespaces or folders must not have the same name in a model. For other types of shortcuts (e.g., a shortcut of a query subject), the name must be unique within the parent namespace.

When you create a shortcut, IBM® Cognos® Framework Manager does not set the **Screen Tip** and **Description** properties. Unless you define these properties, the values shown in the IBM Cognos studios are the same as those defined in the object that the shortcut references.

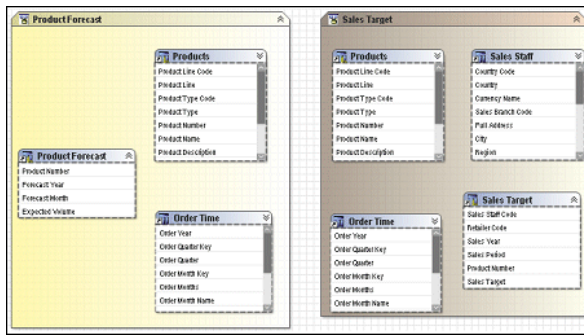
Tip: To go to the object that the shortcut references, right-click the shortcut and click **Go To Target**.

Shortcuts are less flexible from a presentation perspective than model objects, but they require much less maintenance because they are automatically updated when the target object is updated. If maintenance is a key concern and there is no need to customize the appearance of the query subject, use shortcuts.

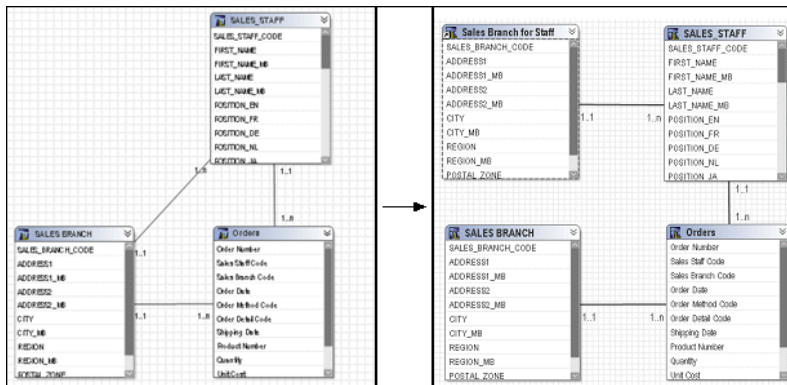
IBM® Cognos® Framework Manager has two types of shortcuts:

- regular shortcuts, which are a simple reference to the target object.
- alias shortcuts, which behave as if they were a copy of the original object with completely independent behavior. Alias shortcuts are available only for query subjects and dimensions.

Regular shortcuts are typically used as conformed dimensions with star schema groups, creating multiple references with the exact same name and appearance in multiple places. In the example below, the shortcuts created for Products and Order Time behave as references. If a query is written that brings Products from both Product Forecast and Sales Target, the query uses the definition of Products based on the original and this definition appears only once in the query.



Alias shortcuts are typically used in role-playing dimensions or shared tables. Because there is already an example in this document for role-playing dimensions, we will look at the case of shared tables. In this example, Sales Staff and Sales Branch can be treated as different hierarchies. From our knowledge of the data, we know that because staff can move between branches, we need to be able to report orders against Sales Branch and Sales Staff independently as well as together. To achieve this, we need to create an alias to Sales Branch that can be used as a level in the Sales Staff hierarchy.



With the new alias shortcut in place, it is possible to create queries that require orders by sales branch and orders by sales staff with their current branch information simultaneously.

Shortcuts and Relationships

When you decide where to place shortcuts, consider how the scope of the shortcut affects relationships. Shortcuts in a different folder from the target query subject use the relationships of the target query subject. Shortcuts in the same folder as the target query subject ignore the relationships of the target query subject and use only those specified for the shortcut.

You can specify a different relationship for a shortcut than the relationships of the target query subject. By creating relationships from the shortcut to other query subjects, you avoid cross-join errors in the model.

You cannot create shortcuts to scope relationships.

Shortcuts and Dimensions or Query Subjects

Shortcuts result in fewer dimensions or query subjects to maintain. You can keep dimensions or query subjects in the import view and keep shortcuts in the business view. Shortcuts are most frequently used when creating star schema groups.

When you create a shortcut to a dimension or query subject, you cannot customize which query items are in the shortcut. The entire dimension or query subject is included in the shortcut.

When you open a model from a previous release, the **Shortcut Processing** governor is set to **Automatic**. When **Automatic** is used, shortcuts work the same as in previous releases, that is, a shortcut that exists in the same folder as its target behaves as an alias, or independent instance, whereas a shortcut existing elsewhere in the model behaves as a reference to the original. To take advantage of the **Treat As** property, it is recommended that you verify the model and, when repairing, change the governor to **Explicit**. The repair operation changes all shortcuts to the correct value from the **Treat As** property based on the rules followed by the **Automatic** setting, this means that there should be no change in behavior of your model unless you choose to make one or more changes to the **Treat As** properties of your shortcuts.

When you create a new model, the **Shortcut Processing** governor is always set to **Explicit**.

When the governor is set to **Explicit**, the shortcut behavior is taken from the **Treat As** property and you have complete control over how shortcuts behave without being concerned about where in the model they are located.

Shortcuts can be created by the **Create Star Schema Grouping** wizard. For example, a fact table and its dimension tables are stored in the import view. If you want to represent conformed dimensions in several star schema groups, only one dimension or query subject can exist for each dimension table. Use shortcuts for all other instances of the dimension or query subject. By using shortcuts, you can build queries involving multiple fact tables that are related through shared dimension tables.

In the case of role-playing dimensions, there are two main approaches documented in ["Role-Playing Dimensions" \(p. 339\)](#). You can create a query subject and regular dimension for each role and then use shortcuts as references in star schema groups to allow the use of the role-playing dimensions as conformed dimensions, or you can create a shortcut for each role in each star schema group and create a new join between each role-playing shortcut and its related fact shortcut. As long as the target of the shortcuts is not contained in the same folder, all shortcuts behave as aliases.

The security you specify for an object is passed to shortcuts that reference the secured object. If you have a shortcut to a secured object, only users with permission to see the secured object can see the shortcut in the published package.

If a shortcut is to a dimension or query subject, you can specify the behavior for the shortcut in the **Treat As** property. You can set the property to one of the following:

- **Reference**
Use when you want an exact replica of a query subject in several places to behave as one object if referenced in the same query.
- **Alias**
Use when you want an exact replica of a query subject to behave as an independent object that follows an independent join path. Independent join paths must first be defined in the model.

The **Shortcut Processing** governor controls the behavior for all shortcuts. The **Shortcut Processing** governor takes priority over the **Treat As** property. For example, if the governor is set to **Automatic**,

the behavior of the shortcut is determined by the location of the shortcut relative to its target no matter what the setting of the **Treat As** property. For more information, see ["Set Governors"](#) (p. 304).

Steps

1. Right-click the query subjects, dimensions, or folders that you want to create shortcuts to, and do one of the following:
 - Click **Create, Alias Shortcut**.
 - Click **Create, Shortcut**.
 - Click **Create Star Schema Grouping**. This command is also available from the **Tools** menu.
2. For shortcuts to query subjects or dimensions, in the **Properties** pane, set the **Treat As** property to **Alias** or **Reference**.

Create a Folder or Namespace

Information about folders and namespaces for SAP BW metadata appears in a different topic ([p. 248](#)).

You can create folders or namespaces to organize objects in the model.

The most important thing to know about namespaces is that once you have begun authoring reports, any changes you make to the names of published namespaces will impact your IBM Cognos content. This is because changing the name of the namespace changes the IDs of the objects published in the metadata. Because the namespace is used as part of the object ID in IBM Cognos Framework Manager, each namespace must have a unique name in the model. Each object in a namespace must also have a unique name. Part of the strategy of star schema groups is placing shortcuts into a separate namespace, which automatically creates a unique ID for each object in the namespace. For relational databases, this allows us to use the same name for shortcuts to conformed dimensions in different star schema groups.

The next time you try to run a query, report, or analysis against the updated model, you get an error. If you need to rename the namespace that you have published, use **Analyze Publish Impact** to determine which reports are impacted.

Folders are much simpler than namespaces. They are purely for organizational purposes and do not impact object IDs or your content. You can create folders to organize objects by subject or functional area. This makes it easier for you to locate metadata, particularly in large projects.

The main drawback of folders is that they require unique names for all query subjects, dimensions, and shortcuts. Therefore, they are not ideal for containing shared objects.

Tip: When viewing metadata in the **Diagram** tab, you can expand or collapse folders and namespaces. From the **Diagram** menu, click **Collapse All** or **Expand All**.

If you set security on a folder and then move objects into the folder, confirm that exclusions are set correctly.

Steps to Create a Namespace

1. From the **Actions** menu, click **Create, Namespace**.
2. Right-click the namespace, click **Rename**, and give the namespace a descriptive, unique name.

3. Add objects by importing metadata or moving model objects or shortcuts to the objects into the namespace.

Steps to Create a Folder

1. From the **Actions** menu, click **Create, Folder**.
2. In the **Folder name** box, type a name for the new folder.
3. Click **Next**.
4. Choose whether to move the objects or to create shortcuts:
 - To move selected objects to the folder, click **Move the selected items to the new folder**. When you move an object that participates in a relationship, the relationships to this object also move.
 - To create shortcuts that reference selected objects, click **Create a shortcut for the selected items**. Do not select all the objects in the namespace to avoid creating a recursive structure in the published package.
5. Select the objects you want to add to the folder.
6. Click **Finish**.

Create a Query Item Folder

You can create query item folders to organize query subjects or dimensions that contain a large number of query items. A query item folder can contain only query items and query item folders.

If you create a query item folder, you cannot change the order of query items in the **Edit Definition** dialog box for the model query subject. You can change the order of items only in the **Project Viewer**.

Steps

1. In the **Project Viewer** pane, click a query subject or dimension.
2. From the **Actions** menu, click **Create, Query Item Folder**.

A new query item folder appears in the **Project Viewer**, under the query items that belong to that query subject or dimension.
3. Drag the query items that you want into the query item folder.

You cannot add query items that do not exist in the parent query subject or dimension.

Create a Measure Folder

You can create measure folders to organize measure dimensions that contain a large number of query items. You can nest measure folders within other measure folders.

You cannot create a measure folder from a measure shortcut.

If you create a measure folder, you cannot change the order of measures in the **Edit Definition** dialog box for the measure dimension. You can change the order of measures only in the **Project Viewer**.

Steps

1. In the **Project Viewer** pane, click a measure dimension.
2. From the **Actions** menu, click **Create, Measure Folder**.

A new folder appears in the **Project Viewer**, under the measures that belong to that measure dimension.

3. Drag the query items that you want into the measure folder.

You cannot add measures that do not exist in the parent measure dimension.

Create a Durable Model

When building a model, you must consider the possibility of later changes to user requirements that you might need to reflect in the model. Your goal is to build a flexible model that can withstand necessary changes without impacting existing reports, report authors, and end users.

Renaming query items is one of the most frequent changes that modelers need to implement in their models. If your models are durable, you can make these types of changes quickly, with no impact on existing reports. The calculations and filters that reference the renamed query items also remain valid.

Durable models are useful in both single-language and multilingual environments when renaming query items as a result of changing business requirements. In a multilingual environment, durable models also simplify the translation process by allowing you to specify language-specific labels for query items without the risk of breaking existing report references for other languages.

When working with durable models, remember the following conditions:

- Specify a proper design language for your project.
Choose the locale version of the language that is not included in your business requirements. This could be a less-frequently used locale, for example, English (Zimbabwe). You cannot change the design language of a project after you create a model.
- Do not use the design language as the active language.
When you create a new project, the design language automatically becomes the project active language. Ensure that you change this by assigning other language as the active language for your project.
- Rename your query items only in the active languages of the project.
When making changes, ensure that your current project active language is not the same as the project design language.
- Do not change the structure of published packages.

The structure of namespaces, query subjects, query items, dimensions, shortcuts, and so on, must remain unchanged.

To make your model durable, set the project property **Use Design Locale for Reference ID** to true.

For more information about durable models, access the Proven Practices documentation in the Support section of the IBM® Cognos® Customer Service Center (http://www.ibm.com/software/data/support/cognos_crc.html).

Steps to create a durable model

1. From the **Welcome** page, click **Create a new project**.

Tip: If you are in IBM Cognos Framework Manager, click **New** from the **File** menu.

2. In the **New Project** page, specify a name and location for the project, and click **OK**.

3. In the **Select Language** page, click the design language for the project, and then click **OK**.

Ensure that you choose the proper design language, as documented earlier in this section. You cannot change the language that you select after you click **OK**, but you can add other project languages later.

4. In the **Metadata Wizard**, click **Next** to import your metadata.

5. Follow the instructions in the **Metadata Wizard**:

- Select a data source connection, and click **Next**.
If the data source connection that you want is not listed, you must first create it ([p. 56](#)).
- Select the check boxes for the objects that you want to import.
- Specify how the import should handle duplicate object names. Choose whether to import and create a unique name. If you choose to create a unique name, the imported object appears with a number. For example, you see QuerySubject and QuerySubject1 in your project.
- If you want to import system objects, select the **Show System Objects** check box, and then select the system objects that you want to import.
- Specify the criteria to use to create relationships, and click **Import**.
For more information, see "[Relationships](#)" ([p. 78](#)).

You see a list of objects that could not be imported, and a count of objects that were imported.

6. Click **Finish**.

Save the project file (.cpf) and all related files in one folder. When you save a project with a different name or format, ensure that you save the project in a separate folder.

7. Click the project name in **Project Viewer**, and set the project property **Use Design Locale for Reference ID** to true.

Note: Changing this property back to false later, after renaming query items in the model, would result in breaking reports based on this model.

8. From the **Project** menu, click **Languages, Define languages**, and add the required languages to the project choosing one of them as **Active language**.

When you specify the active language, ensure that it is not the same as the design language. For more information, see ["Add a Language to a Project"](#) (p. 134).

9. Save the project.
10. Create the required packages, and publish them to IBM Cognos Connection.

Use the published packages to create content in IBM Cognos Report Studio, IBM Cognos Query Studio, or IBM Cognos Event Studio. For example, create reports in Report Studio or Query Studio.

You can now test the model to ensure that it works as expected ([p. 191](#)).

Steps to test a durable model

1. Launch Report Studio using the package published in [Steps to create a durable model](#), and create and save a report.
2. In Framework Manager, open the project created in [Steps to create a durable model](#), and ensure that the project active language is different than the design language.
3. In the active language, rename some of the query items included in the package published in step 1.
4. Re-publish the package to IBM Cognos Connection overriding the original package.
5. Launch Report Studio again using the re-published package, and open the report created in step 1.

The report shows the changed query item names in the **Insertable Objects** pane. However, the report specification shows the query item names in the design language, not in the active language, in which you made the changes.

6. Run the report.

The columns representing the renamed query items show the new names.

The model is durable because renaming its query items did not break existing reports.

Analyze a Model

You can analyze the metadata in a model by using the **Model Advisor**, which is an automated tool that applies rules based on current modeling guidelines and identifies areas of the model that you need to examine. To assist you in understanding the nature of the highlighted issue as well as some possible actions, you are provided with links to the appropriate sections of the documentation. The **Model Advisor** is not a replacement for a knowledgeable modeler; it provides new modelers with an assistive tool and more experienced modelers with a diagnostic tool.

You can select one or more tests to run against the selected model or subset of a model. We recommend that you verify the model and fix errors before analyzing the model ([p. 251](#)).

If you are analyzing a new model, we recommend the following workflow:

- Analyze newly-imported objects, especially their relationships and determinants.
- Use the issues that are identified to resolve potential query generation issues.
- As you build additional views, use the **Model Advisor** to analyze each one for potential issues.
- Before publishing the model, use the **Model Advisor** on objects that will be published.

If you are analyzing an older, established model or a model that is not yet complete, use the **Model Advisor** to validate modeling practices. The workflow is similar to that used for new models: start at the database view and work up.

Facts Identified by Cardinality

This test looks for query subjects that have only the many (n) cardinality on the ends of the relationship in which they are involved. Query subjects with this cardinality are treated as facts when generating queries so it is important to ensure that they are correctly identified.

Query Subjects That Can Behave as Facts or Dimensions

This test looks for query subjects that have a combination of the many (n) and one or zero (1,0) cardinality. Mixed cardinality means that the behavior of a query subject can change depending on the other query subjects used in a query. This can lead to unpredictable queries in some cases. If the query subject is evaluated as a fact, it will be included in the query path. If it is evaluated as a dimension, it may be skipped if it is not directly referenced in the query. In the interests of ensuring predictable behavior, it is recommended that you resolve these scenarios.

Mixed cardinality can indicate a query subject that is part of a snowflaked dimension or master-detail relationship. In the case of intermediate tables in snowflaked dimensions, there is no issue to resolve. Problems may arise in situations where there are multiple query paths available; these cases should be highlighted by the test for query subjects with multiple relationships.

Query Subjects with Multiple Relationships

This test looks for query subjects that have either many relationships between two objects or a loop join that does not represent the star schema join pattern.

Multiple relationships between two query subjects is often associated with role-playing dimensions. With role-playing dimensions, you create aliases that enable each role to have a different join and different independent behavior. Multiple join paths that are indicative of loop joins (except for star schemas) may lead to problems of incorrectly split queries. This impacts the predictability of query generation for your users.

If IBM® Cognos® software has multiple relationships with no distinguishing criteria to choose from, it uses the relationship that comes first alphabetically. If you need to create a query that uses a different relationship, you always have a problem. As well, if you want to use filters on criteria that are specific to the role defined by the relationships and those filters are mutually exclusive, then no data is shown in the report. For information about role-playing dimensions and an example, see ["Role-Playing Dimensions" \(p. 339\)](#).

Note: When cardinality clearly identifies facts and dimensions, IBM Cognos software can automatically resolve loop joins that are caused by star schema data when you have multiple fact tables joined to a common set of dimension tables. The **Model Advisor** ignores star schema join patterns because they do not qualify as problematic joins.

Query Subjects That Join to Themselves

This test looks for reflexive and recursive relationships. These relationships imply two or more levels of granularity. IBM® Cognos® Framework Manager imports reflexive relationships but does not use them when executing queries. Reflexive relationships, also called self-joins, are shown in the model for the purpose of representation only.

Most often a query subject that joins to itself indicates a parent-child relationship. Data sets that have parent-child relationships can be of definite or indefinite depth. The only way to know which you are dealing with is to understand the data and the business concept represented.

Whether you have a problem when you encounter a parent-child relationship depends on how you intend to use the data in your application. There may be cases where only one level of the relationship is necessary, and this is a fairly simple scenario to model. If you know the data to be fully populated for each level and to have a definite number of levels, you can also choose a modeling-only solution. However, for cases where there is a significant amount of data, the data is not fully populated at each level (ragged or unbalanced hierarchy) or the number of levels in the data could change over time or both, it is recommended that you transform the data to a flat structure with a fixed number of columns before modeling in Framework Manager.

Determinants That Conflict with Relationships

This test looks for determinants that conflict with the relationship defined for the query subject. Determinants are used to ensure that the aggregation in reports is correct and that queries generate correctly.

Determinants reflect granularity by representing subsets or groups of data in a query subject and are used to ensure correct aggregation of this repeated data. Determinants are most closely related to the concept of keys and indexes in the data source and are imported based on unique key and index information in the data source. We recommend 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.

The **Model Advisor** checks query subjects with determinants and flags the ones that meet one of the following criteria:

- the relationship references all query items in a unique determinant and the cardinality of the relationship is not 1:1 or 0:1
- the relationship references some of the query items in a unique determinant and the cardinality of the relationship is not 1:n or 0:n

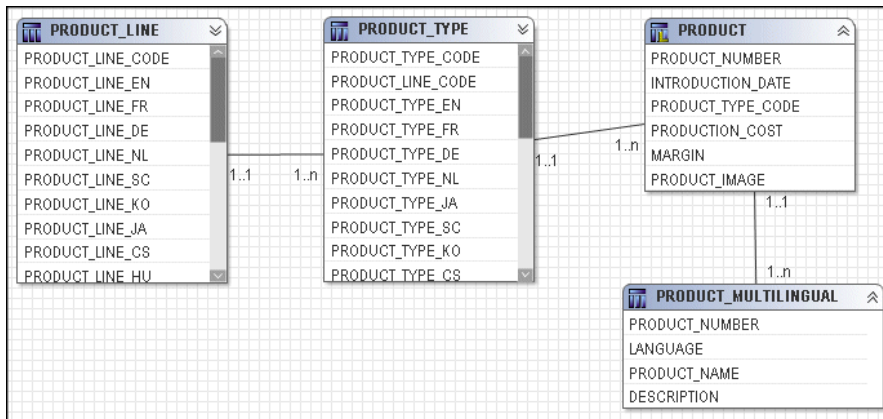
The Model Advisor also flags occurrences where the keys of a relationship do not match the keys of a group by determinant.

Factors That Will Override the Minimized SQL Setting

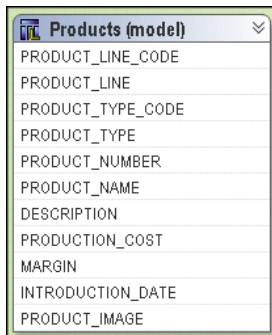
This test looks for various factors that override the SQL Generation type setting of Minimized, such as modified data source query subjects, relationships between model query subjects, or determinants for model query subjects.

When you use minimized SQL, the generated SQL contains only the minimal set of tables and joins needed to obtain values for the selected query items.

To see an example of what minimized SQL means, here are four query subjects, Product Line, Product Type, Product, and Product Multilingual that join to each other.



They can be combined in a model query subject.



For example, if you test the Products (model) query subject as a whole, you see that four tables are referenced in the `from` clause of the query.

```
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.DESCRPTION 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_82..gos1.PRODUCT_LINE PRODUCT_LINE,
  gos1_82..gos1.PRODUCT_TYPE PRODUCT_TYPE,
  gos1_82..gos1.PRODUCT PRODUCT,
  gos1_82..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 Product Name, you see that the resulting query uses only Product Multilingual, which is the table that was required. This is the effect of minimized SQL.

```

select
  PRODUCT_MULTILINGUAL.PRODUCT_NAME as Product_Name
from
  gos1_82..gos1.PRODUCT_MULTILINGUAL PRODUCT_MULTILINGUAL
where
  (PRODUCT_MULTILINGUAL."LANGUAGE" - N'EN')

```

Embedded Calculations That Use the Calculated Aggregation Type

This test detects where you have set the **Regular Aggregate** property to **calculated** for embedded calculations.

The calculated aggregation type is supported only for the following:

- stand-alone calculations
- calculations that are embedded within measure dimensions and are based on measures from the same measure dimension

For more information about calculated aggregations, see ["Rules for Interpreting Calculated Aggregations" \(p. 143\)](#).

Query Subjects That Can Cause a Metadata Caching Conflict

This test looks for factors that override cached metadata, such as data source query subjects whose SQL has been modified or query subjects that contain calculations or filters.

IBM® Cognos® Framework Manager stores the metadata that is imported from the data source. However depending on the governor settings and certain actions you take in the model, this metadata might not be used when preparing a query. If you select the **Allow enhanced model portability at run time** governor, Framework Manager always queries the data source for information about the metadata before preparing a query. If you have not selected the **Allow enhanced model portability at run time** governor, Framework Manager accesses the metadata that has been stored in the model instead of querying the data source. There are exceptions and the main cases are:

- Any modification of the SQL in a data source query subject. This includes the use of macros.
- Adding a calculation or filter to a data source query subject.

Note: The metadata queries generated by IBM Cognos software are well supported by most relational database management system vendors and should not have a noticeable impact on most reporting applications.

Constraints

The **Model Advisor** is only intended for use with relationally-based metadata models. We do not recommend running the **Model Advisor** against an entire model; instead apply it to specific views

one at a time to ensure that the feedback is being taken in context. For example, if an issue in an import view has not been addressed, the issue might be resolved by modeling that is done in an intermediate view.


For large models or namespaces, the **Model Advisor** might not return results immediately.

Because the **Model Advisor** is not data sensitive, you must know the data and model the metadata appropriately for your business intelligence needs. Not all items flagged by the **Model Advisor** are indicative of a problem. The context of each issue raised by the **Model Advisor** is important.

Steps

1. Click one or more objects to analyze.
 - Select query subjects, dimensions, hierarchies, calculation, query items, or shortcuts to analyze objects that will appear on a specific report to test the report before it is created.
 - Select a folder or namespace to analyze all its objects. If an object references an object in another folder or namespace, the referenced object is also analyzed.
 - Select a package before publishing it to ensure it follows the recommended guidelines for modeling.
2. From the **Tools** menu, click **Run Model Advisor**.

Tip: You can also right-click one or more objects and then click **Run Model Advisor**.
3. In the **Options** tab, select the criteria that you want to use in the analysis.
4. Click **Analyze**.
5. In the **Model Advisor** tab, review the issues that are identified.

There is a description of each issue, a link to more information about each issue, and a list of objects that are impacted by the issue.
6. To understand whether there is a problem with an object, click the **context explorer** button  in the **Action** column of the report.

The **Context Explorer** shows the objects that the selected object is connected to. You can select a related object and see which objects it is connected to.

Chapter 6: Working with SAP BW Metadata

Note: Information on relational metadata is in "[Modeling Relational Metadata](#)" (p. 77).

Note: For information about modeling for use with dynamic query mode, see the *Dynamic Query Guide*.

After importing metadata, you must ensure that it is set up to meet your users' reporting requirements, and provide any additional information that they require. Enhancements you make in IBM® Cognos® Framework Manager do not affect the original data source.

Tip: To verify that the model meets the reporting requirements, you can select objects that will appear in a report and test them. The test results show you the report that your users will see as well as the SQL and messages from the IBM Cognos software, if any. Or you can publish a package at any time and then use the package to create reports.

You can check the project at any time to ensure that the references between the objects it contains are valid ([p. 251](#)).

You can do the following when working with SAP BW metadata in IBM® Cognos® Framework Manager:

- ☐ Import the metadata ([p. 197](#)).
- ☐ Work with dimensions ([p. 204](#)).
- ☐ Control how data is used and formatted by checking query item properties ([p. 219](#)).
- ☐ If required, add more business rules, such as calculations and filters, to refine the retrieved data and to ensure that the right information is available for your users ([p. 237](#)).
- ☐ Organize the model by creating separate views for each user group that reflect the business concepts familiar to your users ([p. 246](#)).
- ☐ If required, adjust settings in Framework Manager and the IBM Cognos studios to optimize performance.

After working with the model, you can create a package and publish it for your users. For more information, see "[Publish a Package](#)" ([p. 266](#)).

Note: You can also create packages for SAP BW cubes and queries directly in IBM Cognos Connection. For more information, see the section about packages in the *Administration and Security Guide*.

Import from an SAP BW Data Source

When you import from an SAP BW data source, you can import all the metadata or import only the objects you select. For information about mapping SAP BW metadata objects to IBM® Cognos® Framework Manager objects, see "[Mapping SAP BW Objects to Framework Manager](#)" ([p. 203](#)).

You may want to have different views (or layers) in the model: an import view to contain the metadata you imported from the data source and a business view where you enhance the metadata. After importing, you can copy the metadata to the business view. You then have two views that must be synchronized with the BW InfoProvider.

Tip: If you want to expose calculated key figures from a SAP BW Query, you must import the SAP BW Query (p. 202).

Access to SAP BW Metadata and Data

When using an SAP BW data source, users' access to an InfoCube or InfoQuery metadata does not imply that they also have access to data within those objects. To enable Framework Manager to retrieve metadata from SAP BW, access privileges must be set up within the SAP BW system. To ensure that users have proper access permissions, verify the permissions assigned to the users' roles. The following authorization objects must be configured so that Framework Manager can import information cubes or data sources, known as InfoCubes in the SAP BW system.

- S_RFC

Set the **Activity** field to the value: 16

Set the **Name of RFC to be protected** field to the value: SYST, RSOB, SUGU, RFC1, RS_UNIFICATION, RSAB, SDTX, SU_USER

Set the **Type of RFC object to be protected** field to the value: FUGR

- S_TABU_DIS

Set the **Activity** field to the value: 03

Set the **Authorization Group** field to the value: &NC&

Note: &NC& represents any table that does not have an authorization group. For security reasons, create a new authorization group and assign the table RSHIEDIR to it. The new authorization group restricts the user's access to the above table only, which is needed by the modeling tool. Create the new authorization group as a customization in the SAP system.

- S_USER_GRP

Set the **Activity** field to the value: 03, 05

Set the **User group in user master main** field to the default value.

- S_RS_COMP

Set the **Activity** field to the default value.

Set the **Info Area** field to the value: *InfoArea Technical Name*

Set the **Info Cube** field to the value: *InfoCube Technical Name*

Set the **Name (ID) of reporting components** field to the default value.

Set the **Type of reporting components** field to the default value.

- S_RS_COMP1

Set the **Activity** field to the default value.

Set the **Name (ID) of reporting components** field to the default value.

Set the **Type of reporting components** field to the default value.

Set the **Owner (Person Responsible)** field to the default value.

- **S_RS_HIER**

Set the **Activity** field to the value: 71

Set the **Hierarchy Name** field to the value: *Hierarchy Name*

Set the **InfoObject** field to the value: *InfoObject Technical Name*

Set the **Version** field to the value: *Hierarchy Version*

- **S_RS_ICUBE**

Set the **Activity** field to the value: 03

Set the **InfoCube sub-object** field to the values: DATA and DEFINITION

Set the **Info Area** field to the value: *InfoArea Technical Name*

Set the **InfoCube** field to the value: *InfoCube Technical Name*

Tips

- &NC& represents any table that does not have an authorization group. For security reasons, create a new authorization group and assign the table RSHIEDIR to it. The new authorization group restricts the user's access to the above table only, which is needed by Framework Manager. Create the new authorization group as a customization in the SAP BW system.
- You can use the asterisk (*) to represent all values, when it appears alone, or partial values, when used anywhere in a string.

SAP BW Hierarchies

When importing metadata, Framework Manager generates a dimension in each SAP BW characteristic.

Only one hierarchy associated with a given characteristic can be used in a report. Therefore, you should group dimensions that represent the hierarchies of a single characteristic into a folder or model query subject to make reporting easier for your users.

If there are multiple hierarchies in an SAP BW data source, the first hierarchy that is imported becomes the default hierarchy.

Framework Manager supports the following types of hierarchies:

- **characteristic**

This is a list of all the characteristic values.

- **text node**

Non-leaf nodes contain only text and do not reference any other data source object.

- **characteristic value**

The nodes of each level of a presentation hierarchy are values from another characteristic.

- recursive

The nodes of the entire presentation hierarchy are from the characteristic itself.

If a characteristic is not in a time dimension but it is a date and is treated as a date in SAP BW, the characteristic is imported with the date data type.

Framework Manager does not support hierarchies that contain two or more types of nodes. These hierarchies are imported but are hidden in the Framework Manager model.

Because hierarchical metadata is automatically generated for SAP BW, you cannot change it in Framework Manager.

Versioned Hierarchies

You can import the following types of versioned hierarchies from an SAP BW data source:

- Version dependent hierarchy

A hierarchy can have multiple versions. Each version of a hierarchy can have a different structure, such as Sales by Region and Sales by Manager. During metadata import, Framework Manager identifies each version as a separate hierarchy and creates a dimension for each.

- Entire hierarchy time dependent

Each version has an associated time period that does not overlap with any other version of the same hierarchy. The structure of each version can be different. During metadata import, Framework Manager identifies each version as a hierarchy and includes the applicable time period as part of the dimension name.

- Time-dependent hierarchy structure

There is a single version of the hierarchy, but nodes within the hierarchy can be assigned applicable time periods. Over time, the structure of the hierarchy can change with new levels being introduced or removed. For example, levels that represent different sales districts can be added over time. During metadata import, Framework Manager identifies a time-dependent hierarchy structure as a non-versioned hierarchy and recognizes the structure of the hierarchy as of the current date.

The type of dimension determines which hierarchy is used and, for time-dependent hierarchies, which date to use to control the version.

Framework Manager sets the query key date of time-dependent hierarchies based on dates that are contained within the time-dependent hierarchy. You can then select specific versions of hierarchies. For hierarchies with versions on time, the default is the current date and time. The hierarchy that you apply to a characteristic depends upon the type of query key date: fixed, current, or variable. The query key date is set for a specific date.

For fixed date, include only the version that corresponds to the fixed date in the underlying SAP BW Query. For example, if the SAP BW Query has a fixed date such as 2005, only 2005 is imported.

For current date, include only the version that encompasses a time span appropriate for the present until some reasonable time in the future.

For variable, set the date for the variable in Framework Manager and include only the version of the hierarchy applicable to that date.

When you use Framework Manager to model SAP BW data, any versions or dates applied to a presentation hierarchy in SAP BW are not imported into the model. Therefore, all versions of the hierarchy are accessible in Framework Manager.

You may have a time-dependant hierarchy and a variable defined in SAP BW to establish the effective date for the hierarchy. In this case, assign a fixed date to the variable in Framework Manager and include only the dimension that corresponds to that date in the model.

In Framework Manager, if a versioned hierarchy is not time-dependent and has a fixed version, include only the version of the hierarchy associated with the selected version. Otherwise your users are presented with a hierarchy that is inaccessible.

SAP BW Structures

Many existing SAP BW queries contain structures that you can use in IBM® Cognos® queries to control the amount and order of information that your users see. For example, with dual structures, you can create a crosstab report with one structure on each axis.

The structures are:

- **key figure structure**
The SAP BW Query Designer automatically creates a key figure structure when you add key figures to a query. You must have at least one key figure to import the query metadata into IBM Cognos Framework Manager. This is true even when you do not use the key figure in reports. Therefore, you will always have a key figure structure.
- **characteristic structure**
A characteristic structure is a collection of characteristic values (members) from one or more dimensions. You create a characteristic structure in SAP by adding a structure to the query, and then adding the required members to the structure. In IBM Cognos software, the structure appears as an additional dimension that has only one multiple-root level.

When you import the SAP BW query into Framework Manager, the key figure structure appears in the measure dimension called Key Figures and the characteristic structure appears as an additional dimension.

If you re-import the same SAP BW query into Framework Manager, you must use the same setting for the **SAP BW Dual Structures Support** check box. Framework Manager does not allow you to select a different setting for the same query because different objects are then generated in the model and this leads to errors. You can use different settings for different queries.

Steps to Access a Secured InfoCube

1. Create a query in Business Explorer Query Designer that accesses the InfoCube.
2. Create an authorization variable for each InfoObject in the underlying InfoCube for which there are authorizations.
3. For each variable, ensure that the Ready for Input option is disabled.
By default, this option is enabled.
4. Enable the query for access through OLE DB for OLAP.

5. Save the query.
6. In Framework Manager, reference the query instead of the InfoCube.

Steps to Import from an SAP BW Data Source

1. Ensure that there is a connection to the data source (p. 56).

For information about creating data source connections, see the IBM Cognos *Administration and Security Guide*.

2. Click the namespace, folder, or segment into which you want to import, and from the **Actions** menu, click **Run Metadata Wizard**.
3. Select **Data Sources**, and click **Next**.
4. Select an SAP BW data source connection and click **Next**.
5. Select the objects that you want to import.

If you are re-importing, the existing object is updated.

If you are importing new objects and an object with the same name exists, the new object is imported and a number is appended to the original name. For example, you see QuerySubject and QuerySubject1 in your project.

After they are imported, you cannot delete query items without deleting the entire query subject.

6. If you want to import a characteristic structure and a key figure structure, select the **SAP BW Dual Structures Support** check box.

The content of the **Select Objects** page is updated to reflect the dual structures in your data source.

Note: If you are re-importing the same SAP BW query (for example, because the underlying data source has changed), you must use the same setting for this check box. You cannot use a different setting for the same query because different objects are then generated in the model, and this leads to errors.

7. Select the languages that you want to import.

These languages must exist in the data source.

You can add languages to your project later, but you cannot return and import the language-specific metadata from the data source. After the import is complete, you must manually add the language-specific metadata.

8. Indicate whether you want Framework Manager to show the short name, long name, or the technical name for the dimensions.
 - If you select the short name and the field is empty, the long name is shown.
 - If you select the long name and the field is empty, the short name is shown.
 - If you select either short name or long name and both fields are empty, the technical name is shown.

9. To organize objects in the model the same way as in Business Explorer Query Designer, select the **Enhance model for SAP BW organization of objects** option.

You will then have a folder for each characteristic.

10. Click **Next**.

A list of objects that could not be imported appears with counts of objects that were imported.

11. Click **Finish**.

After importing, verify the usage and aggregation property values (p. 222). Fact tables may contain numeric columns that should not be aggregated, such as exchange rates.

When you want to recreate a query on another SAP BW system, use the SAP BW migration mechanism to transport the query. This ensures that the technical name of each measure remains the same so that any project that references the query can be directed to either system without any modifications to the project.

Mapping SAP BW Objects to Framework Manager

SAP BW objects are mapped to the following IBM® Cognos® Framework Manager objects.

SAP BW object	Framework Manager object
Query, InfoCube, RemoteCube, MultiCube	Namespace.
Characteristic	<p>A folder that contains dimensions.</p> <p>You must select the Enhance model for SAP BW organization of objects option when importing metadata to have a folder for each characteristic.</p> <p>Note: By default, Framework Manager imports SAP BW Currency and Unit of Measure characteristics. You can remove these characteristics if you do not need them.</p>
Dimension	<p>Dimension. The dimension may contain hierarchies representing each presentation hierarchy. The default hierarchy contains two levels representing</p> <ul style="list-style-type: none"> - the aggregation of all characteristic values, also known as the All value - all characteristic values
Key figure	Query item that is part of a measure dimension called Key Figures.

SAP BW object	Framework Manager object
Presentation hierarchy level	Level. Note: Level names must be defined in the Administrator Workbench to be meaningful.
Attribute	Query item associated with a level whose Usage property value is set to Attribute .
SAP BW variable	Data source property. For information about the SAP BW variables that Framework Manager supports, see "SAP BW Variables" (p. 230) .

For information about setting access privileges to retrieve metadata from SAP BW, see ["Access to SAP BW Metadata and Data" \(p. 198\)](#).

Dimensions

Information about dimensions based on relational metadata appears in a different topic ([p. 114](#)).

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 with in IBM® Cognos® Framework Manager are regular dimensions and measure dimensions. In SAP BW, measure dimensions are called key figures.

For example, in a project for sales analysis, you include these dimensions:

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

Modify a Regular Dimension

Information about creating a regular dimension for relational metadata appears in a different topic ([p. 115](#)).

A regular dimension contains descriptive and business key information and organizes the information in a hierarchy, from the highest level of granularity to the lowest. It usually has multiple levels and each level requires a key and a caption. If you do not have a single key for your level, it is recommended that you create one in a calculation.

Model regular dimensions are based on data source or model query subjects that are already defined in the model. You must define a business key and a string type caption for each level. When you verify the model, the absence of business keys and caption information is detected. Instead of joining model regular dimensions to measure dimensions, create joins on the underlying query subjects and create a scope relationship between the regular dimension and the measure dimension.

When dimensions are based on SAP BW metadata, you cannot edit the underlying query.

Steps

1. Click the regular dimension you want to modify.
2. From the **Actions** menu, click **Edit Definition**.
3. Choose the action that you want:
 - Embed calculations by selecting the level, clicking **Add**, and then defining the expression ([p. 237](#)).
 - Embed filters ([p. 239](#)).
 - Test the dimension ([p. 213](#)).
4. Click **OK**.

Hierarchies for a Regular Dimension

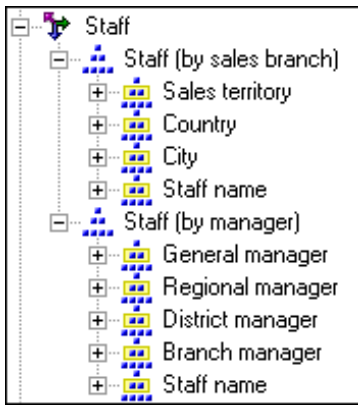
Information about hierarchies for relational metadata appears in a different topic ([p. 117](#)).

A hierarchy is an ordered list of levels or a collection of items. Each query item in a hierarchy must have a unique name.

You can specify multiple hierarchies on regular dimensions in IBM® Cognos® Framework Manager. Multiple hierarchies for a regular dimension behave as views of the same query. The first hierarchy is the primary or default hierarchy.

You can use only one hierarchy at a time in a query. For example, you cannot use one hierarchy in the rows of a crosstab report and another hierarchy from the same dimension in the columns. If you need both hierarchies in the same report, you must create two dimensions, one for each hierarchy. For more information, see "[Modeling Dimensions with Multiple Hierarchies](#)" ([p. 346](#)).

For example, sales staff can be viewed either by manager or by geography and you can model it as a single dimension with two hierarchies.



IBM Cognos software uses default settings that will not fail for the hierarchy type.

- For dimensions that represent SAP BW characteristics, the **Balanced** property is set to **true** and the **Ragged** property is set to **false**.
- For dimensions that represent presentation hierarchies, the **Balanced** property is set to **false** and the **Ragged** property is set to **true**.

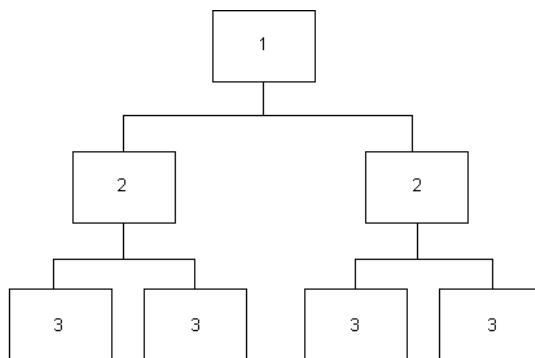
These settings may not reflect the appropriate values as IBM Cognos software does not determine the actual hierarchy structure. You can improve performance of SAP BW models and queries by adjusting dimension settings in IBM Cognos Framework Manager and in the IBM Cognos studios. For each dimension, check the settings for the **Balanced**([p. 206](#)) and **Ragged**([p. 208](#))properties to ensure that the values are set appropriately.

In addition to hierarchies in dimensions, there are hierarchies in SAP BW metadata ([p. 199](#)).

Balanced Hierarchy

Each path in a balanced hierarchy descends to the same depth.

For example, in the following diagram, the highest level is Product Line(Level 1); Level 2 is Product Type; Level 3 is Products.



In SAP BW, all leaf nodes of a hierarchy are values of the characteristic, but each path does not need to descend to the lowest level of the hierarchy.

You can define whether a dimension represents a balanced hierarchy by modifying the **Balanced** property of a dimension. The value that you set depends on the type of object that the dimension represents, and whether the hierarchy is balanced.

Dimension represents	Balanced property value
characteristic	true
presentation hierarchy that is balanced	true
presentation hierarchy that is unbalanced	false
presentation hierarchy whose structure is unknown	false

For a dimension that represents a characteristic without a presentation hierarchy, this property is read-only and is assigned a value of **true**.

If a presentation hierarchy is balanced, then set the **Balanced** property of its associated dimension to **true**. By default, it has a value of **false** for all presentation hierarchies. A hierarchy is balanced if all leaf characteristic values occur at the lowest level of the hierarchy. By setting the **Balanced** property to **true** as appropriate, the IBM Cognos BI server can generate more efficient MDX.

When all paths are of the same depth, set the **Balanced** property to **true**, otherwise set it to **false**.

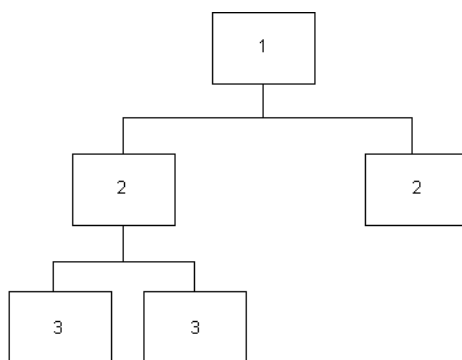
If you assign a value of **true** to the **Balanced** property of an unbalanced hierarchy, queries that involve this dimension may return incorrect data.

If you assign a value of **false** to the **Balanced** property of a balanced hierarchy, performance may be slower.

Unbalanced Hierarchy

The branches in an unbalanced hierarchy descend to different levels.

For example, in the following diagram, the highest level in an organization is the CEO (Level 1); Level 2 is the vice-presidents and the CEO's executive assistant. The executive assistant does not have subordinates, unlike the vice-presidents.



An unbalanced hierarchy can also be ragged. In a ragged-unbalanced hierarchy, there are gaps in the levels and the levels descend to different depths.

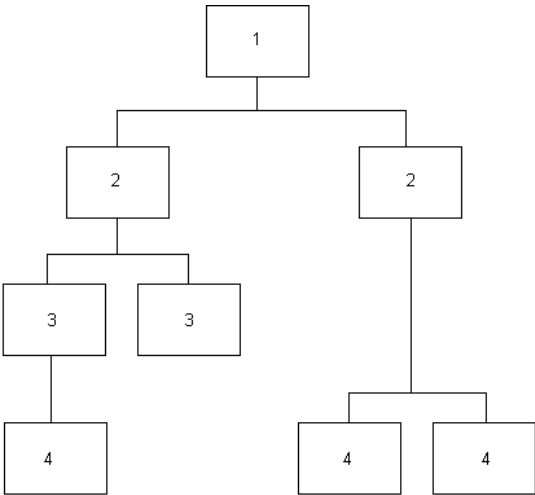
In SAP BW, this occurs only when there are "not assigned" (or "#") nodes in a presentation hierarchy. However, the presence of such a node does not ensure that the hierarchy is unbalanced. You must study the layout of a hierarchy to be certain.

An unbalanced hierarchy can also be ragged. In a ragged-unbalanced hierarchy, there are gaps in the levels and the levels descend to different depths.

Ragged Hierarchy

At least one path in the hierarchy skips at least one level.

For example, the highest level is Company (Level 1); Level 2 is Branch; Level 3 is Building; Level 4 is Department. Some branches may only have one building location, with the same departments as multi-building locations.



In SAP BW, this occurs only when there are "not assigned" (or #) nodes in a presentation hierarchy. However, the presence of such a node does not ensure that the hierarchy is ragged. You must study the layout of a hierarchy to be certain.

IBM Cognos software uses default settings that will not fail for the hierarchy type.

- For dimensions that represent SAP BW characteristics, the **Balanced** property is set to **true** and the **Ragged** property is set to **false**.
- For dimensions that represent presentation hierarchies, the **Balanced** property is set to **false** and the **Ragged** property is set to **true**.

You can define whether a dimension represents a ragged hierarchy by modifying the **Ragged** property of a dimension. The value that you set depends on the type of object that the dimension represents, and whether you know if the hierarchy is ragged.

Dimension represents	Ragged property value
characteristic	false
presentation hierarchy that is not ragged	false
presentation hierarchy that is ragged	true
presentation hierarchy whose structure is unknown	true

A dimension that represents a characteristic without a presentation hierarchy is read-only.

If a presentation hierarchy is not ragged, set the **Ragged** property of its associated dimension to **false**. By setting the **Ragged** property to **false** as appropriate, the IBM Cognos BI server is able to generate more efficient MDX

If you assign a value of **true** to the **Ragged** property of an unragged hierarchy, queries that involve this dimension may return incorrect data.

If you assign a value of **false** to the **Ragged** property of a ragged hierarchy, performance may be slower.

Limitations with Ragged and Unbalanced Hierarchies and Aggregated Values

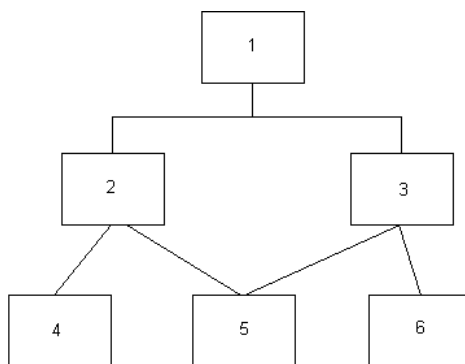
Ragged and unbalanced hierarchies can create gaps within individual paths of a hierarchy, as well as nodes at intermediate levels with no descendants at lower levels. If a report includes query items from a single dimension that are from consecutive or adjacent levels in a hierarchy, the fact values associated with the lower levels should always aggregate to the values associated with the higher levels of the dimension in the report.

However, if the query items are not from consecutive or adjacent levels, and the underlying SAP BW hierarchy is ragged or unbalanced, it is possible that the values of the higher levels may not reflect the aggregation of the fact values from the lower levels. The aggregated fact values associated with the higher levels reflect the aggregated values in the data source. This is typical behavior for OLAP data sources, but may be counterintuitive to those accustomed to reporting against relational data sources.

Network Hierarchy

A member of the hierarchy has more than one parent.

For example, an employee may report through different organizations and have multiple managers. For SAP BW, this employee will be included in the count of all employees only once, and not be included in every organization.



Levels for a Regular Dimension

A level is a collection of attributes, typically of a common granularity. Each level needs an item that is defined as a key and another item that is defined as a caption. For SAP BW data sources, levels contain members.

The first level of the hierarchy is automatically defined as the All level. It contains a single root member, which represents the top level of the hierarchy. For example, the All level for the Time

dimension is named Time (All). You cannot delete or move the All level. You can change its name, description, and screen tip.

If you do not specify the levels of the hierarchy correctly, incorrect aggregation could occur.

Member Unique Names

The member unique name (MUN) is how the member is found in the data source, much like using business keys to find records in a table.

The member unique name is used in the expression for a member data item that is used in a report, a reference to members in filters and expressions, and used in drill-through between OLAP data sources. The member keys in the MUN for the different OLAP data sources must match.

Roles

Information about roles for relational metadata appears in a different topic ([p. 122](#)).

Roles define what appears in the member tree in the IBM Cognos studios. Use roles to organize and manage metadata and to determine how to present data to your users.

You can also create expressions that refer to roles instead of query items. You must use the `roleValue` function to refer to a particular role. For example, you want to query against a specific role in a hierarchy but the query item playing that role is different at each level of the hierarchy. A single query can span the different query items at each level. You can also use the `roleValue` function when you know the role but not the underlying query item.

You can assign multiple roles to one query item, but the same role cannot be assigned to different query items in the same level.

Default roles are pre-defined for all parent-child hierarchies and for all levels in level-based hierarchies. Most of these roles are not visible in the IBM Cognos studios.

The roles that are reserved by IBM Cognos BI start with an underscore. The name for a custom role cannot start with an underscore.

Default Roles

The default roles include the following:

- `_businessKey`
Represents the key for the level. This role is also used to drill through from one data source to another because the business key should be consistent across your organization.
The `_businessKey` role can be assigned to only one attribute in a level.
- `_dimensionUniqueName`
Returns the name of the dimension as defined in the Framework Manager model.
- `_hierarchyUniqueName`
Returns the name of the hierarchy as defined in the Framework Manager model.
- `_levelLabel`
Returns the label that is assigned to the level.

- `_levelNumber`
Returns the number for the level.
- `_levelUniqueName`
Returns the name that is assigned to the level.
- `_longName`
Is assigned to the query item that represents the long name for a level.
- `_memberCaption`
Presents the caption for a member that will be shown in the IBM Cognos studios.
- `_memberDescription`
Returns the description for a member within a dimension.
- `_memberUniqueName`
Returns the IBM Cognos member unique name.
- `_parentUniqueName`
Defines the name that is assigned to the parent of the selected query item.
- `_planningDefault`
Specifies which query item to use when measures are selected. This role is applicable only for IBM Cognos Planning and SAP BW.
- `_rollupType`
Defines how a query item is aggregated.
- `_shortName`
Is assigned to the query item that represents the short name for a level.

If a query item uses a default role, you cannot change its role. This applies to SAP BW metadata only.

Custom Roles

You can create custom roles. Each role that you create must have a unique name. The roles that are reserved by IBM Cognos start with an underscore. The name for a custom role cannot start with an underscore. You can translate the custom roles in the model.

User-defined properties in OLAP data sources are assigned roles with the same name as the query item.

Specify Roles

Roles define what appears in the member tree in the IBM Cognos studios. Use roles to organize and manage metadata and to determine how to present data to your users.

Steps

1. Click the dimension whose roles you want to define.
2. From the **Actions** menu, click **Edit Definition**.
3. Click the **Dimension** tab.
4. In the **Hierarchies** box, click the level you want.
5. In the **Select a level in the hierarchy control to see the query items** box, click a query item.
6. Under **Role**, click the ellipsis (...) button.
7. Click the **Custom Roles** tab, and click **Add**.
8. Click **Close**.
9. Click **OK**.

You can also use the **Dimension Map** tab to define roles. Click **Attributes**, right-click the query item, and click **Edit Roles**.

Modify a Key Figures Dimension

Information about creating a measure dimension for relational metadata appears in a different topic ([p. 124](#)).

A key figures dimension is a collection of facts such as Quantity Sold or Price. Key figures are related to each other through the regular dimensions. When used in a report or analysis, the key figures dimension shows the value of the query item such as a name or number, or shows null, zero, or invalid. To create reports that fully compare and contrast functional areas, you may need to use more than one key figures dimension in a report.

You can add value by embedding calculations based on existing business rules, such as Profit Margin.

You can also modify the properties for multiple dimensions at the same time ([p. 34](#)).

You cannot define hierarchies or levels for a key figures dimension.

When dimensions are based on SAP BW metadata, you cannot edit the underlying query. However, you can add calculations and filters.

Steps

1. Click the key figures dimension you want to modify.
2. From the **Actions** menu, click **Edit Definition**.
3. Choose the action that you want:
 - Embed calculations by clicking **Add** and then defining the expression ([p. 237](#)).

- Change the order of measures, items, or calculations by using the arrow buttons. However, if the measure dimension contains a folder, you can change the order only in the **Project Viewer**.
 - Embed filters (p. 239).
 - Test the dimension (p. 213).
4. Click **OK**.

View Related Objects

Information about viewing related objects for relational metadata appears in a different topic (p. 127).




You can explore a visual representation of the objects that are connected to the query subject or dimension that you select in the **Project Viewer**. The **Context Explorer** shows the objects that the selected object is connected to. You can also select a connected object and see its references.

You can hide an object in the **Context Explorer**. You can also change the layout, fit all objects in the **Context Explorer**, zoom in and out, print, preview diagrams before printing, and change the page setup.

You can also use the **Dimension Map** tab to explore dimensions.

Steps

1. Select one or more objects that you want to explore.
2. From the **Tools** menu, click **Launch Context Explorer**.
3. To see the connected objects, click one or more objects and click the appropriate button.

Goal	Button
View the objects that are related to the selected object.	
View the immediate references for the objects.	
View all references for the objects.	

4. If you want to see details about an object, such as its relationships and query items, right-click the object, click **Navigate Diagram**, **Diagram Settings**, and then select the details you want.

Test a Dimension or Other Object

Information about testing relational metadata appears in a different topic (p. 128).

You can test a dimension, a level, a hierarchy, query items, or model query subject.

Testing a regular dimension returns the attributes associated with the first hierarchy encountered in the dimension.

You can see the results that an object returns by testing it. You can test when creating an object or later on. The objects you can test are dimensions, query subjects, query sets, hierarchies, levels, calculations, and query items.

You can view the data that will appear in a specific report before publishing a package by selecting and testing the objects that will appear in the report. This makes it easier to debug a model and to verify that the model meets the reporting requirements because you do not need to create and publish packages first.

When you test an object, IBM® Cognos® Framework Manager returns sample data. Formatting is not applied to the sample data. If you must test formatting, you must publish the package and view the objects in the IBM Cognos studios.

You may see different results depending on what you test. For example, if you use the expression editor to test a calculation that is embedded in a query subject, Framework Manager tests only the expression, not the item, so the aggregation setting for the query item is not applied to the test. Testing the entire query subject, which includes the calculation, gives a different result because the aggregation setting is applied. For example, if the aggregation setting is summarize, you can see a smaller number of rows in the test.

If you test a child segment of a segmented model, you may see an error if an object you are testing refers to an object in another child segment and the referenced object is not available to the project you are in. We recommend that you check that the parent model contains all the objects and that this error message does not appear when you test the parent model.

Governor settings may affect the testing results. For more information, see ["Set Governors" \(p. 304\)](#).

You can change existing test settings to customize the results that the test shows. For example, in addition to other settings, you can control the number of rows returned.

Steps When Creating or Modifying the Object

1. Select the object you want to test.
2. From the **Actions** menu, click **Edit Definition**, and click the **Test** or **Query Information** tab.
The **Test Results** box is initially empty until you run the query.
Any result sets that contain binary large objects are shown as [blob].
3. To run the query and bring back all the test results, click **Test Sample**.
4. If you are testing an expression and you want to apply the **Regular Aggregate** property of the query item or measure that is referenced in the expression, select the **Auto Sum** check box.
If you clear this check box, a row is returned for each row in the result set of the query.
5. If you want to obtain more information about the query results, click the **Query Information** tab.
6. Click **OK**.

Steps to View the Data That Will Appear in a Specific Report

1. Select the objects that will appear in the report.

2. From the **Tools** menu, click **Test**.
3. To run the query and bring back all the test results, click **Test Sample**.
4. To view details about any problem that is found, click the **Query Information** tab.

If you do not see the results of the query in the test window, the data from your data source may exceed the value of one of the governors. The query stops at the specified limit, but the test result window does not contain any data. **Tip:** Set each governor to zero.

Change the Test Settings

You can customize the tests by changing the test settings.

Steps

1. Select the object that you want.
2. From the **Actions** menu, click **Edit Definition** and then click the **Test** tab or the **Query Information** tab.
3. Click **Options** and then click the **Test Settings** tab.
4. Choose the options that you want.

Goal	Action	Persistence
Retrieve all data and show a specified number of rows	<p>Select the Restrict the maximum number of rows to be returned check box and type the required number of rows.</p> <p>This setting does not improve performance for retrieving data when testing dimensions, query subjects, and query sets.</p>	<p>This setting applies to all dimensions, query subjects, and query sets in the model.</p> <p>This setting is saved and used in your next session with any model.</p>
Specify the level of detail	<p>Drag the Level of Information shown in Results Information slider to the location that represents the amount of detail you require.</p>	<p>This setting is saved and used in your next session with this model.</p>
Temporarily override session parameters	<p>In the Session Parameters box, click Set.</p> <p>The Session Parameters dialog box appears.</p>	<p>The override values are not saved with the model. This setting is for your current session only.</p>

Goal	Action	Persistence
Apply relevant design mode filters	<p>Select the Apply all relevant design mode filters when testing check box.</p> <p>This applies all relevant filters whose usage is set to design mode in another dimension, query subject, or query set.</p>	This setting is saved and used in your next session with any model.
Apply a security filter	In the Security Filters box, click Edit .	This setting is saved and used in your next session with this model.
Change the prompt values	<p>In The Current Prompt Values box, click Prompts.</p> <p>The Model Prompts Manager dialog box appears, which shows all prompts, and their values, that are in the model.</p>	<p>The prompt values are not saved with the model.</p> <p>This setting is for your current session only.</p>

5. Click **OK** two times.

You may be interested in the following related topics:

- working with dimensions ([p. 204](#))
- working with query subjects ([p. 216](#))

Working with Model Query Subjects

Information about query subjects that are based on relational metadata appears in a different topic ([p. 85](#)).

A query subject is a set of query items that have an inherent relationship.

You use IBM® Cognos® Framework Manager to modify query subjects to optimize and customize the data that they retrieve. For example, you can add filters or calculations. When you change the definition of a query subject, Framework Manager regenerates the associated query items, ensuring that any changes to query subject properties are reflected in all query items for that query subject.

For SAP BW metadata, you can work with model query subjects in IBM® Cognos® Framework Manager.

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.

If you create a model query subject containing multiple dimensions, you may encounter problems when using the model query subject in conjunction with other query subjects or dimensions. Ensure that the items in a model query subject do not contravene the logic of the model, for example, the product item inserted between the country and city items. Test the model query subject in a report; if grouping works, the model query subject is valid.

Create a Model Query Subject

Information about query subjects based on relational metadata appears in a different topic ([p. 85](#)). You can also create a new model query subject by merging existing query subjects and query items ([p. 217](#)).

Steps

1. Select the namespace folder and, from the **Actions** menu, click **Create, Query Subject**.
2. In the **Name** box, type a name for the new query subject.
3. Click **Model** and click **OK**.

Note: For SAP BW metadata, you can only create model query subjects.

4. Click the **Query Subject Definition** tab.
5. To add items to the model query subject, drag items from the **Available Model Objects** box to the **Query Items and Calculations** box.

You can change the order of items and calculations. However, if the query subject contains a query item folder, you can change the order only in the **Project Viewer**.

6. To embed calculations in the model query subject, click **Add** and define the calculation.
7. To embed filters in the model query subject, click the **Filters** tab.
8. To test the model query subject, click the **Test** tab.
9. Click **OK**.

A warning appears if any modifications invalidated relationships, other query subjects, calculations, or filters.

You may be interested in the following related topics:

- embedded calculations ([p. 237](#))
- embedded filters ([p. 239](#))
- testing and setting test options ([p. 213](#))
- modifying the properties for multiple query subjects at the same time ([p. 34](#))

Create a Model Query Subject Based on Existing Objects

Information about model query subjects based on relational metadata appears in a different topic ([p. 96](#)).

You can select existing model objects and merge them into a new model query subject. This means that you can reuse existing metadata to quickly create query subjects.

The objects that you can merge include

- model query subjects and their shortcuts
- query items, filters, and calculations in model query subjects

You can merge any number of the same type of objects into a new query in a single operation. The merge always creates a new model query subject.

The new query subject contains any filters that exist in the original query subject.

Notes

- Ensure that model query subjects do not contravene the logic of the model. For example, if a query subject with multiple characteristics is used in combination with other query subjects, there can be problems when you run the report.
- Do not include query items from different query subjects or hierarchies from the same dimension. This causes a run-time error.

Steps

1. Ctrl+click the objects that you want to merge into a single query subject.
2. From the **Actions** menu, click **Merge in New Query Subject**.

Validate a Model Query Subject

Information about validating relational query subjects appears in a different topic ([p. 104](#)).

You can validate the definition of the query subject without having to open the **Query Subject Definition** dialog box. This is useful to do when

- new query items were added to a query subject
- the definition of the underlying query subject has changed

The **Evaluate Object** command evaluates the selected objects and ensures that they can run.

When IBM® Cognos® Framework Manager evaluates a query subject, a request is sent to the SAP BW data source. Physical attributes, such as data type, are then updated as needed for the query subject.

You can also synchronize the entire project ([p. 301](#)).

Steps

1. Select the query subject you want to evaluate.
2. From the **Tools** menu, click **Evaluate Object**.

If you changed the **Regular Aggregate** property to **unsupported**, the property is reset when you evaluate the query subject. If the property is set to any other value, the property is not changed.

Note: An error message is displayed for each invalid query subject. The object will also have a status of Invalid.

Query Items

Information about query items for relational metadata appears in a different topic ([p. 138](#)).

A query item is the smallest piece of the model that can be placed in a report. It represents a single instance of something, such as the date that a product was introduced.

Key figures and attributes are imported as query items in IBM® Cognos® Framework Manager.

Only one hierarchy from a dimension should be used in the same report.

For SAP BW metadata, you can modify only text-based properties, such as the name or screen tip.

Because reports can contain different query items from one or more objects in the model, query item properties control many aspects of the final report. When you create a model dimension or model query subject, the query items inherit the properties of the data source query items on which they are based.

The properties for query items or measures include the following.

Query item property	Description
Name	The name of the query item or measure.
Description	A description of the query item or measure.
Last Changed	The date that the query item or measure was last changed. The property is automatically updated with the current date time.
Last Changed By	The user who last changed the query item or measure. This property is automatically updated when the item is changed. The value is the current logon username.
Model Comments	Used to add internal comments about the model. The information is used on the Analyze Publish Impact dialog and in the Model Report . Comments are not accessible to package users.
Screen Tip	A description that can appear in the published package for your users.
Expression	Used to create embedded calculations that provide your users with calculated values that they regularly use. This property is for measures only. Note: The Expression property is not used by SAP BW.

Query item property	Description
External Name	The name that appears in the data source.
Is Hidden	<p>Whether to hide or show the query item or measure in the published package.</p> <p>Even when Is Hidden is set to True and the query item or measure is invisible to your users, it is always present in the published package because the query item or measure may be needed by other objects in the model. You do not see the query item or measure in the Package Publish wizard.</p> <p>For example, a calculation may make use of a hidden query item.</p>
Usage	<p>The intended use for the data represented by the query item.</p> <p>This property is for query items only.</p>
Format	How information appears in a report.
Currency	<p>Which currency is used.</p> <p>This property cannot be changed in the Property pane. Use the Format property to change the currency.</p>
Data Type	<p>The data type that was set in the data source.</p> <p>Because this property is set in the data source, it is read-only in Framework Manager.</p>
Precision	<p>The total number of digits.</p> <p>Because this property is set in the data source, it is read-only in Framework Manager.</p>
Scale	<p>How many digits are represented in the scale.</p> <p>For example, you can show numbers in thousands so that 100,000 means 100,000,000.</p> <p>Because this property is set in the data source, it is read-only in Framework Manager.</p>
Size	<p>The size of the query item or measure.</p> <p>Because this property is set in the data source, it is read-only in Framework Manager.</p>

Query item property	Description
Is Nullable	Whether the query item or measure can contain a null value. Because this property is set in the data source, it is read-only in Framework Manager.
Display Type	How the query item is shown. The column value can appear in the IBM Cognos studios as a picture, as a link, or as a value. The default is value. This property is for query items only.
MIME Type	The format that the column value uses. For example, if Display Type is set to picture, MIME Type could be jpeg. This property is for query items only. Note: The MIME Type property is not used by SAP BW.
Prompt Info	Prompt behavior.
Regular Aggregate	The type of aggregation that is associated with the query item, measure, or calculation in the published package.
Aggregate Rules	For dimensionally modeled relational metadata, the rules for semi-additive aggregation. For SAP BW metadata, the Aggregate Rules property is read-only.
Allocation Rule	Specifies the type of allocation defined for the measure. A value of default specifies that constant allocation is used in list queries and once-only allocation is used in crosstab queries. A value of constant specifies that constant allocation is used in all queries.
Is Unsortable	Whether the values of this query item can be sorted. This property is for query items that contain large objects such as BLOBs.

You may be interested in the following related topics:

- the Usage and **Regular Aggregate** properties ([p. 222](#))
- changing the currency symbol ([p. 225](#))

- prompts (p. 226)
- modifying the properties for multiple query items at the same time (p. 34)

Modifying How Query Items Are Aggregated

Information about aggregation of relational query items appears in a different topic (p. 141).

You can change how some query items and measures are aggregated in reports. IBM® Cognos® Framework Manager applies aggregate rules when your users create a report that summarizes a query item or measure.

When you import metadata, Framework Manager assigns values to the **Usage** and **Regular Aggregate** properties for query items and measures depending on the type of object that the query item or measure is in. The **Usage** property identifies the intended use for the data represented by the query item (p. 144). The **Regular Aggregate** property identifies the type of aggregation that is applied to the query item or measure (p. 145). Your users can override the values of the **Regular Aggregate** property. For semi-additive measures, you can specify additional aggregate rules by modifying the **Aggregate Rules** property (p. 146).

When modifying the **Regular Aggregate** property, you can select values that are not available through importing, such as average and maximum. You must understand what the data represents to know which aggregate rule is required. For example, if you aggregate a part number, the only aggregate values that apply are count, count distinct, maximum, and minimum.

Rules for Setting Properties for Dimensions

IBM® Cognos® Framework Manager uses the following rules to set the **Usage** and **Regular Aggregate** properties.

Object	Usage property	Regular Aggregate property
Query item in a regular dimension	Attribute	Unsupported
Query item in a measure dimension	Identifier	Count
Measure in a measure dimension	Fact	Automatic if the measure is a calculation Sum if the measure is not a calculation

If the measure is semi-additive, use the **Aggregate Rules** property to define rules for semi-additive aggregation (p. 146).

For SAP BW metadata, you cannot change these properties for dimensions.

Rules for Setting Properties for Calculations

The **Regular Aggregate** property for a calculation in SAP BW metadata is set to **Automatic**. To determine what automatic means, these rules apply.

Calculation	Aggregation type
key items	unsupported
all other items	calculated

Rules for Setting Properties for Model Query Subjects

For model query subjects, Framework Manager uses the settings for the object that the model query subject is based on.

Note: If you change an aggregation value for SAP BW metadata, the aggregation cannot perform time-based queries because the aggregation rules are not applied.

Usage Property

The **Usage** property identifies the intended use for the data represented by each query item. During importing, the **Usage** property is set according to the type of data that the query items represent in the data source.

You need to verify that this property is set correctly. For example, if you import a numeric column that participates in a relationship, the **Usage** property is set to **identifier**. You can change the property.

For SAP BW query items, the value of the **Usage** property depends on the type of dimensional item the query item is based on.

Usage property value	SAP BW object	Description
Identifier	hierarchy level	Uniquely identifies characteristic values at a particular level in a hierarchy.
Fact	key figure	Represents a key figure that typically is numeric data. Date and time data are also supported.
Attribute	display attribute	Represents a display attribute that is associated with a characteristic.

Regular Aggregate Property

The **Regular Aggregate** property identifies the type of aggregation for the query item or calculation when you publish it. Your users can either use this default setting to perform calculations on groups of data, or apply a different type of aggregation.

For example, if the **Regular Aggregate** property value of the Quantity query item is sum, and it is grouped by Product Name in a report, the Quantity column in the report shows the total quantity of each product.

The following aggregation types are supported for SAP BW data sources:

- automatic
- average
- average non-zero is supported only when it is set in the data source. You cannot change the property to average non-zero in Framework Manager.
- calculated
- count
- count distinct
- count non-zero is supported only when it is set in the data source. You cannot change the property to count non-zero in Framework Manager.
- maximum
- median
- minimum
- standard deviation
- sum
- variance

Rules to Determine the Automatic Aggregation Type

If the calculation is in an SAP BW object, these rules apply.

Calculation	Aggregation type
key items	unsupported
all other items	calculated

Aggregate Rules Property

For dimensionally modeled relational metadata, the method by which a semi-additive measure is aggregated for the dimensions that you select.

For SAP BW metadata, the **Aggregation Rules** property is read-only. The **Semi-Aggregate** property is used instead.

Semi-Aggregate Property

For SAP BW metadata, the **Semi-Aggregate** property shows the value that is set in the data source, and the property is read-only.

If the value is set to **unsupported** in IBM® Cognos® Framework Manager, the semi-aggregate behavior is ignored in the IBM Cognos studios.

The **Semi-Aggregate** property will not be supported in future releases. Instead, use the **Aggregate Rules** property for semi-additive measures.

Format Query Items

Information about formatting relational items appears in a different topic ([p. 148](#)).

You can specify how query item values appear in reports. Use the **Format** property to choose a format type, such as text, date, and currency. Each format type contains properties that further specify how the data appears.

For example, you can assign the **Currency** format type to a numeric query item, and then use the **No. of Decimal Places** property in the **Data Format** dialog box to specify how many decimal places appear in reports.

Some characters are language-sensitive and appear properly only when your locale supports the applicable font. For example, for Japanese currency symbols to appear correctly, your locale must be set to Japanese.

If IBM® Cognos® Framework Manager does not show the currency you require, you must ensure that you install the appropriate language packs to support the currency symbols. For example, to have the Indian currency symbol (rupee) appear, you must run an operating system or install a language pack that can show this symbol. The Japanese operating system or Japanese language is one that can show the Indian currency symbol.

You can define properties for several query items at the same time. However, if the query items have different format types, all properties that were previously specified are overridden and the default values from the data source are used. If the original format types of the selected query items are the same, all the properties for the selected query items are set identically.

For example, to use the same decimal separator for two query items and to keep the number of decimals different, each query item must be changed individually. If both are selected and changed at the same time, all properties including the number of decimals are set identically for both query items.

Steps

1. In the **Project Viewer** pane, click the query item you want to format.
2. In the **Properties** tab of the **Properties** pane, click the **Format** property.
3. Set the format type to currency to ensure that currency formatting is applied to all types of reports.
4. In the **Currency scope** box, specify the type of currency. If you do not see the currency you want to use, click the **Add** button.

5. In the **Properties** box, select or type the appropriate property value.
6. Click **OK**.

Define a Prompt Control

Information about prompt controls for relational metadata appears in a different topic([p. 149](#)).

Prompts help your users quickly find the information they need in a report. Prompts are generally defined in reports. However, you can change the behavior of prompts in the studios by modifying the definition of dimensions or query subjects in the model.

This is useful for query items, such as ProductTypeCode, whose values are not shown in a report but are useful for filtering data. In general, it is better to define type-in prompts in the reports to make use of the additional prompt features. However, your users cannot modify some variables. For these variables, you can use IBM® Cognos® Framework Manager instead of the reports to define type-in prompts.

The Prompt Info properties set in Framework Manager give you the ability to control default filtering and prompting. The properties are used by:

- Query Studio to create a filter expression and set the use and display items in a prompt and prompt query
- the Build Prompt Page tool in Report Studio to create a filter expression and set the use and display items in a prompt and prompt query
- generated prompts in Report Studio to set the use and display items in the prompt and prompt query

The syntax for using a prompt as a value is

?<PromptName>?

You can use prompts in

- parameter maps
- session parameters
- expressions, including filters and calculations

Steps

1. Click the query item.
2. In the **Properties** pane, click the **Properties** tab.
3. Click the plus sign (+) next to the **Prompt Info** property.
This is a compound query item property.
4. Modify the following properties to reflect the behavior you require.

Goal	Property
Set the type of prompt control that is generated when the report is run.	Prompt Type
Set the generated prompt as part of a series of generated cascading prompts.	Cascade On Item Reference
Specifies which query item is displayed to the report user in the prompt.	Display Item Reference
The values in the prompt are data values of the query item.	
Each value in the prompt is associated with a value in the query item specified in the Use Item Reference property.	
Specifies which query item is passed from the prompt to the filter.	Use Item Reference
Each value is associated with a value in the query item specified in the Display Item Reference property.	
Specifies which query item is used in the filter expression to retrieve data.	Filter Item Reference

Prompt Type Property

The **Prompt Type** property sets the type of prompt control that is generated when the report is run, such as an edit box or a pull-down list.

The default value for this property is **Server Determined**.

Note: Prompt types set on attributes are now processed. The report user will see the prompt that matches the prompt type on the attribute. Because prompt types on attributes were not processed in the previous release, some differences may occur.

Value	Prompt Control
Server Determined	The type of prompt control is based on information in the server, such as the data type of the query item.
Edit Box	A simple text box. If the data type of the column is date or dateTime, this value generates a date or date-time control as well as the text box.
Select Date	A date control with a calendar interface.

Value	Prompt Control
Select Date/Time	A date-time control with a calendar interface. For SAP BW metadata, this value is not relevant.
Select Interval	A date-time interval control. For SAP BW metadata, this value is not relevant.
Select Time	A time control that filters data based on the selected time period. For example, if you define a Select Time prompt for Order Time, the user can use the time control to show all orders placed after 1:00, or all the orders placed between 10:00 and 11:00. If you are referring to a time member, you must use the exact values only. If you are using a range, the end points of the range must correspond to values in the data source.
Select Value	A drop-down list.
Select with Search	A list control so that users can search for values. For SAP BW metadata, this value is not relevant.
Select with Tree	A tree prompt control for prompts that are based on a hierarchy node.

Note: If the caption is a different datatype than the business key (MUN) for the level, use the **Filter Item Reference** in conjunction with setting the **Prompt Type** for the caption. This ensures that the right datatype is used when filtering in the studios.

Cascade On Item Reference Property

The **Cascade On Item Reference** property indicates that the generated prompt is part of a series of generated cascading prompts. The query item that you reference in this property is the parent item in the cascade. The system prompts the user for the cascade item before prompting them for the current query item.

For example, if you want to prompt for Product Line and then Product within the selected line, set the **Cascade On Item Reference** property of the Product query item to Product Line.

Display Item Reference and Use Item Reference Properties

The **Display Item Reference** property specifies which query item is displayed to the user in the prompt. The **Use Item Reference** property specifies which query item is passed from the prompt to the filter. Each value in the list of display items is associated with a value of the query item specified in the **Use Item Reference** property.

For example, you want the prompt to display Country Name while using Country Code to retrieve data. Set the **Display Item Reference** property to Country Name and the **Use Item Reference** property to Country Code. The prompt for Country Name makes it easy for the report user to select required values. However, using the Country Code in the filter is more efficient for data retrieval.

These properties are used by

- Query Studio to create a filter expression and set the use and display items in a prompt and prompt query
- the Build Prompt Page tool in Report Studio to set the use and display items in a prompt and prompt query
- generated prompts in Report Studio to set the use and display items in the prompt and prompt query

Note: The values of the **Use Item Reference** and **Filter Item Reference** properties must be compatible. Otherwise, the report user may receive unexpected results. For more information, see the **Filter Item Reference** property (p. 153).

Default: If no values are set, the properties default to the name of the query item.

These properties are used only for data driven prompt controls whose **Prompt Type** property is set to either **Select Value** or **Select with Search**.

Filter Item Reference Property

The **Filter Item Reference** property identifies the query item used when Report Studio or Query Studio generates a filter. This property can help create more efficient queries by ensuring that a filter uses an indexed numeric column rather than a non-indexed string column.

For example, a report author wants to create a filter for the Country Name query item. You set the **Filter Item Reference** property to use Country Code instead of Country Name for any filter that uses the Country Name query item.

In another example, a report author wants to create a filter for the Country Code query item that appears in the Orders table. You want that filter to use the Country Code in the Country table because there are fewer rows to read in the Country table so you set the **Filter Item Reference** in the model to `Country.Country Code`.

This property is used by:

- Query Studio to create a filter expression
- the Build Prompt Page tool in Report Studio to create a filter expression

Default: If no value is set, the property defaults to the name of the query item.

Using the Filter Item Reference and Use Item Reference Properties

The values of the **Filter Item Reference** and **Use Item Reference** properties must be compatible. The value of the **Use Item Reference** property must be a type that is expected by the **Filter Item Reference** property. Otherwise, the report user may receive unexpected results. This may occur when a report user creates a filter without creating a prompt page.

In an example model, the **Use Item Reference** property is set to Employee Number and the **Filter Item Reference** property is Employee Name. In Report Studio, a report author creates the following filter without creating a prompt page:

```
Reference.EmployeeName in ?parml?
```

Report Studio automatically generates prompts when you create a filter without creating a prompt page. Because the prompt is generated, Report Studio uses the Prompt Info properties from the Employee Name query item in the Framework Manager model.

The **Use Item Reference** indicates that the values being passed to the filter are employee numbers. The **Filter Item Reference** is filtering data based on Employee Name. The filter is as follows:

```
Reference].[Employee Name] in ("1", "2").
```

Since there are no Employee Name values of "1" or "2", the report will be blank.

Using Filter Item Reference for Dimensionally Modeled Relational Metadata

For dimensionally modeled relational metadata, **Prompt Info** is specified on the attribute with the role of `_memberCaption`, instead of the level. Although set on the attribute, the **Prompt Info** properties are processed as if they were on the level. By default, when the level is included in a report, users are prompted to enter MUNs in the level's prompt. To enter caption values instead, set the attribute's **Filter Item Reference** property to itself. When the prompted filter is applied, the filtered values will be based on the attribute values.

For example, the level Product Line has an attribute of Product Line with a role of `_memberCaption`. If the **Filter Item Reference** property value is set to Product Line, report users are prompted to enter Product Line values. If the **Filter Item Reference** property value is left blank, users are prompted to enter MUNs.

Note: Do not use the **Filter Item Reference** property with the Select with Tree prompt type. Because a Select with Tree prompt can only filter on a level or hierarchy, setting the **Filter Item Reference** property will cause an error.

Testing a Prompt

When you test a model object that references a prompt, IBM® Cognos® Framework Manager asks you to enter the prompt value. Framework Manager uses this value for either the duration of the session, or until you clear the prompt value.

You can change the session value of prompt values through the **Options** dialog box. This dialog box is available when you modify a dimension or query subject, or define a calculation, filter, query set, or complex relationship. You can change the prompt value at the time that you are testing the expression that references that value.

If you select the **Always prompt for values when testing** check box in the **Prompt** dialog box, Framework Manager prompts you for a value every time you test the object. When updating the object or performing a count, Framework Manager uses the existing prompt value, if one exists.

A prompt on a query item in a model query subject is associated only with that query item. A prompt on a query item in a data source query subject is associated with the entire query subject and therefore, the prompt appears when you test any query item in the query subject.

SAP BW Variables

SAP BW variables are parameters of an SAP BW Query that are set up during query definition. When you run the query, the SAP BW variables are filled with values. They function as placeholders and can be processed in different ways. They are automatically exposed as prompts at run time.

SAP BW variable information is included in a composite custom property named **SAP BW Variables** that exists only if a data source has one or more variables associated with it. The **SAP BW Variables** property contains one or more composite properties, each of which must be assigned a unique name. Each property represents a description of a single SAP BW variable. Because the variable information is specified in a custom property, Framework Manager does not validate these properties.

The SAP BW variable information is obtained using the SAP BW BAPI *MDDDataProviderBW::GetVariables*.

Framework Manager supports these types of SAP BW variables:

- characteristic

There are two kinds of characteristic variables: characteristic value and hierarchy node. Characteristic values variables select characteristic values. Hierarchy node variables select values from any position in a presentation hierarchy.

- hierarchy

The user is not prompted for a value because IBM Cognos software automatically populates it at run time based on the selected hierarchy. Variables for hierarchies function act as placeholders for the hierarchy of a characteristic. All the values for hierarchy variables are read-only.

- formula

The user types a numeric value at run time. Use formula variables if a formula component should be entered only when a query is run. For example, you can use a formula variable for a value-added tax rate to process the current rate at run time.

- authorization

Authorization variables are like other variables, but IBM Cognos software automatically populates the variable values with the user's credentials. SAP BW uses these credentials to supply the information needed by an SAP BW Query that has security applied to it.

Variables for hierarchies function as placeholders for the hierarchy of a characteristic. All the values for hierarchy variables are read-only.

Name Property

This property is a string value.

SAP BW equivalent: `VARIABLE_NAME`

Restrictions: Read-only.

Caption Property

The string value for this property is a composite and is locale-dependent. Represent each locale in the model by a custom property whose value is the locale name. For example, if the locales en-ca and fr-fr exist in the model, define two custom properties named en-ca and fr-fr.

The default value is obtained from SAP BW.

Default Low Caption and Default High Caption Properties

The value for each of these properties is a composite, locale-dependent string value. Represent each locale in the model by a custom property whose value is the locale name. For example, if the locales en-ca and fr-fr exist in the model, define two custom properties named en-ca and fr-fr.

The default value is obtained from SAP BW.

Restrictions: The **Default High Caption** property is applicable only for variables with a **Selection Type** of **interval**.

Selection Type Property

The possible values are value, interval, complex, multiValued.

Value	SAP BW Equivalent
value	SAP_VAR_SEL_TYPE_VALUE
interval	SAP_VAR_SEL_TYPE_INTERVAL
complex	SAP_VAR_SEL_TYPE_COMPLEX
multiValued	SAP_VAR_SEL_TYPE_COMPLEX

Restrictions: Read-only.

Entry Type Property

The default value is obtained from SAP BW.

Value	SAP BW Equivalent
optional	SAP_VAR_INPUT_TYPE_OPTIONAL
mandatory	SAP_VAR_INPUT_TYPE_MANDATORY
mandatoryNotInitial	SAP_VAR_INPUT_TYPE_MANDATORY_NOT_INITIAL

Restrictions: Read-only.

Default Low Value and Default High Value Properties

Each of these properties specifies a range of values.

The default value is obtained from SAP BW.

Restrictions: The **Default High Value** property is applicable only for variables with a **Selection Type** of **interval**.

Description Property

This property is a string value.

SAP BW Variable Type Property

The possible values are numeric, characteristic, hierarchy, or hierarchicalNode.

The default is obtained from SAP BW.

Restrictions: Read-only.

Prompt Type Property

The default value depends on the variable type. If the value of this property is not one of the predefined values, the value used is hierarchyPickList. The predefined values for the Prompt Type property are as follows.

Value	Restrictions
typeIn	Required for numeric variables and optional for characteristic values
pickList	Optional for characteristic variables
calendar	Only for characteristic variables based on 0CALDAY
hierarchyPickList	Optional for all presentation hierarchies
notApplicable	Required for hierarchy variables

Use this property to specify the type of prompt.

You can improve the performance of variable prompts that use either a picklist or hierarchical picklist. Use the Level Restriction, Initial Number of Picklist Values, and Use Default Value properties to control the performance of those types of variable prompts.

Changing a picklist or hierarchical picklist prompt to a type-in prompt can dramatically improve performance because it does not require the application server to populate a picklist with values. However, it requires your users to be able to accurately enter characteristic values.

Restrictions: Read-only for some types of variables such as characteristic and formula.

Level Restriction Property

This property is a numeric value.

The default value is 1.

Use this property to reduce the number of characteristic values that populate a hierarchical picklist. There is a limited number of levels of a hierarchy from which values are obtained.

If the value is zero (0), which is the default, then characteristic values from all levels of a hierarchy (if applicable to the type of prompt) populate the picklist. Otherwise, the property specifies a colon-separated range of levels from which values are obtained (the root level is zero).

For a ragged hierarchy, you must specify all levels that you may want to use even if some branches do not have that level.

Restrictions: Applicable only for hierarchical node variables with a **Prompt Type** of **hierarchyPickList**.

Trim Levels Property

This property is a string value that reduces the number of members in a hierarchical picklist. If the property is set to zero (0), members from all levels of a hierarchy are included in the prompt. You can also specify a range such as 2:4 to include only the members from certain levels. If the starting and ending ranges are the same, such as 3:3, only members from that level are included.

The default value is zero (0).

Restrictions: Applicable only for characteristic variables with a **Prompt Type** of **hierarchyPickList**.

Use Default Values Property

This property is a boolean property that determines whether the default values are used. If this property is set to **true**, users are not prompted for the associated variable, and the default value is always applied.

Use this property to set the variable to a single value. Users are not prompted for the value of a variable and consequently, the IBM Cognos BI server does not populate a picklist with values. However, users can no longer change the value of a variable.

The default value is **false**.

Show Key and Caption Property

To show keys and captions for SAP BW variables, set this property to **true**. This property is applicable only for pick list prompts and hierarchy node prompts.

The default value is **false**.

Initial Number of Pick List Values Property

This property specifies the initial number of values used to populate a picklist, hierarchical picklist, or prompt.

The default value is zero (0), which means all.

Numeric Variable Property Values

The following variable properties are applicable to numeric variables:

Property	Default value
Type	numeric
Caption	
Selection Type	value

Property	Default value
Entry Type	obtained from SAP BW
Default Low Value	
Default High Value	
Prompt Type	typeIn
Use Default Value	false

You can change the default values for a numeric variable except for the **Prompt Type** property, which is read-only.

Characteristic Variable Property Values

There are two kinds of characteristic variables: characteristic value and hierarchy node. Characteristic values variables select characteristic values. Hierarchy node variables select values from any position in a presentation hierarchy.

Characteristic Value Variable Property Values

The following variable properties are applicable to characteristic value variables:

Property	Default value
Type	characteristic
Caption	
Selection Type	obtained from SAP BW
Entry Type	obtained from SAP BW
Default Low Value	<p>If the entry type is value or complex, the default property is shown.</p> <p>If the entry type is interval, the default low property is shown. This value is obtained from SAP BW.</p>
Default High Value	<p>If the entry type is value or complex, the default property is shown.</p> <p>If the entry type is interval, the default high property is shown. This value is obtained from SAP BW.</p>

Property	Default value
Prompt Type	typeIn or pickList This depends on the number of members in the referenced dimension. If the value is invalid, typeIn is used.
Use Default Value	false
Show Key and Caption	false
Initial Number of Picklist Values	zero (0)

A characteristic value variable for the 0CALDAY dimension is shown in the model as a date. The **Data Type** property is set to **xsdDate** and the **Prompt Type** property is set to **calendar**. The **Prompt Type** property is read-only for the 0CALDAY dimension.

Hierarchy Node Variable Property Values

The following variable properties are applicable to hierarchy node variables:

Property	Default value
Type	characteristic
Caption	
Selection Type	obtained from SAP BW
Entry Type	obtained from SAP BW
Default LowValue	
Default HighValue	
Prompt Type	hierarchy PickList You can change the Prompt Type property to typeIn or pickList .
Level Restriction	zero (0)
Use Default Value	false

Picklist Prompts

Each picklist prompt contains a pre-defined number of values. These values are determined by the **Maximum Number of Values** property.

If the number of actual values is less than or equal to the default number of values, the prompt is generated as a single picklist prompt. If the number of actual values exceeds the default number, two prompts are generated in this order:

- a bound range parameter with a starting value of **1** and an ending value determined by the **Maximum Number of Values** property

This parameter is of the type `xsdUnsignedLong` and is optional. The name of the parameter is the name of the original prompt followed by `_range_prompt`. The caption is locale-specific. If this is a multilingual model, you must store the template for the caption in a message file.

- a picklist prompt containing the default number of values

Adding Business Rules

Information about business rules for relational metadata appears in a different topic ([p. 155](#)).

Business rules that were created in SAP BW are imported into IBM® Cognos® Framework Manager. You can add more business rules to your model to refine the retrieved data and to ensure that the right information is available for your users.

Creating business rules and storing them in the model instead of in reports has many advantages. You save time because you and your users do not have to re-create the business rules every time they are needed. The business rules ensure consistency because your users all use the same definitions. For example, Low Margin means the same thing throughout the organization. They are easy to update because you maintain the business rules centrally so that all reports are updated automatically as the rules evolve. For example, if the definition for Low Margin changes, all reports that use the Low Margin calculation are updated automatically. The business rules enhance security. For example, you can use filters to limit the data that your users can see.

You can

- ☐ add calculations so that your users can include calculated data in their reports ([p. 237](#)).
- ☐ create and apply filters so that you can limit the data that a query subject retrieves ([p. 239](#)).
- ☐ add prompts that will automatically appear whenever a dimension or query subject is used in a report; report consumers are then prompted to filter data ([p. 226](#)).
- ☐ use session parameters ([p. 245](#)) and parameter maps ([p. 243](#)) to dynamically resolve expressions.
- ☐ create a security filter to control the data that is shown to your users when they set up their reports ([p. 257](#)).

Create a Calculation

Information about calculations for relational metadata appears in a different topic ([p. 155](#)).

You can create calculations to provide your users with calculated values that they regularly use. Calculations can use query items, parameters, variables, calculated members, expressions, and expression components, such as functions.

Punctuation characters, such as the question mark (?), must be in 7-bit ASCII character code. If you type a punctuation character from a multi-byte enabled keyboard, ensure that you type the 7-bit ASCII representation of the character. For example, type Alt+063 for the question mark.

Avoid using characters that are used for expression operators in the name of the calculation. Syntax errors may occur when the expression is evaluated. For example, a calculation named Margin * 10 causes errors when used in an expression such as [Margin * 10]< 20.

In expressions, an operator or function may require operands to be of a particular dimensional type. When an operand is not of the required type, one or more coercion rules may be applied to coerce the operand to the appropriate type. Because coercion rules are not applied to expressions in model query subjects, ensure that those expressions are valid without relying on coercion rules. For more information about coercion rules, see the IBM® Cognos® Report Studio *User Guide*.

If you insert an imported user-defined function in the calculation, ensure that the function name does not repeat vendor-specific names. For example, if the user-defined function name is CHAR you will receive an error when testing the function in the **Calculation Definition** dialog box because this name is considered identical as **char** in Microsoft® SQL Server. For information about function names used in your database, see the database product documentation.


At query time, IBM® Cognos® Framework Manager returns a null value for any calculation that contains a divisor whose value is zero. Framework Manager cannot detect zero-division errors in functions such as average and mod, because the division operator is not explicit.


You can apply a stand-alone calculation to one or more dimensions or query subjects to provide calculated data to a report, or include it in a package to make it available to your users. By moving a stand-alone calculation or a shortcut to it into a folder, you can better organize the model objects.

Steps to Create a Calculation

- 1. Click the namespace or folder and, from the **Actions** menu, click **Create, Calculation**.
- 2. In the **Name** box, type a name for the calculation.
- 3. Define the expression.

Goal	Action
Add items	On the Model tab, click a query item, filter, or calculation and click the arrow.
Add functions	On the Functions tab, choose a component and click the arrow.
Add parameters	On the Parameters tab, click a parameter and click the arrow.

Goal	Action
Retrieve all data and show a specified number of rows	<p>Click the options button, select the Restrict the maximum number of rows to be returned check box, and type the required number of rows to be returned.</p>  <p>This setting does not improve performance for retrieving data when testing dimensions and query subjects.</p>
Override session parameters	Click the options button, click Set , enter a value in the Override Value field, and click OK .
Override prompt values	<p>Click the options button, and then click Prompts.</p> <p>The Model Prompts Manager dialog box appears, which shows all prompts, and their values, that are in the model.</p>

4. To test the calculation, click the **test** button .

You can test only calculations that contain query items. If a calculation contains a function, for example `_add_days`, the **Test Sample** button is not available.

5. Click **OK**.
6. Modify the **Data Type** property to identify the type of data the calculation returns.

The IBM Cognos studios use this information to format the data that the calculation returns.

For information about functions, see ["Using the Expression Editor" \(p. 381\)](#).

Create a Filter

Information about filters for relational metadata appears in a different topic ([p. 158](#)).

A filter is an expression that specifies the conditions that rows or instances must meet to be retrieved for the dimension, query subject, calculation, or report to which the filter is applied. A filter returns a boolean value so that you can limit the rows returned by a dimension or query subject.

For example, you can use the `in_range` function to create a filter that retrieves data for products introduced in a specific time frame. The syntax for this example looks like this:

```
[gosales_goretailers].[Products].[Introduction
date]
in_range {Feb 14, 1999 : July 14, 2007}
```

Note: When using a date or time function, you must use a 24-hour clock. IBM® Cognos® Framework Manager does not support "a.m." or "p.m." in expressions. For example, use 20:00 to signify 8 p.m.

You can restrict the data represented by dimensions or query subjects in a project by creating a security filter. The security filter controls the data that your users can see when they set up their reports.

You can also apply governors to restrict the data that the queries in a package retrieve.

Framework Manager supports stand-alone filters and embedded filters.

- Use a stand-alone filter when you want to reuse the expression.

You can add a stand-alone filter to one or more dimensions or query subjects to limit the data that the query retrieves when the filtered dimension or query subject is used in a report, or you can include it in a package to make it available to your users. By moving a stand-alone filter or a shortcut to it into a folder, you can better organize the model objects.

- Use an embedded filter when you want to use a filter with only one dimension or query subject. You can create an embedded filter when modifying a dimension, relational data source query subject, or model query subject.

If you start with an embedded filter, you can later convert it into a stand-alone expression that you can apply to other dimensions or query subjects. **Tip:** Right-click the filter expression in the **Filters** tab and click **Convert to Stand-alone Filter**.

When you embed a filter, the data source query subject must have a relationship to any query subject referenced by the expression. This relationship is necessary even if the expression references a model query subject based on the same table as the data source query subject in which you are embedding the expression.


To create a filter on an unrelated query subject, do one of the following:

- Ensure that there is a join path between the new query subject and the one that contains the filter.
- Base the embedded filter on a query item that is based on the data source query subject you want.
- Convert the calculation to a stand-alone filter, so that it is not part of the query subject.
- Create a stand-alone filter that references the embedded object.

Steps

1. Do one of the following:
 - If you want to create a stand-alone filter, click the namespace or folder and, from the **Actions** menu, click **Create, Filter**.
 - If you want to create an embedded filter, double-click the dimension or query subject that will contain the filter, click the **Filters** tab, and then click **Add**.
2. In the **Name** box, type a name for the filter.
3. Define the expression.

Tip: If there is an invalid expression in the filter, review the **Tips** box in the expression editor for more information.

Goal	Action
Add query items and filters	On the Model tab, drag the objects you want to the Expression Definition box.
Add functions	On the Functions tab, drag the functions to the Expression Definition box.
Add parameters	On the Parameters tab, drag the parameters to the Expression Definition box.
Retrieve all data and show a specified number of rows	<p>Click the options button, select the Restrict the maximum number of rows to be returned check box, and type the required number of rows to be returned.</p>  <p>This setting does not improve performance for retrieving data when testing dimensions, query subjects, and query sets.</p>
Override session parameters	Click the options button, click Set , enter a value in the Override Value field, and click OK .
Override prompt values	<p>Click the options button, and then click Prompts.</p> <p>The Model Prompts Manager dialog box appears, which shows all prompts, and their values, that are in the model.</p>

4. Click **OK**.

You may be interested in the following related topics:

- security filters ([p. 257](#))
- functions "Using the Expression Editor" ([p. 381](#))
- parameters ([p. 243](#))
- session parameters ([p. 245](#))

You can also apply governors to restrict the data that the queries in a package retrieve ([p. 304](#)).

Apply a Filter

Information about filters for relational metadata appears in a different topic (p. 160).

To apply a filter, you must modify the dimension, data source query subject, or model query subject. The query subject must either contain the query items that the filter references, or have a relationship path to the query subjects that contain the query items.

You can embed a stand-alone filter in dimensions or query subjects, but if you want a different usage for each embedded filter, you must create different versions of the stand-alone filter. Otherwise, your users could be required to fill in a prompt that you thought was optional if there is any instance where the usage is set to mandatory. For information about mandatory and optional prompts, see (p. 171).

For example, in query subject A, you embed a stand-alone filter and define it as optional. In query subject B, you define it as mandatory. When your users create a report that uses both query subjects, they are required to choose values in both filters, even the one defined as optional. All instances of the filter are considered to be mandatory when used in the same query. The solution is to create different versions of the filter, each with its own name.

Steps

- 1. Create a filter.
- 2. Select the filter and, from the **Actions** menu, click **Edit Definition**.
- 3. Click the **Filters** tab, and drag the filter you created to the **Filters** box.
- 4. Select a usage value for the filter.

Usage Value	Description
Always	Use this usage value to ensure specified data is filtered out of all reports. For example, your company may have obsolete information that it stores but does not want to report on. Always is the default usage value.
Design Mode Only	Retrieves a small subset of the data for the sample report. Use this usage value when you do not need to see all the data, for example when testing a query subject. To apply design mode filters in Framework Manager, select the Apply all relevant design mode filters when testing option. This option is available on the Test Settings tab. Your users may need the design mode filter in Query Studio when they want to focus on designing the layout and format of a report and not retrieve all the data as they work. To access the design mode filter in Query Studio, run the report with limited data.

Usage Value	Description
Optional	<p>Specifies that the filter is optional. The user is prompted to filter data and can leave the prompt blank. If the prompt is blank, Framework Manager ignores the filter and retrieves all data for the dimension or query subject.</p> <p>The ? ? syntax is required for optional prompts.</p> <p>Use this usage value if your users want to control when the filter is applied. For example, you want to see on one country sometimes and see the data for all countries at other times. An optional filter for country looks like this:</p> <pre>([GeoNamespace].[Countries].[CountryName] = ?WhichCountry?)</pre>

5. If you want to view the SQL, click the **Query Information** tab.
6. Click **OK**.

Create a Parameter Map

Information about parameter maps for relational metadata appears in a different topic ([p. 163](#)).

Use parameters to create conditional query subjects that allow for substitutions when the report is run. 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. You can manually enter the keys and values, import them from a file, or base them on existing query items in the model.

You can also export parameter maps to a file. To modify the parameter map, you can export the map values to a file, do additions or modifications and then import it back into IBM® Cognos® Framework Manager. This is especially useful for manipulating large, complex parameter maps.

All parameter map keys must be unique so that Framework Manager 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.

The value of a parameter can be another parameter. However, you must enclose the entire value in number signs (#). 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 percentage sign (%) instead of a dollar sign (\$).

We recommend that you assign an alias to a query item that uses a parameter map as part of its name and to add the multilingual names to the object in the **Language** tab (**Properties** pane).

Note: If you are using SAP BW metadata, you cannot use a query item to generate the keys and values of a parameter map.

Steps to Manually Create a Parameter Map

- 1. Click the **Parameter Maps** folder and, from the **Actions** menu, click **Create, Parameter Map**.
- 2. In the **Name** box, type a name for the new parameter map.
- 3. Click **Manually enter the parameter keys, and/or import them from a file** and click **Next**.
- 4. Do one of the following:
 - To manually enter values, click **New Key**, type a key, and press Tab to enter a value for that key.
 - To import keys and values, click **Import File** and identify the location of the appropriate .csv or .txt file. For a .txt file to be used for import, the values must be separated by tabs and the file must be saved as UTF8 or Unicode format. ANSI text files are not supported.

Note: If you are going to use a parameter in a data source query subject, the value must use English-specific punctuation. This means that you must use a period (.) to represent a decimal and a comma (,) to separate lists of values.

- 5. Modify existing parameters as required.

Goal	Action
Assign a default value	In the Default Value box, type a value. If the key used in an expression is not mapped, the default value is used. Setting a default value is optional. However, if no default is provided, an unmapped key could produce an error.
Remove a parameter	Select a row and click Delete .
Modify a parameter	Select the row you want to modify, click the Edit button, and type a value.
Clear all keys and values	Click Clear Map .

- 6. Click **Finish**.

Default Values and Parameter Maps

The **Default Low Value** and **Default High Value** properties may contain expressions that use parameter maps. You can use parameter maps to define a value for a target currency variable based on the user’s locale. For example, you define a parameter map that provides an ISO currency code. The value for the **Default Low Value** property could be defined as

```
#$Currency_Map[runLocale]#
```

This parameter map is used when the SAP BW variable Target Currency is used in a report.

These are the only properties related to SAP BW variables that can use parameter maps.

Create a Session Parameter

Information about session parameters for relational metadata appears in a different topic ([p. 165](#)).

A session parameter is a variable that IBM® Cognos® Framework Manager associates with a session. For example, user ID and preferred language are both session parameters. Because session parameters are key and value pairs, you can think of each session parameter as an entry in a parameter map named Session Parameters. 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 Content Manager. By default, the following session parameters appear 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. For more information, see the section "[Add a Language to a Project](#)" ([p. 134](#)).

- **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 IBM 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 see other session parameters, such as `account.personalInfo.email` or `account.personalInfo.surname`.

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, as well as static values.

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

The rules governing the use of parameters include the following:

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

Steps

1. From the **Project** menu, click **Session Parameters**.
2. Click **New Key** and type a session parameter key and value.
3. Choose how to handle the override value.
 - To avoid having to set the override value every time you edit the project, set the session parameter as a value.
 - To avoid having to remove the project setting each time before you publish it, set the session parameter as a session override.
4. Modify existing parameters as required.

Goal	Action
Change the parameter value	Click the row that contains the value you want to change, click Edit , and type a value.
Assign a default value	In the Default Value box, type a value. Framework Manager uses the default value if a key has an invalid value.
Remove a parameter	Click a row and click the Delete button. You cannot delete an environment session parameter.
Clear an override value	Click a row and click Clear Override .

5. Click **OK**.

Organizing the Model

Information about organizing a relational model appears in a different topic ([p. 179](#)).

When you organize the model, you make it easier for your users to find and understand the data in the model. You also make the model easier for you to manage and maintain. We recommend that you create several views, or layers, in the model:

- Keep the metadata from the data source in a separate namespace or folder.

In IBM® Cognos® Framework Manager, this is called the import view.

- Create one or more optional namespaces or folders for resolving complexities that affect querying using query subjects or dimensional objects.

To use IBM Cognos Analysis Studio or any OLAP-style queries, there must be a namespace or folder in the model that represents the metadata with dimensional objects.

- Create one or more namespaces or folders for the augmented business view of the metadata that contains shortcuts to dimensions or query subjects.

In Framework Manager, these are called the business view. Use business concepts to model the business view. One model can contain many business views, each suited to a different user group. You publish the business views.

Security can be defined in any of the views. It depends on your business requirements. For example, if you need to keep everyone from viewing an object, you add security to the object in the import view. Typically security is applied in the business view.

You can

- ☐ include metadata in several folders by using shortcuts ([p. 247](#))
- ☐ organize objects by creating namespaces or folders ([p. 248](#))

Use Shortcuts

Information about shortcuts for relational metadata appears in a different topic ([p. 184](#)).

A shortcut is a pointer to an object, such as a relationship, a dimension, a query subject, or a folder. We recommend that you use shortcuts in the business view when there is an overlap between user groups and you want to include the metadata in more than one folder. With shortcuts, you can have multiple references to an object.

For example, you create folders named Orders, Products, and Customers. If you want both Orders and Customers to contain the same dimension, you must create a shortcut to the dimension and add it to both folders.

Note: Two shortcuts to namespaces or folders must not have the same name in a model. For other types of shortcuts (e.g., a shortcut of a query subject), the name must be unique within the parent namespace.

When you create a shortcut, IBM® Cognos® Framework Manager does not set the **Screen Tip** and **Description** properties. Unless you define these properties, the values shown in the IBM Cognos studios are the same as those defined in the object that the shortcut references.

Tip: To go to the object that the shortcut references, right-click the shortcut and click **Go To Target**.

Shortcuts are less flexible from a presentation perspective than model objects, but they require much less maintenance because they are automatically updated when the target object is updated. If

maintenance is a key concern and there is no need to customize the appearance of the query subject, use shortcuts.

IBM® Cognos® Framework Manager has two types of shortcuts:

- regular shortcuts, which are a simple reference to the target object.
- alias shortcuts, which behave as if they were a copy of the original object with completely independent behavior. Alias shortcuts are available only for query subjects and dimensions.

Shortcuts and Dimensions

Shortcuts result in fewer dimensions to maintain. You can keep dimensions in the import view and keep shortcuts in the business view.

When you create a shortcut to a dimension, you cannot customize which query items are in the shortcut. The entire dimension is included in the shortcut.

The security you specify for an object is passed to shortcuts that reference the secured object. If you have a shortcut to a secured object, only users with permission to see the secured object can see the shortcut in the published package.

Step

- Right-click the query subjects, dimensions, or folders that you want to create shortcuts to, and click **Create, Shortcut**.

Create a Folder or Namespace

Information about folders and namespaces for relational metadata appears in a different topic ([p. 187](#)).

You can create folders or namespaces to organize objects in the model.

The most important thing to know about namespaces is that once you have begun authoring reports, any changes you make to the names of published namespaces will impact your IBM Cognos content. This is because changing the name of the namespace changes the IDs of the objects published in the metadata. Because the namespace is used as part of the object ID in IBM Cognos Framework Manager, each namespace must have a unique name in the model. Each object in a namespace must also have a unique name. Part of the strategy of star schema groups is placing shortcuts into a separate namespace, which automatically creates a unique ID for each object in the namespace. For relational databases, this allows us to use the same name for shortcuts to conformed dimensions in different star schema groups.

The next time you try to run a query, report, or analysis against the updated model, you get an error. If you need to rename the namespace that you have published, use **Analyze Publish Impact** to determine which reports are impacted.

Folders are much simpler than namespaces. They are purely for organizational purposes and do not impact object IDs or your content. You can create folders to organize objects by subject or functional area. This makes it easier for you to locate metadata, particularly in large projects.

The main drawback of folders is that they require unique names for all query subjects, dimensions, and shortcuts. Therefore, they are not ideal for containing shared objects.

Tip: When viewing metadata in the **Diagram** tab, you can expand or collapse folders and namespaces. From the **Diagram** menu, click **Collapse All** or **Expand All**.

If you set security on a folder and then move objects into the folder, confirm that exclusions are set correctly.

For SAP BW metadata, shortcuts to namespaces are not supported.

Steps to Create a Folder

1. From the **Actions** menu, click **Create, Folder**.
2. In the **Folder name** box, type a name for the new folder.
3. Click **Next**.
4. Choose whether to move the objects or to create shortcuts:
 - To move selected objects to the folder, click **Move the selected items to the new folder**. When you move an object that participates in a relationship, the relationships to this object also move.
 - To create shortcuts that reference selected objects, click **Create a shortcut for the selected items**. Do not select all the objects in the namespace to avoid creating a recursive structure in the published package.
5. Select the objects you want to add to the folder.
6. Click **Finish**.

Steps to Create a Namespace

1. From the **Actions** menu, click **Create, Namespace**.
2. Right-click the namespace, click **Rename**, and give the namespace a descriptive, unique name.
3. Add objects by importing metadata or moving model objects or shortcuts to the objects into the namespace.

Chapter 7: Publishing Packages

You publish a package to make the metadata available to your users. You create packages based on your user groups. Packages must contain all the information that a specific user or group of users needs to create reports.

We recommend that you do the following:

- ❑ verify the model and repair any problems ([p. 251](#))
- ❑ analyze the model for potential modeling issues that you should examine before publishing the metadata ([p. 191](#))
- ❑ set security to restrict access to metadata and data across IBM Cognos products ([p. 256](#))
- ❑ specify the languages published with each package ([p. 262](#))
- ❑ create or modify a package ([p. 254](#))
- ❑ specify the suppression options that will be available to package users. ([p. 263](#))
- ❑ publish the package ([p. 266](#))

In addition, you have the option to externalize query subjects and dimensions to convert them to formats that you can use in Transformer or other applications ([p. 263](#)).

If you have published packages, you verify them and repair any problems ([p. 251](#)).

Note: Externalizing queries will not be supported in future releases of IBM® Cognos® Transformer.

Verify a Model or Package

At any point in the modeling process, you can check the validity of the whole model, or selected objects in the model such as a package. We recommend that you first verify the model and repair any problems, and then verify each package within a model individually. By verifying a package, you can find and remove invalid objects that can cause a query to fail.

Verification Categories

When you verify a model or package, IBM® Cognos® Framework Manager looks for messages in the following categories:

Category	Description
Internal Model Inconsistencies	Verifies that objects are properly defined and that duplicate names are not used. We recommend that you always run this group of tests.

Category	Description
Invalid or Incomplete Object References	Checks for dangling references or references to missing objects.
Determinant Completeness	Verifies that determinants are completely defined with keys and attributes, and that all query items are accounted for in a determinant.
Dimension Completeness	Verifies that dimensions are fully defined with level member captions and business keys.
Query Status	<p>Verifies that the evaluation status for query subjects and dimensions is valid and is not set to "needs reevaluation" or "invalid".</p> <p>Query status does not verify shortcut objects. For example, you create a shortcut to an object. The object becomes invalid for some reason. If you verify the object, a message is shown because it is invalid. When you verify the shortcut, no message is shown.</p>
Backward Compatibility	Informs you that a model contains features from a previous release that have been maintained but will be deprecated in a future release.
Verify Relationship Cardinality	Warns when many-to-many relationships are found.

Each category can generate multiple messages for an object. For each message, the severity, object icon, object ID, description of the message, explanation of how to correct the problem, and possible actions that can be performed on the object are provided. You may be able to correct a problem immediately by modifying the object that caused the problem or by asking Framework Manager to repair the problem.

Steps

1. Choose one of the following:

Goal	Action
Verify a model	From the Project menu, click Verify Model .
Verify a package	In the Project Viewer , right-click a package, and click Verify Selected Objects .

Goal	Action
Verify selected objects	In the Project Viewer , select one or more objects, right-click, and click Verify Selected Objects .

2. On the **Options** tab, select the message severity levels that you want to see in the results.
By default, all message severity level check boxes are selected. Clear the ones that you do not want.
3. Select the categories that you want to verify.
By default, all category check boxes are selected. Clear the ones that you do not want.
4. Click **Verify Model**.
The **Verify Model Results** tab shows the results.
5. To sort the messages, click **Sort** in the severity, object type, or name column heading.
6. To see the object that is related to the message, under the **Actions** heading, click **Find in Project Viewer**.
7. To repair problems for a group, select a grouping criteria from the list.
8. Select the check box beside each message for the problem that you want to repair.

Tips:

- To select all messages, select the check box at the top of the check box column. To clear all selected messages, clear the check box at the top of the check box column.
 - If you grouped the messages, select the check box at the top of the group check box column to select all messages in the group.
 - To hide the message detail information, click **Collapse** in the group heading.
9. Click **Repair Selected**.

Framework Manager repairs the problems in the following order:

- invalid references
- invalid objects, invalid relationships, invalid aggregation rules, missing locales, unsupported prompt types, inferred roles, and re-evaluate; in no particular order
- upgraded model errors

You can also repair problems by clicking the **Edit Definition** icon under the **Actions** heading for the message. In the dialog box, modify the definition of the object as required.

When the repair process is finished, a summary of the repair results appears. The model or package is verified again and the results are shown in the **Verify Model Results** tab.

Tip: Problems that cannot be repaired during the verification of a package may be repaired using verify model.

For stand-alone filters, if the underlying objects might not be valid, a "needs reevaluation" message appears. The Repair option does not work for stand-alone filters. To ensure that the stand-alone filters are valid, open each filter and save it.

Create or Modify a Package

You create a package to make metadata available to your users. A package is a subset of a project. It must contain all the information that a specific user or group of users needs to create reports. You can also apply security to the package (p. 256).

For example, if your data source contains information from different areas of a business, you might decide to create different packages for Human Resources and Finance. Ensure that your package meets a broad but related reporting need. Each report can contain information from a single package only.

After a package is published to the server, it is available to your users.

Reusing Packages

You can reuse packages by creating nested packages. When you create nested packages, you create a master package that is based on other existing packages. Using nested packages saves you time, and they are easier to maintain. Another advantage of using nested packages is that you publish only the master package.

For example, you create three separate packages named Canada, Mexico, and the United States. Each package contains the project objects and security appropriate for that package. You can create one master North America package and include the packages Canada, Mexico, and the United States.

You can also reuse packages to create a consolidated package with connections to multiple data sources.

Selecting, Hiding, Unselecting

When you create a package, you can choose whether objects in a project can be selected based on the requirements of your users.

Option	Description
Select	The object can be used in reports and can be selected by your users.
Hide	<p>The data within the object cannot be used in reports because it cannot be selected by your users. Any existing reports referencing this object will not run.</p> <p>For example, you include a model query subject in a package. Because model query subjects are dependent on data source query subjects (p. 85), you must add the data source query subject to your package. If you do not want your users to see the data source query subject, hide it.</p>

Option	Description
Unselect	The object is not published. It cannot be used for reports and cannot be selected by your users.

Note: IBM® Cognos® Framework Manager supports Ctrl+shift and Alt+shift functionality. Use these keystrokes to select multiple objects that you wish to include or hide. For example, if you wish to only include two items in a large branch, select the entire branch, then use Ctrl+shift to deselect the items you wish to include, and hide the remaining selected items.

Including a Model Query Subject in a Package

If a model query subject references other query subjects in a macro or a prompt, ensure that you include the referenced query subjects in the package. This can occur in the following situations:

- A macro for the model query subject references query items in another query subject.
- Another query subject is referenced in the **Prompt Info** properties.

Steps to Create a Package

1. Click the **Packages** folder, and from the **Actions** menu, click **Create, Package**.
2. In the **Provide Name** page, type the name for the package and, if you want, a description and screen tip. Click **Next**.
3. Specify whether you are including objects from existing packages or from the project and then specify which objects you want to include.

If you created other packages, we suggest that you add package references by clicking **Using existing packages**.

4. Choose whether to use the default access permissions for the package:
 - To accept the default access permissions, click **Finish**.
 - To set the access permissions, click **Next**.
5. Specify who has access to the package, and click **Next**.
You can add users, groups, or roles ([p. 256](#)).
6. Move the language to be included in the package to the **Selected Languages** box, and click **Next**.
7. Move the sets of data source functions you want available in the package to the **Selected function sets** box.

If the function set for your data source vendor is not available, make sure that it was added to the project. For more information, see "[Select Function Sets](#)" ([p. 314](#)).

8. Click **Finish** and choose whether to publish the package ([p. 266](#)).

Steps to Modify a Package

1. Click the package you want to modify and, from the **Actions** menu, click **Edit Definition**.
2. Click the objects you want to add to or remove from the package.

Tip: To toggle through the options for an object, click the object icon, or select an option from the list. For more information, see ([p. 254](#)).

3. Click **OK**.
4. If you want to add or remove package references to the package you are modifying, click **Edit**.

Security

In IBM® Cognos® Framework Manager, security is a way of restricting access to metadata and data across IBM Cognos products.

There are different types of security in Framework Manager:

- data security ([p. 257](#))
You create a security filter and apply it to a specific query subject. The filter controls the data that is shown to your users when they set up their reports.
- object security ([p. 259](#))
You secure an object directly by allowing users access to the object, denying users access to the object, or keeping it hidden from all users.
- package security ([p. 261](#))
You apply security to a package and identify who has access to that package.

Each type of security uses [users, groups, and roles](#) to define access.

There are business reasons for restricting access to data. For example, you may have confidential data that only specific users are allowed to see. You may have a variety of data, and your users only need to retrieve data from specific tables or columns. Or, you may have a table that contains many records, and your users only need to retrieve a subset of records from that table.

If you are using SAP BW metadata, there can be underlying SAP BW security that affects your users' access to level members. You cannot override SAP BW security in Framework Manager. For more information, see "[Import from an SAP BW Data Source](#)" ([p. 197](#)).

Before you add security in Framework Manager, ensure that security was set up correctly in IBM Cognos BI. For more information, see the *Administration and Security Guide*.

Users, Groups, and Roles

Users and groups are created for authentication and authorization purposes. You can create your own users and groups in IBM Cognos BI or use users and groups created in other authentication providers.

For more information about security, users, groups, and roles, see the *Administration and Security Guide*.

Users

A user entry is created and maintained in an authentication provider to uniquely identify a human or a computer account. You cannot create users in IBM Cognos BI.

Information about users, such as first and last names, passwords, IDs, locales, and e-mail addresses, is stored in the providers.

Users can become members of groups defined in authentication providers and groups defined in IBM Cognos BI. A user can belong to one or more groups. If users are members of more than one group, their access permissions are merged.

Groups and Roles

Examples of groups are Employees, Developers, or Sales Personnel. Members of groups can be users and other groups. Group membership is part of the user's basic identity. When users log on, they cannot select a group they want to use for a session. They always log on with all the permissions associated with the groups to which they belong.

A role is a special group. It represents a collection of users that have similar responsibilities and similar privileges in the organization. Members of roles can be users, groups, and other roles. Role membership is not part of the user's basic identity.

You can use groups created by your organization in authentication providers, or create new groups in the Cognos namespace.

Create IBM Cognos groups when

- you cannot create groups in your authentication provider
- groups are required that span multiple namespaces
- portable groups are required that can be deployed
- you want to address specific needs of IBM Cognos administration
- you want to avoid cluttering your organization security systems with information used only in IBM Cognos BI

Add Data Security

You can restrict the data represented by query subjects in a project by creating a security filter. The security filter controls the data that is shown to your users when they set up their reports.

For example, your Sales team consists of a Sales Director, and four Sales Managers. You create a security filter that includes a group for the Sales Director and a group for Sales Managers, and apply the filter to the Salary query subject. When the package is available for your users, and a report is generated for the Sales Managers and the Sales Director, only the Sales Director can see the salary information for the Sales Managers. For more information about groups, see ["Users, Groups, and Roles"](#) (p. 256).

If a user has multiple roles, the security filters belonging to these roles are joined together with ORs. If a role is based on another role, the security filters are joined together with ANDs.

You can base the security filter on an existing security filter. If you choose this option, the security filter inherits the filter and all the filter properties.

When you create a security filter, you can also use existing project filters, or create new filters using the expression editor. For more information, see ["Create a Filter" \(p. 158\)](#).

Steps

1. Click the query subject you want, and from the **Actions** menu, click **Specify Data Security**.
2. To add new users, groups, or roles, do the following:
 - Click **Add Groups**.
 - In the **Select Users and Groups** window, add users, groups, or roles. For information about how to do this, see the *Administration and Security Guide*.
 - In the **Select Users and Groups** window, click **OK**.
3. If you want to base the group on an existing group, click a group in the **Based On** column.
Tip: If you do not see the group you want in the list, you must add the group to the security filter.
4. If you want to add a filter to a group, in the **Filter** column, click either **Create/Edit Embedded Filter** or **Insert from Model**.

Using the CSVIdentityName Macro Function

If you want row-level security based on UserClass values stored in your data source, implement a parameter map that maps the values in the data source to the corresponding roles and groups based on the user you are logged on as.

You do this by using a parameter map as an argument with the `CSVIdentityName` macro function. This macro function retrieves account, group, and role information for the current user. It returns a string of comma-separated values from the parameter map in single quotation marks, such as 'clerks', 'technicians', or 'typists'.

The `CSVIdentityName` macro function is used as a key in the specified map. You can use the list that is returned to build partial `IN` clauses or to filter data based on the identity name of the current user.

For example, you have user classes whose names do not correspond to the `Roles_Groups` parameter map:

Key (Role or Group)	Value (User Classes)
Everyone	Group1
Authors	Group2

Key (Role or Group)	Value (User Classes)
System Administrators	Group3
Query Users	Group2
NTLM	Group2

You have this query subject:

```
(Security_column, value 1, value 2, value 3)
```

When you add a filter to the query subject, the filter uses a macro to look up a list of values, for example:

```
Security_column in (#CSVIdentityName(%Roles_Groups)#)
```

For users in the Everyone, Authors, and System Administrators roles, testing shows this as:

```
Security_column in ('Group1','Group2','Group3')
```

Using the CSVIdentityNameList Macro Function

If security data in the data source is identical to the roles and groups defined in IBM Cognos BI, you can use the `CSVIdentityNameList` macro function. The macro function optionally accepts a list separator as a parameter and then returns a separator-delimited list that can be used in a filter with the `In` operator. You do not need a parameter map.

Here is an example:

```
Security_column in (#CSVIdentityNameList()#)
```

For users in the Everyone, Authors, and System Administrators roles, testing shows this as:

```
Security_column in ('Everyone','Authors','System  
Administrators')
```

Consider the following:

- users can belong to several groups or roles
- there is no way to distinguish between groups and roles so each group and role must have a unique name.
- this function works only in a filter and always returns 0..n values

Add or Remove Object Security

Metadata security can be applied directly to objects in a project. When you add object-based security, you apply a specific user, group, or role ([p. 256](#)) directly to the object. You choose to make the object visible to selected users or groups. You cannot deny the **Everyone** group access to all objects.

If you do not set object-based security, all objects in your project are visible to everyone who has access to the package. Users, groups, or roles that do not have allow or deny settings for an object are considered to be undefined. The object then inherits the security that was defined for its parent object. When you explicitly allow or deny access to an object, you override the inherited settings. The parent and child objects then have different settings. When you apply security to a parent

object, all its child objects in the model will also have security applied to them. After you set security for one object, you must set it for all objects. You can do this by setting security on the root namespace.

You may want an object to be visible to only selected users, groups, or roles. For example, in your project, you may have a Salary query subject. You can make the Salary query subject visible to the Managers group, and keep it hidden from everyone else.

If a user is a member of multiple user groups and an object is visible to one user group and denied to the other, the user will not have access to the object. For example, Jane belongs to two user groups: Sales and Marketing. The Sales group has access to the Products and Sales query subjects, and is denied access to the Sales Forecast query subject. The Marketing group has access to Products, Sales, and Sales Forecast query subjects. Jane does not have access to Sales Forecast.

When you secure an object, a package is automatically created in IBM® Cognos® Framework Manager. The package name consists of an underscore (_) and the name of the secured object. These object-based packages are visible in the **Explorer**. You can use this package to see which objects in the project are included, hidden, or excluded from a specific user group.

Every time you include that object in a package, and publish it for your users, the same security rules apply for that object. When you publish a package that contains secured objects, the visible objects for users are the intersection of the package definition and the object security settings. If object-based security is not used, security applied to a package remains unchanged.

Scope of Object Security

The security you specify for an object is passed to shortcuts that reference the secured object. If you have a shortcut to a secured object, only users with permission to see the secured object are able to see the shortcut in the published package.

If a model query subject, calculation, or filter references a secured object, the object's security is not passed to the model query subject, calculation, or filter.

When you create a package containing the shortcut, the secured object does not need to be included in the package.

For example, only sales managers are allowed to see the Sales Target query subject. You create a shortcut to Sales Target. When you package the model, you include the shortcut but not the Sales Target query subject. Sales managers are the only ones able to see the shortcut in the published package.

If your model is segmented, object security is not inherited from the master model. You must define object security on all model segments.

Tips:

- To see a list of the object-based packages, double-click the **Packages** folder. The list appears in the **Explorer** tab. To see which objects are secured against that specific object-based package, click the **Packages** folder, and from the **Actions** menu, click **Packages, Explore Packages**(p. 271).
- To determine if object-based security is set in the model, click the **Packages** folder, and from the **Actions** menu, click **Packages, Explore Packages**(p. 271). Click the **Roles Explorer** tab. If object-based security was set, you see a package for the Everyone role.

- To determine which objects are explicitly secured in the model, look at the object's icon in the **Project Viewer**. The top left corner of the icon is marked with an overlay.
- To find all objects that were explicitly secured under a given object, select the object and, from the **Tools** menu, click **Find All Secured Objects**.
- To remove object-based security for a particular user, group, or role, delete the package for that user, group, or role from the **Project Viewer**.
- To completely remove object-based security from the model, delete the package for the Everyone role from the **Project Viewer**.

Steps to Add Object-Based Security

1. Click the object you want to secure, and from the **Actions** menu, click **Specify Object Security**.
Tip: You can select more than one object at a time.

2. Select the users, groups, or roles you want to change. Or, click **Add** to add new users, groups, or roles.

For more information, see the *Administration and Security Guide*.

3. Specify security rights for each user, group, or role by doing one of the following:
 - To deny access to a user, group, or role, select the **Deny** check box next to the name for the user, group, or role. Deny takes precedence over Allow.
 - To grant access to a user, group, or role, select the **Allow** check box.

Tip: To allow everyone to see all objects unless specifically denied access, select the **Allow** check box for the Everyone role.

4. Click **OK**.

A list of new and updated object-based packages appears.

Steps to Remove Object-Based Security for an Individual Object

1. Click the secured object, and from the **Actions** menu, click **Specify Object Security**.
2. Remove security rights by clearing both the **Allow** and **Deny** check boxes for all users, groups, or roles.
3. Click **OK**.

A list of packages that are affected by these changes appears.

Modify Package Security

You can use IBM® Cognos® Connection to define or change metadata security after a package has been published.

Security settings modified through IBM Cognos Connection are added to the portal's security definition of the package. The settings affect access but do not change the package definition in IBM Cognos Framework Manager.

To define metadata security the first time you publish a package, see "[Publish a Package](#)" (p. 266).

You can organize your security by specifying which users, groups, and roles have access to certain parts of the published model.

To add metadata security, do the following:

- ☐ Decide whether the objects can be selected, unselected, or hidden in the package ([p. 254](#)).
- ☐ Decide which users will have administrative access to a package.
- ☐ Add users, groups, and roles to the package.

When you apply administrative access to a package, you give access to the user or users who are responsible for

- republishing a package in Framework Manager to the IBM Cognos server
- ensuring that no reports are impacted when a Framework Manager package is republished to the server

Steps

1. Click the package you want to edit, and from the **Actions** menu, click **Package, Edit Package Settings**. This invokes IBM Cognos Connection.
2. In IBM Cognos Connection, click the **Permissions** tab.
3. Create, add, or remove groups or roles as required. For information about how to do this, see the IBM Cognos Connection *User Guide*.
4. After you finish modifying the security definition for the package, click **OK** to return to Framework Manager.

Specify Languages

You can specify which languages are published with each package. You can create several packages based on the same model, each using a different language.

For example, the package for the Mexican sales office includes Spanish and English. The package for the Canadian sales office includes French and English.

You can also specify the languages for all packages at one time.

You must add languages to the project ([p. 131](#)) before you can specify the languages that your users require in packages.

Steps for One Package

1. In the **Project Viewer**, click the package you want to modify.
2. In the **Properties** tab, find the **Language** property and click **Click to edit**.

3. Click a language (or Ctrl+click multiple languages) in the **Available Project Languages** box and use the arrow button to move it to the **Selected Languages** box.

Steps for All Packages

1. In the **Project Viewer**, click the **Packages** folder.
2. From the Actions menu, click **Packages** and click **Specify Package Languages**.
3. Select the check box for the language you want for each package.

Set Suppression Options

You can set suppression properties for the package published with IBM® Cognos® Framework Manager. These properties determine whether IBM Cognos studio users can choose multi-edge or single edge suppression. The properties also determine the types of values that can be suppressed. Types of values that users can choose to suppress depend on the studio.

When a package is created, the suppression properties are automatically set to `true`. If you want to change the values on a published package, you must re-publish the package.

Steps to Set Suppression

1. In the **Project Viewer**, click the package you want to modify.
2. In the **Properties** tab, find the suppression property and click to select the required value.
 - **Allow Null Suppression** - When `true`, this property makes suppression available to IBM Cognos studio users. When `false`, suppression is not available in the published package.
 - **Allow Multi-Edge Suppression** - When `true`, users can select multi-edge or single edge suppression options. When `false`, users will only have access to single edge suppression. The **Allow Null Suppression** property must also be `true`.
 - **Allow Access to Suppression Options** - When `true`, users can choose the types of values that will be suppressed, such as zero or missing values. By default, all the types of values are suppressed. The **Allow Null Suppression** property must also be `true`.

Externalizing Query Subjects and Dimensions

When publishing a package, you have the option to externalize query subjects and dimensions into formats that you can use in IBM® Cognos® Transformer or other applications. Special considerations must be given when externalizing models based on SAP BW metadata. For more information, see [\(p. 602\) "Working with SAP BW Data Using Externalized CSV Files in Framework Manager"](#).

You first define how each object will be externalized by specifying a method to use. When you publish the package, you specify that the query subjects and dimensions are to be externalized.

If you specified a maximum number of rows to be retrieved in the **Governors** dialog box, this setting is ignored.

You have several options for the externalization method.

The Default Method

Use the Default method to specify the objects in a package that you do not want to be externalized. To improve performance, you may not want to externalize all objects in a package.

The CSV Method

Use the CSV method to generate a comma separated file that contains the results of the query subject. In a CSV file, the first row represents the column names and each of the following rows contains one record from the query result set. One file is generated for each query subject or dimension that is set to be externalized.

With the CSV method, you can use locally processed functions to create a dataset for use in Transformer. You can process Cognos SQL locally or on the data source, and capture the result set in a file that can be used in IBM Cognos Transformer.

The generated file is restricted to 2 GB in size and contains data based on the native encoding of the current operating system. For example, for Windows 2000, this is specified by the default system locale in the Windows regional settings. For Windows XP and 2003, this is specified by the **language for non-Unicode programs** option in the Windows regional settings.

This option is intended for use only with Transformer. For any other purpose, we recommend that you use the tab method.

To externalize SAP BW query subjects, use the CSV method. For more information, see [\(p. 602\)](#).

The Tab Method

Use the tab method to generate a tab delimited file that contains the results of the query subject or dimension. The generated file can be used directly as a data source. The generated file contains data based on Unicode using UTF-16 LE (Little Endian) encoding with BOM (Byte Order Mark). One file is generated for each query subject or dimension that is set to be externalized.

This method does not work with Transformer because Transformer does not support Unicode. Use the CSV method to create files for Transformer.

The IQD Method

Use the IQD method to generate a query definition file to be used in IBM® Cognos® Transformer. One file with Native SQL is generated for each query subject or dimension that is set to be externalized. The generated file contains data based on the native encoding of the current operating system. For example, for Windows 2000, this is specified by the default system locale in the Windows regional settings. For Windows XP and 2003, this is specified by the **language for non-Unicode programs** option in the Windows regional settings.

The query subject must not require any local processing. It must be able to be run entirely on the data server. We recommend that you test the query subject by setting the query processing for this data source to database only. An error message then appears if the query subject requires local processing.

If you must use locally processed functions to create a dataset, we recommend that you use the CSV method. With the CSV method, you can process Cognos SQL locally or on the data source and capture the result set in a file that can be used in Transformer.

Stored procedure query subjects can be externalized for use in Transformer. The stored procedures must not contain any parameters.

Note that the IQD method will continue to be supported in this release but will not be enhanced. For more information, see ["Deprecated Features in Version 8.3" \(p. 18\)](#)

The Externalize Auto Summary Property

You can specify that the output be aggregated or grouped or both. By default, IBM® Cognos® Framework Manager returns rows at the detail level without applying any aggregation or grouping. This property is used when you want to have relational data aggregated when it is externalized. Specify determinants for the query subject before externalizing it.

Use the **Externalize Auto Summary** property to apply the setting of the **Regular Aggregate** property to query items whose **Usage** property is set to **fact**.

If you want to have a specific order of items in the **Group By** clause, specify determinants first, and then set the **Externalize Auto Summary** property.

You can use the **Externalize Auto Summary** property with all externalize methods.

Supported Data Types

Framework Manager supports strings, integers, and dates. It does not support time dimensions. We recommend that you use a date key on the fact query subject in Framework Manager, and let Transformer generate the time dimension.

Shortcuts

If a shortcut is included in a package and it points to a query subject that has been externalized, the shortcut will also be externalized. The name of the data file is the name of the query subject to which the shortcut points. If more than one shortcut points to the same query subject, then the query subject is externalized each time the shortcut is encountered.

Query Processing

Native SQL is used to generate an IQD, so the native SQL produced when externalizing must run successfully.

Some queries cause more than one query to be issued, or local processing to be performed to retrieve data, or both. To prevent this, ensure that the **Query Processing** property for all data source objects in the model is set to **Database Only**. For more information about query processing, see ["Improve Performance by Setting Query Processing Type" \(p. 312\)](#).

Process to Externalize Dimensions

We recommend that you follow this process to externalize dimensions:

- ❑ Create a model query subject or a data source query subject that contains the dimensions you want to externalize [\(p. 85\)](#).

- ❑ Add any filters that you require.

For information about filters for relational metadata, see ["Create a Filter" \(p. 158\)](#). For information about filters for SAP BW metadata, see ["Create a Filter" \(p. 239\)](#).

- ❑ In the **Properties** pane, set the **Externalize Method** property to the method you want.
- ❑ Publish the package to externalize the dimensions you selected [\(p. 251\)](#).

Publish a Package

In IBM® Cognos® Framework Manager, you can publish a package [\(p. 254\)](#) to any folder in Content Manager so your users can access it.

You can also publish a package to a network location. A package on a network location cannot be used by your users. Publishing to a network location is useful for backing up a package.

Note: When you publish to a LAN location, be careful that you do not over-write any existing files, particularly Framework Manager models and the model that is currently open.

Objects that are excluded or hidden, but are necessary in a query path, are included but marked as hidden in a published package.

The governor settings that take precedence are the ones that apply to the model that is currently open (whether it is a parent model or a child model).

To avoid problems, troubleshoot the package before publishing it by using the **Verify the Package Before Publishing** check box in the Publish wizard to ensure that it is complete and does not contain errors.

When you publish a package, you can

- Set the number of model versions to retain on the server. For more information about model versions, see ["Update a Report to Use the Latest Version of a Package" \(p. 270\)](#).
Tip: To see the number of model versions set for a package, select a package and, in the **Property** pane, find the **Max Versions** property.
- Externalize query subjects and dimensions so that you can use them with Transformer [\(p. 263\)](#).
- Specify whether a package will use dynamic query mode.

For more information about dynamic query mode, see the *Dynamic Query Guide*.

Note: You can create packages directly in IBM® Cognos® Connection for IBM Cognos PowerCubes and SAP BW cubes and queries. For more information, see the section about packages in the *Administration and Security Guide*.

Steps

1. Select the package you want to publish.
2. From the **Actions** menu, click **Package, Publish Packages**.
3. Choose where to publish the package:

- To publish the package to the report server, click **IBM Cognos Content Store**, click **open**, and select an existing folder or create a new folder in the Content Store.
 - To publish the package to a network location, click **Location on the network**. Ensure that you select a different location than the directory where the project files are stored. In general, avoid saving to the same location as a model as the model could be overwritten.
4. To enable model versioning when publishing to the IBM Cognos Content Store, select the **Enable model versioning** check box and type the number of model versions of the package to retain.
Tip: To delete all but the most recently published version on the server, select the **Delete all previous model versions** check box.
 5. Click **Next**.
 6. In the **Add Security** window, define security for the package (optional):

Goal	Actions
Create, add, or remove a user, group, or role.	<p>On the User Access tab, click Add.</p> <p>In the Select Users and Groups window, define user security. For information about how to use the Select Users and Groups window, see the <i>Administration and Security Guide</i>. Users, groups, or roles defined in the user Access Tab have Read, Write, Execute, and Traverse permissions.</p>
Grant administrative access to a user, group, or role.	<p>On the Administrator Access tab, click Add.</p> <p>In the Select Users and Groups window, define administrator security. For information about how to use the Select Users and Groups window, see the <i>Administration and Security Guide</i>. Users, groups, or roles defined in the Administrator Access Tab have Read, Write, Set Policy, and Traverse permissions.</p>

Note: The **Add Security** window in the Publish Wizard is only available the first time you publish a package. If you re-publish a package to the same location, you cannot override the existing security. To change security for a published package, see "[Modify Package Security](#)" (p. 261).

7. Click **Next**.
8. If you want to externalize query subjects, select the **Generate the files for externalized query subjects** check box.
9. By default, the package is verified before it is published. If you do not want to verify your model prior to publishing, clear the **Verify the package before publishing** check box.

10. If your package contains supported data sources, you will have the option to enable dynamic query mode. Select the **Use Dynamic Query Mode** check box so all reports using the package will use the dynamic query mode.

Note: If you selected dynamic query mode and your package contains both supported and unsupported data sources, you will get an error if you click **Publish**. For information on data sources supported by dynamic query mode, see the IBM Cognos *Administration and Security Guide*.

11. Click **Publish**.

If you chose to externalize query subjects, Framework Manager lists which files were created.

12. Click **Finish**.

Publish a Package Based on an OLAP Data Source

You can use IBM® Cognos® Framework Manager to connect to an OLAP data source and create a package based on a cube. You can then publish the package directly to IBM Cognos Connection, making it available for use in the IBM Cognos studios.

By default, each package contains a connection to only one cube. If you want to create a package containing multiple cubes, run the metadata wizard and create a package for each cube. Then create a package that includes individual packages as required.

Before creating a package containing multiple cubes, consider the potential performance impacts. In IBM Cognos Connection, each time a package is used, a connection is made to each of the data sources defined in the package. Creating large packages with multiple cubes can have a negative impact on performance. To offset the potential performance impact of creating one large package containing many cubes, create one package per cube and then create smaller combinations of packages as required.

By default, packages based on an OLAP data source do not contain vendor function lists. If you want to include the vendor function list, specify the function sets to include and then republish the package.

Note: To publish a package that contains a single cube, we recommend that you use IBM Cognos Connection. For information on publishing a package from IBM Cognos Connection, see the *Administration and Security Guide*.

Steps to Create and Publish a Package

1. In the **Welcome** page, click **Create a new project**.

Tip: If you are already in Framework Manager, click **New Project** from the **File** menu.

2. In the **New Project** page, specify a name and location for the project and click **OK**.

You may be prompted to provide authentication information.

3. In the **Select Language** page, click the design language for the project.

The language you select cannot be changed after you click **OK**, but you can add others. For more information, see [\(p. 134\) "Add a Language to a Project" \(p. 134\)](#).

4. In the **Metadata Wizard** dialog box, click **Data source** and click **Next**.
5. Select your data source from the list of available data source connections and click **Next**.
If the data source connection is not available in the list, you can click **New** to create the data source connection. For more information, see "[Create a Data Source Connection](#)" (p. 56).
6. Specify a name for the package and click **Next**.
Optionally, you can specify a description and screen tip for the package.
7. Specify who has access to the package.
You can add users, groups, or roles ([p. 256](#)).
8. Click **Finish** to import the metadata and create the package.
9. When prompted, click **Yes** to publish the package or click **No** to return to the **Project Viewer**.

Steps to Add Another Package

1. In the **Project Viewer**, right-click the model that you want to use and click **Run Metadata Wizard**.
2. In the **Metadata Wizard** dialog box, click **Data Sources** and click **Next**.
3. Select your data source from the list of available data source connections and click **Next**.
If the data source connection is not available in the list, you can click **New** to create the data source connection. For more information, see "[Create a Data Source Connection](#)" (p. 56).
4. Specify a name for the package and click **Next**.
Optionally, you can specify a description and screen tip for the package.
5. Specify who has access to the package.
You can add users, groups, or roles ([p. 256](#)).
6. Click **Finish** to import the metadata and create the package.
7. When prompted, click **Yes** to publish the package or click **No** to return to the **Project Viewer**.

The namespace appears in the **Project Viewer**. You cannot see objects in the native metadata model from within Framework Manager. The native metadata objects are visible from within the IBM Cognos studios when the native metadata package is used.

Publishing a Package by Running a Script

IBM Cognos reports are dependent on the objects in the package on which the report is based. If your report uses objects that no longer exist in the package, the reports will not run. You can have this problem if you make changes to the physical data source, and then use a script to generate the model and republish the package. After the script is run and the package is published, analyzing the impact of publishing the package does not identify the broken reports.

To avoid this problem, do the following:

- ☐ Run the script, excluding any steps that publish the package.
- ☐ Verify the model or analyze the impact of publishing the package.
- ☐ Publish the package.

Update a Report to Use the Latest Version of a Package

When you publish a package for the first time, you create a corresponding package on the IBM Cognos server. The package contains a model but no reports.

When you publish a package, you can select the number of versions of the model to keep on the server. The next time you publish the package, the version of the model is updated in the existing package on the server.

New or modified reports use the latest version of the model in the package. When a report is saved, the version of the model used is saved in the report specification. If the package is republished, the report author is notified that the report uses the newest version of the model in the package. The report author must save the report to complete the update. If you open a saved report after the package it is based on is republished, one of two things happens:

- If the original version of the package still exists, the report runs against the original version. If you want the report to run against the latest version of the package, you must update the report to use the latest version of the package. See the steps below.
- If the original version of the package no longer exists, the report is updated to run against the most recent version.

Step

- Do one of the following:
 - Move one report to the latest version of the model by editing and saving the report.
 - Before republishing the model, move all reports to the latest version of the model by selecting the **Delete all previous model versions** check box in the **Publish** wizard.
 - Before republishing the model, disable model versioning by setting the model version limit to 1.

Chapter 8: Managing the Project

During the lifetime of a project, data may change and new requirements may appear that require you to update models and data sources.

After publishing your project, you can do the following to manage the content:

- ❑ Understand what metadata is in your model ([p. 271](#)).
- ❑ Implement multiuser modeling ([p. 273](#)).
- ❑ Administer the metadata ([p. 284](#)).
- ❑ Synchronize projects ([p. 301](#)).
- ❑ Control and optimize query behavior ([p. 304](#)).

Understanding the Metadata in Your Model

Before making changes to a published model, you can better understand the metadata by doing the following:

- ❑ Exploring your projects to see packages and roles in a project ([p. 271](#)).
- ❑ Viewing the objects in your packages to see where specific objects exist ([p. 272](#)).
- ❑ Creating documentation about your model to satisfy the specific requirements of your company ([p. 272](#)).

Explore a Package

When you have a large number of projects and object-based security in a project, it can be difficult to keep everything organized. You can explore packages to see the packages and roles in a project.

On the **Package Contents** tab, you see a list of all the packages (normal and object-based) in a project, as well as the objects that were selected, unselected, or hidden for each package.

On the **Object Security** tab, you see a list of all the users, groups, and roles in a project, and in which package the object-based security is applied. You can also see whether the objects in the project are hidden or visible to that specific user, group, or role.

Steps

1. Select the **Packages** folder.
2. From the **Actions** menu, click **Package** and click **Explore Packages**.
3. Choose what you want to do.

Goal	Action
View the contents of a package	Click the Package Contents tab.
Edit the package	Click the Package Contents tab, select the package and click Edit . For more information, see " Create or Modify a Package " (p. 254).
View the security for each package	Click the Object Security tab and select a package.

4. Click **Close**.

View the Distribution of an Object in Packages

When you view the package inclusion of an object, you see, by package, where that object exists and whether it is selected, unselected, or hidden in that package.

If the object is secured, you will also see the object-based package in which the object exists.

Steps

1. Click the object you want to see, and from the **Actions** menu, select **Edit Package Inclusion**.
2. To edit the package, click **Edit Package**.
For more information, see "[Create or Modify a Package](#)" (p. 254).
3. Click **OK**.

Create Model Documentation

After you model the metadata, you can create an HTML or XML representation of the model that can be customized and printed. This is useful for debugging your model or if your company requires this type of documentation to satisfy process requirements.

When you create model documentation, you can document the entire model or you can select a subset of the model. To document the entire model, you click the top-level namespace. The model documentation shows the selected object and all the properties and children of that object.

You can view, save, or print the report in XML or HTML format. By default, the XML format within IBM® Cognos® Framework Manager is always raw XML. To customize the XML report output in your browser, you can provide your own XSLT transformation.

To customize the HTML output, you can use your own XSLT by specifying the path and XSL file in the ModelDocXSL section of the fm.ini file, located in the *install_location*/configuration directory.

Framework Manager will use the specified XSL file when showing the HTML version of the Model Report.

Steps

1. Click the object that you want to document.

Tip: Click the top-level namespace to document the entire model.

2. From the **Tools** menu, click **Model Report**.

The model report appears.

You can save, print, or change the format of the report.

Multiuser Modeling

You can implement multiuser modeling in IBM® Cognos® Framework Manager by:

- Branching and merging ([p. 273](#))

If you use branching and merging to manage a multiuser project, each user can modify the same objects in a project at the same time. Each user has a copy of the entire project and can work on any part of it. When the branches are merged back into the root project, all conflicts between the root project and the branches are resolved.

- Segmenting and linking ([p. 280](#))

If you use segmenting and linking to manage a multiuser project, each user can look at the same parts of a project at the same time. However, you must ensure that each user modifies discrete parts of the project. Use links to allow different users to refer to a project at the same time as another user is working on it.

Branching and Merging Projects

Branching and merging enables multiple users to work on the same model at the same time. To do this, the project owner creates a branch of the root project, which is a copy of the project. A team member can modify the branch as required, independently of the root project. Branches can be merged back into the root project as required. Conflicts between the root project and a branch are resolved during the merge process.

There is no limit to the number of branches you can create. You can create a branch from a branch.

Recommendations for Branching and Merging

Use the following guidelines when branching a project:

- Decide how you want to share the metadata in your organization. For more information, see "[Methodologies for Branching](#)" ([p. 274](#)).
- Divide the project into logical pieces and branch the project to create the logical pieces. For more information, see "[Ways to Branch a Project](#)" ([p. 276](#)).

- Communication between team members that are working on various branches is very important. Before making a major change to your branch, talk to the other team members to see how the change will impact their branches. For example, adding objects to a branch will probably not impact other team members but deleting an object that other team members are using in their branches will create a conflict when you merge your branch back into the root project.
- Merge the branches in the reverse order that you created them in. For example, Sean creates a branch from the root project. Susan creates a branch from Sean's branch. When it is time to merge the branches back into the root project, Susan first merges her branch back into Sean's branch and then Sean merges his branch back into the root project.
- Merge branches back into the root project often, after making a few changes to your branch, rather than making many changes and merging only occasionally. This makes it easier to resolve conflicts during the merge process. You can merge all branches back to the root project and then branch the project again. Each team member receives an updated version of the model.
- In a branched project, any archived transactions (p. 296) will not be available when you merge back into the main project.

Methodologies for Branching

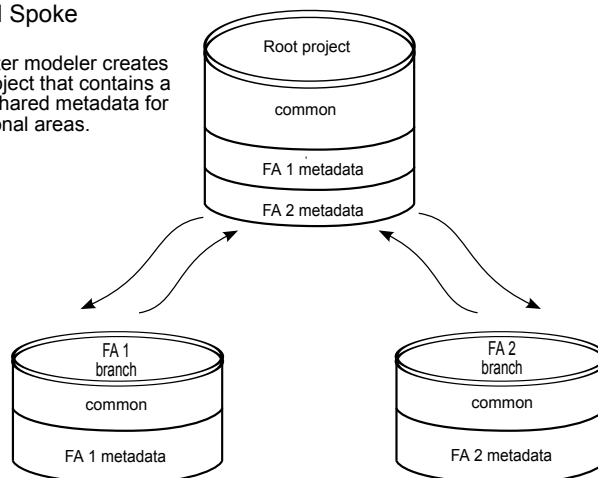
Before you branch a project, you must decide how you want to share the metadata in your project. This section describes some common methodologies for sharing metadata.

Hub and Spoke

In hub and spoke, a project uses common metadata that must be shared by all functional areas. The root project consists of a fully modeled physical layer containing the objects that all functional areas require. The root project is branched for each functional area. Each functional area can create its own branches if there are multiple people working on it. At any time, a functional area modeler can merge a branch back into the root project to update the root project, and then branch again to receive updates. Objects that are common to all functional areas are kept in the root project.

Hub and Spoke

The master modeler creates a root project that contains a layer of shared metadata for all functional areas.



Modeler enhances this branch by importing data sources, adding calculations, and creating and publishing packages specific to functional area 1.

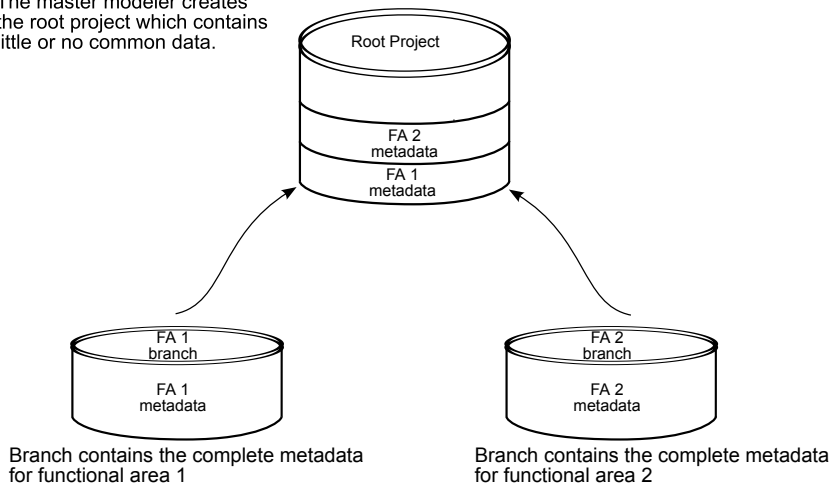
Modeler enhances this branch by importing data sources, adding calculations, and creating and publishing packages specific to functional area 2.

Functional Area Specific Metadata

In functional area specific metadata, there is little or no common metadata in the project. Each functional area develops their own objects in the project independently. Each functional area is unaware of the objects in the other functional areas. The master modeler controls merging of the branches to prevent each functional area from seeing objects in the other functional areas.

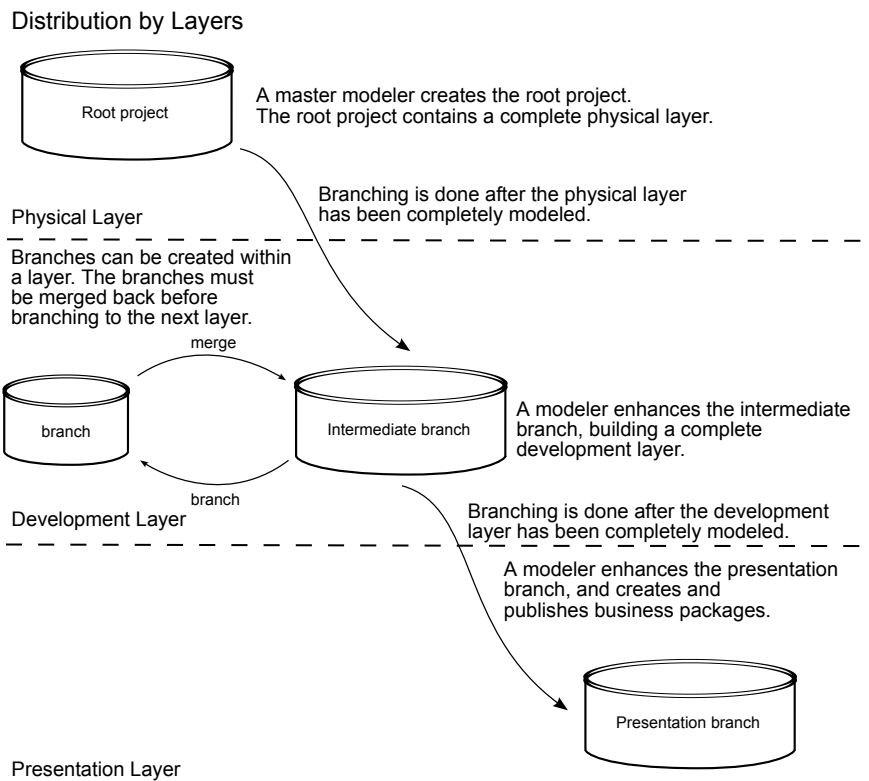
Functional Area Specific Metadata

The master modeler creates the root project which contains little or no common data.



Distribution by Layers

In distribution by layers, the metadata is organized in layers. Each layer requires access to the metadata of the layer above it. For example, a model contains three layers. The top layer is the root project, consisting of a fully modeled physical layer. The root project is branched to create the second layer, the intermediate branch. The intermediate branch contains a fully modeled development layer. The intermediate branch is branched to create the third layer, the presentation branch. The presentation branch contains a fully modeled business layer where reporting packages are defined and published.



Ways to Branch a Project

Here are some ways to divide a project:

- ❑ By task and skill set

Create separate branches for one person to work on star schemas and packages, one person to work on calculations and filters, one person to work on dimensions, and one person to work on queries and object naming.

- ❑ By model layers

Create separate branches for the import layer, the intermediate layer, and the presentation layer.

- ❑ By business unit or functional area

Create separate branches for sales and retailers.

- ❑ By type of data source

Create separate branches for relational, dimensional, and ERP data sources.

Create a Branch

You can create a branch in a project.

Steps

1. Open the project from which you want to branch.
2. From the **Project** menu, click **Branch to**.

3. In **Project name** box, type a name for the project.
4. In the **Location** box, type the path to the folder where you want to store the branched version.

The log file associated with the new branch will be empty.

When you create a branch for a read-only project, the resulting branch is writeable.

Merge Projects

You can merge a branch back into its root project.

To merge a branch back into the root project, IBM® Cognos® Framework Manager shows a list of transactions that have been performed on the branch. Beside each transaction is a check box. For each transaction that has its check box selected, Framework Manager attempts to perform the transaction on the project with which you are merging. If a transaction is completed successfully, it is marked with a check mark and the next transaction is attempted. If a transaction fails, it is marked with an "X" and processing stops.

When a transaction fails, you may be able to correct the problem immediately. For example, if an object is missing, Framework Manager identifies the missing object and prompts you to replace it with another object.

When you reach the end of the transaction list, you can accept or cancel the changes. If you accept the changes, the transactions that were successful are removed from the transaction list. If you do not accept the changes, the transaction list is not updated and you can perform the same merge again at a later time.

Some conflicts cannot be resolved during the merge process or there may be more than one transaction related to a failed transaction. In these situations, you can skip the transactions by selecting the **Uncheck dependent transactions** check box. Framework Manager then clears the check box for each transaction related to the failed transaction. When Framework Manager resumes running the transaction list, it does not run the transactions if their check boxes are cleared. When you reach the end of the transaction list, you can accept the changes to that point, troubleshoot the transactions that were skipped, fix the error, and then perform the merge again. When you perform the merge again, the transaction list contains only the transactions that were not performed.

Note: Before the transaction list is run, Framework Manager makes a backup of the merging project. The backup is stored in the same directory as the merging project.

Steps

1. Open the project into which you want to merge a branch.
2. From the **Project** menu, click **Merge from**.
3. In the **Select Project to Merge From** dialog box, click the **Files of Type** list and click **All Files (*.*)**.
4. Locate the log.xml file for the branch to be merged, and click **Open**.

The **Perform the Merge** window opens, showing a list of transactions. The transactions that you selected are run.

Framework Manager requires only the log.xml and the IdLog.xml files, not the entire set of project files to populate the transaction history list. If you do open the .cpf file directly when prompted, Framework Manager locates and opens the log.xml file. The advantage of directly opening the log.xml file is to reduce the number of large files that may need to be distributed in a multiuser environment.

5. Choose how to run the transactions:

- To run the entire transaction list continuously from start to finish, click **Run**.
- To run one transaction and then pause, click **Step**.

When a transaction is completed, a check mark or an "X" appears beside it. A check mark indicates that the transaction was applied successfully to the model you are merging into. An "X" means that the transaction failed. Detailed information for each transaction is listed under **Transaction Details**. If a transaction fails, the merge process pauses.

6. Choose one of the following:

Goal	Action
Fix a failed transaction	See "Fixing Failed Transactions" (p. 279) .
Skip the current transaction and run the one after it	Click Skip .
Run the current transaction and pause	Click Step .
Run the transaction list from the current transaction to the end	Click Continue .
Accept transactions run to this point and return to the project page	Click Accept .
Cancel all transactions run to this point and return to the project page	Click Revert .

7. Perform the previous step until you reach the end of the transaction list or you have accepted or reverted the changes.

If you accepted the changes, they appear in the **Project Viewer**. If you reverted the changes, none of the changes appear.

8. If you accepted the changes, save the merged project in the project page. If you decide not to save the changes, close the project without saving it.

Note: If you accept the changes in the **Perform the Merge** window but do not save the project you can never perform the same merge again. After you accept transactions they are removed from the transaction list.

Fixing Failed Transactions

During the merge process, transactions may fail. You can fix most transactions by substituting one object for another. You can fix all transactions by modifying the project.

Case 1:

An object used in the branch project is not in the root project. In the **Merge** dialog box, you see the **Replace** box indicating the name of the object that is missing from the root project.

To resolve this problem, you need to select an alternative object from the root project. Do the following:

1. From the **Project Viewer** or **Diagram**, select the name of the object to use in place of the missing object.

In the **Merge** dialog box, the object appears in the **Replacement Field**.

2. Click **Replace**.

The transaction runs again, substituting your replacement object for the missing object.

For more complex problems, you may be prompted to select the object that cannot be found in the root project from the **Object Naming Conflict Resolution** dialog box.

If a warning appears, you are unable to resolve the problem using this dialog box. Instead, do one of the following:

- modify the root project ([p. 279](#))
- skip the transaction ([p. 279](#))

Case 2:

For all failed transactions, you can resolve the problem by modifying the root project.

To modify a root project, do the following:

1. Modify the model as required.
2. From the **Perform the Merge** dialog box, click **Continue** to run the entire transaction list starting from the failed transaction.

Case 3:

You are unable to fix the transaction.

To resolve this problem, do the following:

1. Click **Skip** to skip the next transaction.
2. Clear the check box for the failed transaction.
3. Ensure the **Uncheck dependent transactions** check box is selected.
4. From the **Perform the Merge** dialog box, click **Continue**.

Segmenting and Linking Projects

You can use IBM® Cognos® Framework Manager to create and link segments, projects, and folders. A segment is a project within a main project. A segment is owned by its main project. A link is a shortcut to an existing project. The project that is linked can be shared by multiple projects.

A project segment is a complete project and changes to that project impact all projects to which it is linked. If you want to open a segment as a separate project, it must be structured as a complete project. There must be a physical layer in each segment that contains a subset of the data source query subjects on which they are based. These data source query subjects provide access to the data and metadata and must be included in the appropriate segments.

Do not change the physical layer in a segment. Any change will be reflected in the linked parent model and will impact all model segments that share data source query subjects. Changes may not be apparent outside the model in which they are made until the model is closed and reopened.

Before a project is segmented, ensure that the folder and namespace are named correctly. You cannot rename the folder or namespace after it has been segmented.

Changes made in the root model, such as upgrading and setting object security, are not inherited by the segmented model. You have to apply the changes to each segment of the model.

The governor settings that take precedence are the ones that apply to the model that is currently open (whether it is a parent model or a child model).

The main project has access to the entire model, including the segments. You can make changes to the segments when working in the main project, however, if the segment is being accessed by more than one user, the potential exists for updates to be lost.

Note: When changing the project structure, do not open the segments as individual projects. We recommend that you check out the main project and make changes from within it.

You can link the segments ([p. 283](#)) to other projects that contain related information to maintain consistency and reuse information. If you plan to link model segments, follow the recommendations for segmenting projects. We recommend that you link to relatively complete segments and regularly verify your model ([p. 251](#)).

Recommendations for Segmenting Projects

Understanding project segmentation is critical to ensure stability and minimize complexity in a multiuser modeling environment. If you intend to segment your project, we recommend that you do the following:

- ❑ Model the physical layer as completely as possible by:
 - ensuring that the namespace in the main project and any links in the project to folders have the same identifier ([p. 76](#))
- For example, you have a main project and a link in the project to a folder. The folder you link to must exist in a namespace that has the same name as the main project. If the identifier in the main project and that of the linked folder are not the same, any relationships, shortcuts, or expressions that reference objects in the link, from the main project, may not work.
- ensuring all objects in a project have unique identifiers

For example, you have a main project that contains a query subject named NewQS, and a segment in the project. You open the segment, add a new query subject named NewQS, and save the segment. When you open the main project, an error occurs because a query subject named NewQS already exists in the main project.

- updating references in both the main project and segments in the project

For example, you have a main project and a segment in the project. In the main project, you have a relationship named qs1_qs2 that exists between query subject1 and query subject2. The query subject named query subject 2 is in the segment. You open the segment, rename query subject2 to query subject3, and save the segment. When you open the main project, an error occurs because the relationship qs1_qs2 is broken. In Framework Manager, any object that relies on a reference, such as shortcuts, model query subjects, and expressions are also affected.

- ensuring that the main project and any segments in the project have the same languages

For example, you have a main project and a segment in the project. In the segment, you defined the languages English and French. You open the main project, add the language Chinese, and save the segment. When you open the segment, an error occurs because the language Chinese is not defined in the segment.

- ☐ Organize the physical layer using namespaces.

You should create a namespace for query subjects, calculations, and filters that you expect to be necessary for more than one segment.

You should create a namespace for each collection of query subjects that is unique to a planned model segment.

- ☐ Accept the default project name when creating the segmented project.

The segmented project must be created in a sub-folder within the master project folder. The default project name is the same as the folder or namespace that contains it.

- ☐ Segment the model ([p. 282](#)) for each namespace you created.

- ☐ Use a source control repository when possible to restrict access and track changes to your projects and segments.

Limitations of Segmenting and Linking Projects

The following limitations apply to segmenting and linking projects:

- You cannot test objects in a segment or linked project if they refer to objects that exist in an unavailable segment.
- You cannot create new objects in a segment or linked project if they refer to objects that exist in an unavailable segment.
- When you link to a project, all referenced objects (namespace objects, data sources, and parameter maps) are linked. Packages are not linked.

- Changes that you make to a child segment are not reflected in the main project, even after doing a refresh (F5). This happens because another child segment linked to the parent or the main project itself is open. Close all child segment projects and then reopen the main project.
- The point at which you create a segment in a project determines the point at which you can see the segment. If you create a nested segment from the main project, you can see the nested segment from the main project. If you open the segment containing the nested segment as a standalone project, you cannot see the nested segment. If you create a nested segment from a segment opened as a standalone project, you can see the nested segment from the standalone project. If you open the main project, you cannot see the nested segment created from the standalone segmented project.

Create a Segment

With segments, you can organize a project according to business rules and organizational requirements, and share and reuse project information.

You create segments at the folder level or the namespace level. You can create a new project in a new folder, complete with its own associated project files.

When a new segment is created, existing parameter maps from the main project are copied to the new segment. After the segment is created, parameter maps are unique to each segment and cannot be shared between segments. For example, if you are working in the main project, you can use a parameter map in a query subject belonging to a segment. However, if you open the segment, the parameter map is not available.

You can access a segment only from the project in which it was created. For example, you open the main project and create a segment (Segment A). Then you create another segment (Segment B) inside Segment A. From the main project, you can see Segment A and Segment B. However, if you open Segment A by itself, you do not see Segment B.

Before you create segments, consider dividing your project into business units. For example, you have a project named Sales. You can create two folders, one named Products and the other named Orders. You can divide the Sales project at the Products folder and at the Orders folder.

Steps

1. Click the folder or namespace you want to divide, and from the **Project** menu, click **Create Segment**.

We recommend that you accept the default settings for the project name.

2. To rename the segment, in the **Project Name** box, type a different name.

This does not change the folder name. If you want to rename the folder, you should rename it in **Project Viewer** before creating the segment.

For ease of use, we recommend keeping the same name for both the folder and the segment.

3. Click **OK**.

The **Project Viewer** is refreshed and the icons representing the segmented folder or the segmented namespace are shown.

Create a Link

You create links to help organize work across large projects, to maintain consistency, and to reuse information.

For example, the project named Inventory contains the folder named Products. You can create a link from the Sales Products to Inventory Products. If any changes or additions are made to the Inventory Products folder, you will see them in the Sales Products folder.

If you plan to link model segments, ensure that you follow the recommendations for model segmentation ([p. 280](#)).

A linked project is shared by other projects. It should not be created in a sub-directory within the master project directory.

You must create the project, folder, or namespace before you can link to it.

The projects you link must have and the same design language the same languages defined.

Steps

1. In the **Project Viewer**, click the project, segment, namespace, or folder that you want to link to.

Tip: You can create links only to folders, namespaces, projects, or segments.

2. From the **Project** menu, click **Link Segment**.
3. Locate and click the .cpf file of the project that contains the object that you want to link to.
4. Click **Open**.
 - If the project you selected requires upgrading, you are prompted. For more information, see "[Upgrading Models](#)" ([p. 365](#)).
 - If the project uses a mapped drive letter, you are prompted to keep the mapped drive letter or to change it to a UNC path.

You must choose the UNC path if your project will be shared by others.

5. Choose the project, segment, namespace, or folder to link to:
 - To link to another project, click **Add Project**, locate the .cpf file and click **Open**. Select the project and click **Add**.
 - To link to a segment, click the segment and click **Add**.
6. Click **OK**.

A new folder appears in the **Project Viewer**.

Leverage a Read-Only Project

You can make a read-only project available for other developers to leverage while protecting the project from unwanted changes.

Steps

1. Create a share that will host the project that is to be protected.
2. Give read-only access to that share for any developer leveraging the project.

Note: You can also make a project read-only by changing the file properties.

The appearance of the user interface changes when a project is read-only. Greyed out names, properties or actions indicate that no modifications can be made.

Using External Repository Control

You can use IBM® Cognos® Framework Manager with an external source control system. This procedure explains how to put the Framework Manager project files into an external repository. You can work on the project in Framework Manager and the external repository can manage the version control of the project files.

To use an external source control system, do the following:

- ☐ Ensure that the project is closed.
- ☐ Delete the repository.xml file, if it exists.
- ☐ Add all the files that exist in the project directory to the external repository.

Tip: The project directory is the directory that contains the <project name>.cpf file.

- ☐ Check the project files out of the external repository.
- ☐ Work on the project in Framework Manager.
- ☐ Save your changes.
- ☐ Check the project files into the external repository.

Segment a Project

You can create a segment for a project that is stored in an external repository. The segments are project directories that are stored under the main project directory. Maintain the same hierarchy in the repository as in the project directory.

The segments can be opened individually as stand-alone projects.

A segment can also be opened as part of the main project. In this situation, you must check out the project for each segment that you want to modify.

Administering the Metadata

You can change the metadata in your models to meet your specific modeling goals.

You can do the following:

- ❑ Copy, move, rename, or delete your projects to organize them in meaningful ways for your reporting environment (p. 285).
- ❑ Analyze the changes made to a model to see how they affect the packages and the reports that use the model (p. 287).
- ❑ Remap an object to a new source (p. 290).
- ❑ Export your model to exchange metadata between different data warehouse tools and repositories (p. 291).
- ❑ Reuse the same model and reports with different sets of data (p. 292).
- ❑ Move a model from one relational database to another (p. 294).

Copy, Move, Rename, or Delete a Project

You should organize projects in a meaningful way so that you can easily find them. Within IBM® Cognos® Framework Manager, you can copy (p. 285), move (p. 286), rename (p. 286), and delete (p. 287) projects.

You can manage your projects using segmenting (p. 282), and linking (p. 283). These project management features help organize a project according to business rules and organizational needs, set run time processing options, and give other users access to sections of the project.

You can also identify the vendor-specific functions (p. 314) that you want to use for each data source you import into your project.

If your project is segmented, the segments are treated as standalone projects. If you save or copy a project within an existing project, it is treated as a segment.

Copy a Project

When you copy a project, you create a replica of that project in another location. All files in the project folder, including sub-folders, are copied to the new location. When you make changes to the project in one folder, these changes are not reflected in copies of the project in other folders.

Copying a segmented model copies all segments as well as the main project.

There may be times when you cannot copy a project and must use **Save As** instead. Saving the project with a new name creates a new project folder while saving the project with the existing name overwrites the current project. This is useful if you want to save changes made to a read-only project or if you want to save a project with a different name or to a new location without overwriting the original project.

You cannot create a copy of a project in the same folder as the original. If you copy a project under an existing project folder, Framework Manager treats it like a project segment (p. 280).

If a project or segment is open when you copy it, the last saved version is copied.

Steps

1. From the **File** menu, click **Manage Projects, Copy**.

2. In the **From** box, click the browse button and select the .cpf file for the project you want to copy.
Note: The project folder name is shown in the text box.
3. In the **To** box, type the project name.
By default, the project name and the directory where the project is saved are the same.
4. In the **Location** box, type the new location or click the browse button and select the new project location.
5. Click **OK**.

Move a Project

You may decide to move a project if your folder becomes so full that it is difficult to locate particular projects. When you move a project, you are actually copying it to a new folder and deleting it from the current folder. All files in the project folder, including sub-folders, are moved to the new location.

Moving a segmented model moves all segments as well as the main project.

Before you can move a project, the project must be closed in Framework Manager.

Steps

1. From the **File** menu, click **Manage Projects, Move**.
2. In the **From** box, click the browse button and select the .cpf file for the project you want to move.
Note: The project folder name is shown in the text box.
3. In the **To** box, type the new location or click the browse button and select the new project location.
4. Click **OK**.

Rename a Project

When you rename a project, you provide a new name for the .cpf file. You are not changing the location of the project. Secondary project files and log files keep their original name.

If a project appears in the recent projects list on the Framework Manager **Welcome** page and you proceed to rename it, you cannot open the project by clicking the link. You must open the project using the **Open** command from the **File** menu.

Before you can rename a project, the project must be closed in Framework Manager.

Steps

1. From the **File** menu, click **Manage Projects, Rename**.
2. In the **From** box, click the browse button and select the .cpf file for the project you want to rename.

Note: The project folder name is shown in the text box.

3. In the **To** box, type the new name for the project and click **OK**.

If the original project folder and .cpf file have the same name, both the folder and .cpf file are renamed.

Delete a Project

When you delete a project, the project folder and all its contents, including any user files, are deleted from the file system and sent to the recycle bin.

If your project is segmented and you delete the main project, the segments are deleted as well. Deleting a project segment deletes only the segment and not the model it is based on.

We recommend that you delete segments from within the model. If you delete the segment using **Delete** from the **File** menu, it appears as if the segment still exists within the model. For more information, see ["Segmenting and Linking Projects" \(p. 280\)](#).

Before you delete a project, ensure that the project and all its segments are closed. Framework Manager does not support a file locking mechanism so it is possible under certain circumstances to delete a project with open segments. If you delete a project with open segments, the segments can no longer be saved.

Steps

1. From the **File** menu, click **Manage Projects, Delete**.
2. In the **Project Folder** box, click the browse button and select the .cpf file for the project you want to delete.

Note: The project folder name is shown in the text box.

3. Click **OK**.

The project folder and all its contents are deleted.

Analyze the Impact of Changes to a Package

Before publishing packages and running reports, you can see how the changes you make to a model will affect the package and the reports that use it. You can find the changes that were made to the package, and see details about each change and which reports are affected by a specific selected change.

Reports that are created using the package may be impacted by changes that you made to the model. For example, adding new objects to a package does not affect a report. Changing the name of a query item does affect a report. The report definition will not be valid because the query item is not in the package definition. If you use the durable model capability, you can avoid the impact that changing query item names has on reports. For more information, see ["Create a Durable Model" \(p. 189\)](#).

Note: Because a report uses a published package, if you make changes to the model, but do not publish the package that uses it, the report is not impacted by the changes.

If you change the name of an object, it shows up as "modified" in the results of the analysis.

The analysis is done on objects that a model uses directly, as well as the underlying objects. For example, you have a model query subject that is based on a data source query subject. If you change the model query subject, it will show up as a modified object. If you change the data source query subject, it will also show up as a modified object.

The following types of objects are analyzed: query subjects, query items, measures, regular dimensions, measure dimensions, hierarchies, levels, stand-alone filters, and stand-alone calculations.

Steps

1. In the **Project Viewer**, click a package that has been published.
2. From the **Actions** menu, click **Package, Analyze Publish Impact**.
3. Choose what you want to do:

Goal	Action
View report dependencies	See Find Report Dependencies .
View the dependencies for an object	See Show Object Dependencies .
See the details for an object	Click the row that contains the object. The details for the object appear under Change Details for .
Find an object in the Project Viewer	In the row that contains the object, under Actions , click Find in Project View .
Sort the results	Click Sort at the top of a column.
Display modeler's comments, last changed by, and last date changed	Click the double down arrow.

4. Click **Close**.

Find Report Dependencies

You can find the reports that use an object.

Steps

1. From the **Analyze Publish Impact** dialog box, do one of the following:
 - Select each object for which you want to determine the report dependencies by selecting individual check boxes.
 - Select all objects by selecting the check box at the top of the check box column.
2. Click **Find Report Dependencies**.

- Specify the scope of the search:

Goal	Action
Search all folders	Click All Folders .
Restrict the search to a specific folder	Click Restrict Search (Browse and select a folder) . Type the name of the folder or click Browse to search for a folder.

- Click **Search**.

A list of report names appears in the **Report Dependency** window under **Impacted Reports**. The results show both direct and indirect dependencies. The names of objects that indirectly impact reports are displayed in a lighter color.

- To sort the results, click **Sort** at the top of a column.
- Click **Close**.

Show Object Dependencies

You can find objects that depend on other objects, or show the dependencies of a child object.

Constraint

You cannot show dependencies for parameter maps.

Steps

- In the **Project Viewer**, click an object.
- From the **Tools** menu, click **Show Object Dependencies**.
The objects that depend on the selected object appear under **Dependent objects**.
- To show the object identifier for the dependent objects, select the **Show Object ID** check box.
- If the object has children and you want to see the dependencies for a child object, click the plus sign (+) beside the object that contains the child object.
- Click a child object under the parent object.

The objects that depend on the child object appear under **Dependent objects**.

Note: You can also show object dependencies from the following:

- The **Project Viewer** by right-clicking an object and selecting **Show Object Dependencies**.
- The **Context Explorer** window by right-clicking an object and selecting **Show Object Dependencies**.
- The **Analyze Publish Impact** window by clicking the **Show Dependencies** icon under **Actions** in the row that contains the object.

Remap an Object to a New Source

During the life cycle of an IBM® Cognos® Framework Manager model, you may need to change the data source it uses. For example, you may want to use the model against a different database with the same data, migrate the model from a transactional schema to a star or snowflake schema, or replace a previously existing database or import view with a new view. All of these actions can affect your reports. For example, if you change the names of objects, reports may no longer validate.

You can minimize the effect of model changes and data source changes by remapping higher level model objects so that they continue to run and return correct data. When you remap, you match and substitute object references or names in an original object, to object references or names in another object. You can remap query items and measures. You can remap individual objects manually or you can remap multiple objects at the same time. When remapping multiple objects, Framework Manager matches items in the original object to items in the other object using the matching criteria you specify. Only the objects that meet the matching criteria are remapped. You can use the object name or the object reference as the matching criteria for both the original and other objects.

When you change the matching criteria for remapping, you are specifying the criteria that will be used to remap to subsequent objects.

If a model query subject or model dimension contains a filter or calculation, the model filter or calculation is also remapped when you remap the model query subject or model dimension. You do not see a message or warning about this.

Note: We recommend that you validate all affected reports whenever you make changes to your model. To identify affected reports, see ["Show Object Dependencies" \(p. 289\)](#) and ["Find Report Dependencies" \(p. 288\)](#).

Constraints

You cannot remap data source query subjects or data source dimensions.

Remapping is only supported when using the design locale of the model.

Steps

1. In the Project Viewer, right-click an object and select **Remap To New Source**.
2. If you want to change the matching criteria, click **Options** and do the following:
 - Choose the matching criteria for the object you are using to remap, and for the original object that you are remapping.
You can match objects by name or by object reference.
 - The default criteria options are **By Name** for the object you are using to remap, and **By Object References** for the original object that you are remapping.
 - Click **OK**.
 - To use the criteria you specified, select the **Use matching criteria options** check box.

Notes:

- If the matching criteria is **By Name** to **By Name**, spaces within the string are removed.
 - If there is no object reference, the object name is used.
3. Do one or more of the following:

Goal	Action
Remap an individual object manually	Under Available Model Objects , drag an object to the object that you want to remap under Query Items, Measures, Calculations, and Filters . The new value for the object appears under Remap To .
Remap multiple objects automatically	Under Available Model Objects , drag a query subject to any row under Query Items, Measures, Calculations, and Filters . All of the objects that meet the matching criteria are remapped and their values appear under Remap To .
Change the expression for an object	Click the ellipsis (...) button beside the object. For information on how to create an expression, see " Create a Calculation " (p. 155).
Restore a remap value to the original source value	Right-click the row that contains the object that you want to restore, and select Restore to Original Value .
Clear the remap value and the original value for the selected object	Click the row that contains the object, and click Clear .
Clear the remap value for all objects	Click Clear All .

4. Click **OK** when you are finished remapping.

Export Metadata

You can export your IBM® Cognos® Framework Manager model as a Common Warehouse Metamodel (CWM) file. CWM exchanges metadata between different data warehouse tools and repositories. Each instance of the CWM metamodel is exchanged using XMI (.xml metadata interchange) documents.

When you export a Framework Manager model as a Common Warehouse Metamodel (CWM) file, joins, folders, namespaces, prompts, and calculations are not exported. Only query subjects, query items, and functions are exported.

When you export to CWM, we recommend that you use the default options, which optimize the metadata export. Only change these options if you have specific information that affects your export. For more information about export options, see the Meta Integration® web site.

Constraint

Do not use Japanese characters in the export path.

Steps

1. Right-click the root namespace of the metadata you want to export, and click **Export Model**.
You are prompted to save the project.

2. Select the export target.

3. In the **Framework Manager Specific Export Options** dialog box, click the options you want.

Note: We recommend that you use the default options. These default options optimize the metadata import. If you change the options, you may see unexpected results. To revert to the default options, click **Use Defaults**.

4. Click **Next**.

5. In the **Third Party Specific Export Options** dialog box, use the **File** option to identify the file to contain the exported metadata.

Click the other options you want.

In the **Option Description** pane, you see a description of the options available. The options are based on the selected data source. For more information, see the data source vendor documentation.

Note: We recommend that you use the default options. These default options optimize the metadata export. If you change the options, you may see unexpected results. To revert to the default options, click **Use Defaults**.

6. Click **Next**.

The input validation results from the export process appear.

7. Click **Next** and click **Finish**.

Project Reuse

You may have to use the same model and reports with different sets of actual data. The data sets may be different databases, accounts, or schemas in a single database. You may encounter multiple data sets

- when you use a different data set than used in production
- in large enterprises, where each division has its own data set
- in OEM applications, which have no direct control over customer data

The tables and columns used by the project must be logically the same across all data sets. You must also ensure that the correct data set is identified in each case.

Data sources in IBM® Cognos® Framework Manager contain information that identifies the location of any data source tables needed for the query subjects. This information is the name of the data source in the content store, as well as the optional catalog and schema names. Ensure that the catalog and schema names use the desired data set.

If different content stores are in use, and a different version of the project is deployed to each content store, you can specify the data source information in the project for each site. If you have only one content store, you can publish each project as a separate package. These solutions require a lot of manual maintenance. To reduce this level of maintenance, you can use one of the following options.

Determining the Data Source Information

The simplest solution is to determine the name of the data source in the content store, the name of the catalog, if applicable, and the name of the schema in the database. You can then use these names in all the data sets.

If some data sets use the same content store, create a separate connection for each data set in a single content store. For more information, see the IBM Cognos *Administration and Security Guide*. For information about how Framework Manager handles multiple connections, see "[Multiple Data Source Connections](#)" (p. 53).

Because the data source name in the content store can differ from the name of the customer database, this solution offers a lot of flexibility. However, it still requires that the catalog and schema names be identical across all data sets. Even if all the data sets use the same database type, this may be difficult to ensure. If different database types are involved, it may be impossible. For example, SQL Server has a catalog level, but Oracle does not.

Using User-based Default Data Set Identification

Each database user has access to a default schema and catalog, if applicable. If the schema and catalog are not defined, or if they are blank in the Framework Manager project data source, the default is used. As in the previous solution, this option may be combined with multiple connections so that different users can use different databases for the same data source.

However, when you edit a query subject, IBM® Cognos® Framework Manager uses the catalog and schema names in the data sources to match them to items that are dragged to the SQL windows from the data source tree. For this reason, the catalog and schema names cannot be blank in the project data source while you are modeling.

Therefore, you must use a macro expression in the catalog and schema of each data source in the project. This ensures that the catalog or schema names are blank at run time, but explicitly sets the catalog or schema you want while modeling.

Steps

1. Create a single session parameter whose value identifies whether you are in design mode. When you are in design mode, set the value of this session parameter to a specific value, such as design. Otherwise, leave the value empty.

Tip: If you use a project or override value, you must set it each time you open the model for editing.

2. For each catalog and schema in each project data source, create a parameter map that contains
 - an empty default value.
 - a key whose name is the design value of the session parameter above, and whose value is the name of the design mode catalog or schema for that data source.
3. Select the data source, and replace the catalog and schema property values with a macro that uses the corresponding parameter map and session parameter.

For example, use

```
#DBSchemaName ($DeployOrDesign) #
```

Model Portability

You can use a IBM® Cognos® Framework Manager model to access data from different database instances. The database instances can be from the same or different vendors.

There are several things to consider when moving a Framework Manager model from one relational database to another. Unlike changing from one identical database to another on the same platform, it may not be sufficient to change the data source connection information.

Review the generation of determinants and relationships based on indexes and do not assume that the indexes reliably describe functional dependencies or relationships for reporting.

Scalar functions are imported into a model prefixed by a catalog or schema qualification in the SQL statement. As with tables and views, you may have to remove or alter the location qualification when switching vendors. For example, if you create a model against an ORACLE database, and the connection is changed to point to an equivalent SQL Server database, an error results because the model data source type has remained OR instead of changing to the appropriate data source type.

To move a model from one relational database to another, do the following:

- ❑ Evaluate the DDL (Data Definition Language) to determine portability for physical names by
 - constraining physical names to a lowest common denominator, such as 31 characters.
 - avoiding using reserved key words in the ANSI standard and vendor documentation.
 - avoiding using vendor specific data fields.
 - avoiding conversions.
 - confirming that precision and scale is supported across all vendors.
 - using consistent and compatible collations.
 - using a consistent case on names, such as all lowercase.
- ❑ Evaluate the DDL to determine portability for database qualification.

- ❑ Evaluate the DDL to determine portability for data types in terms of compatibility and the precision and scale of data types.
- ❑ Review any native SQL statements in your models and reports for relational-specific syntax that may or may not be supported.
- ❑ Review usage of vendor-specific functions.
There may not be an equivalent vendor function or common function. A common function that is unsupported by the relational database may result in local processing that did not previously occur.
- ❑ Review the data source properties type.
If you change the RDBMS you use, such as from Oracle to SQL server, change the type property for the data source in Framework Manager.
- ❑ Update the data source queries.
When you import tables, Framework Manager imports physical information about the tables and columns that is used internally at run time. For example, collation information is reconciled only by rebuilding the physical tables.
- ❑ Test the moved model.
There will be other differences, such as performance characteristics, how data is ordered based on collations, and so on, that are revealed only by testing.

Moving a Model to Another Environment by Using Log Files

In IBM® Cognos® Framework Manager, you can view [\(p. 296\)](#) and play back actions [\(p. 296\)](#) performed on the project or you can use Script Player [\(p. 297\)](#) to play back transactions in batch mode. An action log is an XML file that contains a set of transactions. Each transaction has a sequence number and one or more actions. Each action is made up of a name and input parameters. Some actions also have output parameters. The action log file is in the project folder.

For example, you make changes to a project in a test environment. When it is time to move the project to production, you can use log files to play back every action, or series of actions, that you performed in the test environment to create an identical project in the production environment.

There are two action log files. The `log.xml` file contains all the transactions that have been run and saved in the project. This file is created the first time you save the project and exists until you delete the project. The temporary file contains transactions that have been run during the current session, but not saved. The temporary file is deleted when you close the project.

Note: If the script has dependencies on the existing project, you must ensure that the project is aligned with the script transactions to ensure the desired results.

A large log file may affect performance. You can archive entries [\(p. 301\)](#) in log files to reduce their size.

View and Save Transaction History

You can view the transaction history in an action log file and then save it as a script.

Steps

1. From the **Project** menu, click **View Transaction History**.

Tip: To make the dialog box larger, double-click the caption. Double-click again to restore the dialog box to its original size.

2. Click the transaction numbers that you want.

Tip: To view the details of a transaction, click the plus sign (+) next to a transaction number.

3. Click **Save as Script**.

4. Type a name for the file.

5. Click **Save**. Do not save the file in the **logs** folder.

6. Click **Close**.

Play Back Transactions From a Log File

You can choose to play back a specific transaction or a combination of transactions in a project or segment action log file.

When you play back transactions from a log file, the script player applies the commands in the log file to the contents of the existing model. Errors appear if objects created by the log file already exist in the model.

After the script in a log file has run successfully, a backup of the original project is created in the parent directory of the project. If you want to undo the transactions performed in the script, you can use the backup to restore the project to its original state.

You must disable or clear any commands that will conflict with the contents of the model. You can then run the script again. Or, you can use the **Synchronize** command, which begins with an empty model.

If you generate your own script outside Framework Manager, time stamps must be in ascending order with no duplicates.


Steps


1. From the **Project** menu, click **Run Script**.


2. Select the script you want, and click **Open**.

3. If you want to view the details of a transaction, click the transaction.





4. Set the starting or stop point that you want.

- To set the starting point for running the script, select the script and then click **Set the starting point**. You can do this at any time to skip an instruction or run instructions that have already been executed .

- To set a stop point for the script, select the script and then click **Set the stop point** . You can stop the script to make a manual fix and then start it again.

Tip: To remove the stop point, click **Remove the stop point** .

5. Using the toolbar buttons, choose the run action that you want.

Button	Description
	Runs the script After an error is encountered, clicking this button attempts to re-execute the failed instruction.
	Skips to the next transaction and runs the script to the end
	Runs the selected transaction only
	Skips to the next transaction and stops, but does not run any transactions

The project window is updated as the script is run.

6. Fix any errors encountered by the script either by retargeting objects or modifying the temporary project as required.

For more information, see ["Fixing Errors Caused by Invalid Objects" \(p. 300\)](#).

7. When the script has completed, click **Accept** to accept the changes or click **Revert** to undo the changes.

Note: After clicking **Accept** or **Revert**, you cannot use **Undo** and **Redo** for the current session.

Running Action Logs in Batch Mode

The Script Player is a command line utility that runs previously created action logs in batch. The Script Player interfaces with the FM engine and metadata directly, allowing you to bypass the Framework Manager application. You can use action logs that have been created either by the Framework Manager application or manually by using the reference material. It is ideally suited for automated tasks related to model creation and model maintenance. After careful analysis of your log files, you can use the options to run specific transactions.

The Script Player reads the command line parameters and either creates a new model or opens an existing one. After that, an action log consisting of a set of transactions is analyzed and each action is executed in sequence. Actions usually change the model. Before the Script Player terminates, the modified model is saved and replaces the original model. Using the options, you can suppress saving the model. This is useful if you are testing action logs. The Script Player maintains an internal log of all executed actions, allowing you to back out of any changes if necessary.

We do not recommend using action logs to upgrade models. To upgrade a model from ReportNet to IBM® Cognos®, you should open the model in Framework Manager and verify it, choosing the best method to fix reported errors.

You can use the Script Player on UNIX platforms, where the Framework Manager application is not supported.

Syntax

At the command prompt, ensure you navigate to the installation location of the BmtScriptPlayer.exe.

Use the following syntax to run the Script Player:

```
BmtScriptPlayer [-c|-m] <projectname> [-a <actionlogname>][options]
```

where <projectname> is the name of the project and <actionlogname> is the name of the action log.

For example,

```
BmtScriptPlayer -m goSales.cpf -a import.xml
```

Options

You can specify how the Script Player runs using the following options.

Note: If you are working in a UNIX environment, you may want to create a script to hide credentials that are passed on the command line.

Option	Description
-a FILEPATH	Apply the specified action log. FILEPATH is the path, including the file name, to the action log file.
-b NUM	Execute transactions with sequence number equal to or higher than the number specified by NUM. The default is the first transaction.
-c FILEPATH	Create a new project. FILEPATH is the path, including the file name, to the models project (.cpf) file. Using this option without specifying an action log results in the creation of an empty model. Note: If the model specified in the FILEPATH already exists, it is silently replaced.
-e NUM	Execute transactions with sequence number equal to or lower than the number specified by NUM. If the option is not specified, execution ends at the transaction with the highest sequence number or transaction number 9999, whichever comes first. For action logs that contain transactions with sequence numbers 10,000 and higher, this option must be used.

Option	Description
-g	<p>Upgrade the model (if required).</p> <p>If this option is not specified and the model was created with a previous version, execution terminates.</p> <p>If you specify this option without specifying an action log, only the model upgrade is performed.</p>
-k DIRECTORY	Specify the install directory.
-m FILEPATH	<p>Open an existing project.</p> <p>FILEPATH is the path, including the file name, to the models project (.cpf) file.</p>
-n	<p>Do not save the model.</p> <p>This option can be used to test action log files.</p>
-p PASSWORD	Authenticate using the specified password (if required).
-s NAMESPACE	Authenticate using the specified namespace (if required).
-t DIRECTORY	Specify the template directory.
-T PASSPORT	Specify a security passport. A passport is an encrypted string used to allow secure conversations for the plug-ins that need it.
-u USER	Authenticate using the specified user name (if required).
-x	<p>Terminate the test run when there is a transaction error.</p> <p>By default, the script player only terminates with severe errors such as an invalid model or action log, and continues executing, even if some minor transactions fail.</p>
-y PASSPORT	<p>Authenticate using the specified passport (if required).</p> <p>This option overrides other specified credentials (-s, -p, and -u). The Script Player skips authentication and associates the specified passport with the session.</p>

Examples

This table shows some examples of Script Player commands.

Command	Description
<code>BmtScriptPlayer -c <projectname></code>	Create a project.
<code>BmtScriptPlayer -c <projectname> -a <actionlogname></code>	Create a project and apply all the transactions from the action log.
<code>BmtScriptPlayer -c <projectname> -a <actionlogname> -b2 -e20</code>	Create a project and apply the transactions numbered 2-20 from the action log.
<code>BmtScriptPlayer -m <projectname> -a <actionlogname> -e20</code>	Open an existing project and apply the transactions numbered 1-20 from the action log.
<code>BmtScriptPlayer-m <projectname> -a <actionlogname> -n</code>	Open an existing project and apply all the transactions from the action log. Do not save the project.

Fixing Errors Caused by Invalid Objects

You may encounter errors when running script files or verifying models if an object that is referenced by a transaction no longer exists, or if you renamed objects.

If an object no longer exists, retarget the missing object to another object.

Working with Scripts

If you are working with scripts and you retarget an object, all remaining script transactions use the new object. If the script stops for any other reason, you should modify the temporary project to correct the problem.

Note: Fixing errors by making changes to the main project can produce unpredictable results. Always fix errors by changing the temporary project.

When a script encounters errors, you can choose how you want to resolve the problem.

Solution	Action
Skip transactions that include this object	<p>Click Exclude and in the Exclude Transactions that Use this Object dialog box, select the level of exclusion that you want.</p> <p>The current transaction and all subsequent ones that reference the excluded object are ignored. For example, if a transaction attempts to create a package that uses the excluded object, the package is not created.</p> <p>Note: We recommend that you attempt to fix errors before skipping transactions.</p>
Replace this and all following occurrences of the object	Click Replace and in the Replace Missing Objects dialog box, select the option that you want.

Solution	Action
Fix the problem manually	Click Stop and then fix the problem in the temporary project.

Retargeting an Object

If a transaction refers to an object that no longer exists, the script stops and a dialog box appears with the name of the problematic object. You can retarget the object by clicking **Replace** and selecting a new object.

If a missing object appears in an expression, the script stops and a dialog box appears with the name of the problematic object. You must fix the problem manually by opening the expression that contains the missing object.

Fixing Other Errors Encountered by the Script

You must fix script errors by modifying the temporary project. Fixing errors by making changes to the main project can produce unpredictable results.

Tip: You can move or minimize the **Synchronize** dialog box to view and modify the project.

Archive Log File Entries

Over time, log files for a project can become large. A large log file may affect the performance of IBM® Cognos® Framework Manager, especially when manipulating log files. You can remove a portion of the entries in a log file and append them to the contents of the archive-log.xml file ([p. 23](#)). Framework Manager archives all log file entries before the selected transaction. The selected transaction is not archived.

Archived transactions are available when synchronizing projects ([p. 301](#)).

Archived transactions are no longer visible in the transaction history ([p. 296](#)). These transactions are not available when branching or merging projects ([p. 273](#)). For example, in a branched project, any archived transactions will not be available when you merge back into the main project.

Steps

1. From the **Project** menu, click **View Transaction History**.
2. Locate the entry in a log file for the transaction that occurred after the last transaction you want to archive.
All transactions in the current log file prior to the selected transaction will be archived.
3. Click **Archive Log File**.

Synchronize Projects

You can use IBM® Cognos® Framework Manager log files to synchronize your project. You may choose to synchronize your project if you

- updated metadata in another party modeling tool.

- made changes to metadata using a multidimensional modeling tool.

When you synchronize your project, you create a new project by replaying from the log files, all the actions you made in the original project.

Special considerations should be taken before synchronizing projects that contain segmented models or linked models.

If your data source is a relational database, you can update only the query subjects and do not need to perform a full project synchronization. You must perform a project synchronization to synchronize changes made in another data source.

If you import a subset of a data source, any new objects that were added to the data source are not included when you synchronize. The action log recorded the importing of objects that you originally specified. When you synchronize, only the originally imported objects are re-imported.

You can use project synchronization to run the complete action history of the model and update the model's metadata. You can also save portions of the action log to a separate script file for later use, or you can save the entire action log to a script file if you want to build the same model in batch mode. If you encounter errors when trying to run an action log script, see ["Fixing Errors Caused by Invalid Objects" \(p. 300\)](#).

After synchronizing, you can choose to accept the new changes and create a new project, or return to the original project. If you accept the new changes, the original project is replaced.

Because every action that you made in your project is rerun, synchronization may take a long time.

If an object that is referenced by a transaction no longer exists, either because it was renamed or deleted, you will receive errors during the synchronization. For example, if you imported a table named Products and then renamed the table to New Products in your data source, you will receive an error when you synchronize the project. The synchronization cannot detect that the table named New Products was previously imported using a different name. You must manually retarget the object to complete the synchronization. For information about fixing synchronization errors, see ["Fixing Errors Caused by Invalid Objects" \(p. 300\)](#).

Note: Action logs from IBM Cognos ReportNet® are not supported in this release.

Before synchronizing a project, you should understand how synchronization impacts segmented ([p. 302](#)) and linked ([p. 303](#)) models. You should also ensure that data source connections have not changed and that data sources are online. You can check your connections by testing a few key query subjects.

Segmented Models

A segmented model should be synchronized only by synchronizing the main project. The results of synchronizing the entire project are written to the log file of the main project. The ability to synchronize individual segments is lost after the first synchronization of the main project.

If you are working in the main project and change a segment, the main log file is updated. If you are working in the segment and make changes, the segment log file is updated.

Synchronization commands do not necessarily run in the order they appear in the log files. This happens because it is possible to update segments concurrently and the action logs are replayed

based on the time of the original action. Commands may appear to jump between log files, making it difficult to use debugging features such as single stepping.

Linked Models



Log files are contained in the project that is open and not in the model that is updated.


If you open a main project and make changes to a linked model, the actions are logged in the log file of the main project. If you then synchronize the linked model, the change is lost because it did not appear in the set of log files that were used in the synchronization.





Synchronization can be run only on the main project or a stand-alone segment. You cannot synchronize linked projects or segments in the main project. If the segments are updated by the linked project, the synchronization can produce unpredictable results in the main project.

We recommended that you do not use model synchronization in combination with linked projects.

Steps to Synchronize

1. From the **Project** menu, click **Synchronize**.
2. We recommend that you create a backup of your Framework Manager project by selecting the **Backup project into this directory** check box.
3. If you want to view the details of a transaction, click the transaction.
4. Set the starting or stop point that you want.
 - To set the starting point for running the script, select the script and then click **Set the starting point**. You can do this at any time to skip an instruction or run instructions that have already been executed .
 - To set a stop point for the script, select the script and then click **Set the stop point** . You can stop the script to make a manual fix and then start it again.

Tip: To remove the stop point, click **Remove the stop point** .
5. Using the toolbar buttons, choose the run action that you want.

Button	Description
	Runs the script After an error is encountered, clicking this button attempts to re-execute the failed instruction.
	Skips to the next transaction and runs the script to the end
	Runs the selected transaction only
	Skips to the next transaction and stops, but does not run any transactions

The project window is updated as the script is run.

6. Fix any errors encountered by the script either by retargeting objects or modifying the temporary project as required.

For more information, see ["Fixing Errors Caused by Invalid Objects" \(p. 300\)](#).

7. When the script has completed, click **Accept**.

The original project is replaced by the contents of the temporary project.

Tip: To return the project to its previous state, click **Revert**.

Query Behavior

By monitoring and adjusting the behavior of queries in your project, you can improve the performance of your model.

You can do the following:

- ☐ Set governors to reduce system resource requirements and improve performance by ensuring that the metadata in a package contains the appropriate limits. For example, you can set limits on the amount of data retrieved or the time that a query can take ([p. 304](#)).
- ☐ Specify whether aggregate rollups are computed locally or in the database. For relational metadata, you can improve performance by selecting the right type of query processing. After initial report execution, by turning the query reuse feature on, you can create reports without querying the database again ([p. 311](#)).
- ☐ Improve performance by setting the query processing type to determine whether SQL processing is performed by the database server or processed locally ([p. 312](#)).
- ☐ Improve performance by reusing cached data when running a report. By reusing cached data, you can create a report without querying the database again ([p. 313](#)).
- ☐ Select the vendor-specific function sets for the data sources defined in the project ([p. 314](#)).
- ☐ Indicate the behavior of individual functions based on the data sources in the project ([p. 315](#)).
- ☐ Control and optimize how queries are run by modifying the properties of a data source that was created using the Metadata Wizard in IBM® Cognos® Framework Manager ([p. 317](#)).

Set Governors

Use governors to reduce system resource requirements and improve performance. You set governors before you create packages to ensure the metadata in the package contains the specified limits. All packages that are subsequently published use the new settings.

The governor settings that take precedence are the ones that apply to the model that is currently open (whether it is a parent model or a child model).

In a new project the governors do not have values defined in the model. You must open the **Governors** window and change the settings if necessary. When you save the values in the **Governors**

window by clicking **OK**, the values for the governors are set. You can also set governors in Report Studio. The governor settings in Report Studio override the governor settings in the model.

Maximum Number of Report Tables

You can control the number of tables that a user can retrieve in a query or report. When a table is retrieved, it is counted each time it appears in the query or report. The limit is not the number of unique tables. If the query or report exceeds the limit set for the number of tables, an error message appears and the query or report is shown with no data.

A setting of zero (0) means no limit is set.

Note: This governor is not used in dynamic query mode.

Maximum Number of Retrieved Rows

You can set data retrieval limits by controlling the number of rows that are returned in a query or report. Rows are counted as they are retrieved.

When you run a report and the data retrieval limit is exceeded, an error message appears and the query or report is shown with no data.

You can also use this governor to set limits to the data retrieved in a query subject test or the report design mode.

A setting of zero (0) means no limit is set.

If you externalize a query subject ([p. 263](#)), this setting is ignored when you publish the model.

Note: This governor is not used in dynamic query mode.

Query Execution Time Limit

You can limit the time that a query can take. An error message appears when the preset number of seconds is reached.

A setting of zero (0) means no limit is set.

Note: This governor is not used in dynamic query mode.

Large Text Item Limit

You can control the character length of BLOBs (binary large objects) that a user can retrieve in a query or report. When the character length of the BLOB exceeds the set limit, an error message appears, and the query or report is shown with no data.

A setting of zero (0) means no limit is set.

Outer Joins

You can control whether outer joins can be used in your query or report. An outer join retrieves all rows in one table, even if there is no matching row in another table. This type of join can produce very large, resource-intensive queries and reports.

Governors are set to deny outer joins by default. For example, outer joins are not automatically generated when you test a query item in Framework Manager.

SQL is generated automatically when you

- run a report
- test a query item or relationship in Framework Manager
- create a new model query subject based on other objects (p. 96)

If you keep the setting as **Deny**, you are notified only if you create a relationship in the Diagram tab that includes outer joins. You are not notified if you create a relationship in a data source query subject that includes outer joins.

If you set the governor to **Allow**, dimension to fact relationships are changed from inner joins to outer joins.

The outer joins governor does not apply in these circumstances:

- SQL that is generated by other means. If you set this governor to **Deny**, it does not apply to the permanent SQL found in a data source query subject, whether the SQL was generated on import (p. 58), manually entered, or based on existing objects (p. 85).
- Framework Manager needs to generate an outer join to create a stitched query. A stitched query is a query that locally combines the results of two or more sub-queries by using a locally processed outer join.

Note: This governor is not applicable for SAP BW data sources.

Note: This governor is not used in dynamic query mode.

Cross-Product Joins

You can control whether cross-product joins can be used in your query or report. A cross-product join retrieves data from tables without joins. This type of join can take a long time to retrieve data.

The default value for this governor is **Deny**. Select **Allow** to allow cross-product joins.

Shortcut Processing

You can control how shortcuts are processed by IBM Cognos software.

When you open a model from a previous release, the **Shortcut Processing** governor is set to **Automatic**. **Automatic** is a shortcut that exists in the same folder as its target and behaves as an alias, or independent instance. However, a shortcut existing elsewhere in the model behaves as a reference to the original. When you create a new model, the **Shortcut Processing** governor is always set to **Explicit**.

If you set the governor to **Explicit**, the shortcut behavior is taken from the **Treat As** property. If the **Shortcut Processing** governor is set to **Automatic**, we recommend that you verify the model and, when repairing, change the governor to **Explicit**. This changes all shortcuts to the correct value from the **Treat As** property based on the rules followed by the **Automatic** setting.

The **Shortcut Processing** governor takes priority over the **Treat As** property. For example, if the governor is set to **Automatic**, the behavior of the shortcut is determined by the location of the shortcut relative to its target regardless of the setting of the **Treat As** property is.

SQL Join Syntax

You can control how SQL is generated for inner joins in a model by selecting one of the following settings:

- If the governor is set to **Server determined**, the CQEConfig.xml file is used to determine the governor value. If there is no active CQEConfig.xml file or no parameter entry for the governor in the CQEConfig.xml file, then the **Implicit** setting is used.
- The **Implicit** setting uses the `where` clause.

For example,

```
SELECT publishers.name, publishers.id,
books.title FROM publishers, books WHERE publishers.id
= books.publisher_id ORDER BY publishers.name, books.title;
```

- The **Explicit** setting uses the `from` clause with the keywords `inner join` in an `on` predicate.

For example,

```
SELECT
publishers.name, publishers.id,
books.title FROM publishers INNER JOIN books ON publishers.id
= books.publisher_id ORDER BY publishers.name, books.title;
```

You can set the join type on the query property in Report Studio to override the value of this governor.

Regardless of the setting you use for this governor, the **Explicit** setting is used for left outer joins, right outer joins, and full outer joins.

This governor has no impact on typed-in SQL.

Grouping of Measure Attributes (query items)

If the governor is set to **Server determined**, the CQEConfig.xml file is used to determine the governor value. If there is no active CQEConfig.xml file or no parameter entry for the governor in the CQEConfig.xml file, then the **Disabled** setting is used.

The **Disabled** setting prevents aggregation of the measure for the attributes. This is the default behavior. For example,

```
select
Product.Product_line_code as Product_line_code,
Order_method.Order_method_code as Order_method_code, //measure attribute
XSUM(Sales.Quantity for Product.Product_line_code) as Quantity //aggregated
measure
from ...
```

The **Enabled** setting allows aggregation of the measure for the attributes. **Note:** This is the default behavior for IBM Cognos Framework Manager versions prior to 8.3.

```
select
Product.Product_line_code as Product_line_code, Order_method.Order_method_
code as Order_method_code, //measure attribute XSUM(Sales.Quantity for Order_
method.Order_method_code, Product.Product_line_code) as Quantity //
aggregated measure
from ...
```

SQL Generation for Level Attributes

You can control the use of the minimum aggregate in SQL generated for attributes of a level (member caption).

If the governor is set to **Server determined**, the CQEConfig.xml file is used to determine the governor value. If there is no active CQEConfig.xml file or no parameter entry for the governor in the CQEConfig.xml file, then the **Minimum** setting is used.

The **Minimum** setting generates the minimum aggregate for the attribute. This setting ensures data integrity if there is a possibility of duplicate records. For example,

```
select XMIN(Product.Product_line
for Product.Product_line_code) as Product_line, //level attribute
Product.Product_line_code as Product_line_code
from
(...) Product
```

The **Group By** setting adds the attributes of the level in the `group by` clause. with no aggregation for the attribute. The `distinct` clause indicates a `group by` on all items in the projection list. The **Group By** setting is recommended if the data has no duplicate records. It can enhance the use of materialized views and may result in improved performance. For example,

```
select distinctProduct.Product_line
as Product_line, //level attribute
Product.Product_line_code as Product_line_code
from
(...) Product
```

Note: This governor is not used in dynamic query mode.

SQL Generation for Determinant Attributes

You can control the use of the minimum aggregate in SQL generated for attributes of a determinant with the group by property enabled.

If the governor is set to **Server determined**, the CQEConfig.xml file is used to determine the governor value. If there is no active CQEConfig.xml file or no parameter entry for the governor in the CQEConfig.xml file, then the **Minimum** setting is used.

The **Minimum** setting generates the minimum aggregate for the attribute. This setting ensures data integrity if there is a possibility of duplicate records. For example,

```
select
PRODUCT_LINE.PRODUCT_LINE_CODE as Product_line_code,
XMIN(PRODUCT_LINE.PRODUCT_LINE_EN for PRODUCT_LINE.PRODUCT_LINE_CODE)
as Product_line //attribute
from
great_outdoors_sales..GOSALES.PRODUCT_LINE PRODUCT_LINE
group by
PRODUCT_LINE.PRODUCT_LINE_CODE //key
```

The **Group By** setting adds the attributes of the determinants in the `group by` clause with no aggregation for the attribute. This setting is recommended if the data has no duplicate records. It can enhance the use of materialized views and may result in improved performance. For example,

```
select
PRODUCT_LINE.PRODUCT_LINE_CODE as Product_line_code,
PRODUCT_LINE.PRODUCT_LINE_EN as Product_line //attribute
from
great_outdoors_sales..GOSALES.PRODUCT_LINE PRODUCT_LINE
```

```
group by PRODUCT_LINE.PRODUCT_LINE_CODE //keyPRODUCT_LINE.PRODUCT_LINE_  
EN //attribute
```

SQL Parameter Syntax

This governor specifies whether generated SQL uses parameter markers or literal values.

If the governor is set to **Server determined**, the CQEConfig.xml file is used to determine the governor value. If there is no active CQEConfig.xml file or no parameter entry for the governor in the CQEConfig.xml file, then the **Marker** setting is used.

You can override the value of this governor in Report Studio.

Dynamic SQL applications have the ability to prepare statements which include markers in the text which denote that the value will be provided later. This is most efficient when the same query is used many times with different values. The technique reduces the number of times a database has to hard parse an SQL statement and it increases the re-use of cached statements. However, when queries navigate larger amounts of data with more complex statements, they have a lower chance of matching other queries. In this case, the use of literal values instead of markers may result in improved performance.

Allow Enhanced Model Portability at Run Time

This governor is selected upon initial upgrade of a Cognos ReportNet® 1.x model. It prevents rigid enforcement of data types so that an IBM Cognos model can function as a ReportNet® 1.x model until you update the data types in the metadata. After you have verified that the model has been upgraded successfully, clear this governor.

Other than for initial upgrade, there are limited uses for this governor. For example, you have created a model for use with a data source and you want to run it against a different data source. The new data source must be structurally similar to the original data source, and the database schema must be the same between the two data sources. If you select this governor, IBM Cognos BI retrieves metadata from the data source and caches it instead of using the metadata already cached in the model. When you have completed modifying and testing the model against the new data source, clear this governor.

If you do not use this governor, you must ensure that the following metadata is the same in the original and new data sources:

- collation sequence name
- collation level
- character set
- nullability
- precision
- scale
- column length
- data type

Allow Usage of Local Cache

Select this governor to specify that all reports based on this model will use cached data. For a new model, this governor is enabled by default.

This setting affects all reports that use the model. Use Report Studio if you want a report to use a different setting than the model. For more information, see ["Improving Performance by Reusing Cached Data When Running a Report" \(p. 313\)](#).

Allow Dynamic Generation of Dimension Information

This governor is selected only upon initial upgrade of a ReportNet® 1.x model. This governor allows consistent behavior with ReportNet® 1.x by deriving a form of dimension information from the relationships, key information, and index information in the data source.

Use With Clause When Generating SQL

You can choose to use the `with` clause with IBM Cognos SQL if your data source supports the `with` clause.

The `with` clause is turned on for models created in IBM Cognos BI. For upgraded models, it is turned off unless it was explicitly turned on in the Cognos ReportNet® model prior to upgrading.

Suppress Null Values for SAP BW Data Sources

You can control whether or not nulls are suppressed by any report or analysis that uses the published package. The governor is also applied to test results during the current Framework Manager session. It is supported for SAP BW data sources only.

Some queries can be very large because null values are not filtered out. Null suppression removes a row or column for which all of the values in the row or column are null (empty). Null suppression is performed by SAP BW. This reduces the amount of data transferred to the IBM Cognos client products and improves performance.

By default, nulls values are suppressed. If you clear this governor, null values are not suppressed.

There is a property called **Suppress** in Report Studio that overrides this governor. If the **Suppress** property is set to **None**, null values are included in the result set even if the governor is set to suppress null values.

Note: This governor is not applied when creating CSV files; therefore, CSV files include null values if they exist in the data.

Publish Entire Model When Processing

A published package includes the model objects selected when the package was created. In addition, those model objects are analyzed in order to identify and include dependent objects in the package.

In a complex or very large model, the analysis can take considerable time. To shorten the publish time, set this governor to skip this analysis step and have the entire model written to the content store. The resulting package may be larger because the entire model is published instead of only required objects, however the time required to publish should be reduced.

Maximum external data sources that can be merged with a model

To use external data, report users import their data into an existing package. This governor controls the number of external data files that can be imported.

The default is 1.

For more information about external data sources, see the IBM Cognos Report Studio *User Guide*.

Maximum external data file size (KB)

To use external data, report users import their data into an existing package. This governor controls the size of each external data file.

By default, the maximum file size that report users can import is 2560 KB.

For more information about external data sources, see the IBM Cognos Report Studio *User Guide*.

Maximum external data row count

To use external data, report users import their data into an existing package. This governor controls the number of rows that can exist in each external data file.

By default, the maximum number of rows that report users can import is 20000.

For more information about external data sources, see the IBM Cognos Report Studio *User Guide*.

Specify Where Aggregate Rollups are Processed

The Rollup Processing property for data sources determines how aggregate rollups above the detail level in the report are computed. The default is set to local if local query processing is enabled, and is otherwise set to database.

Note: This property is not applicable for SAP BW data sources.

The possible options for this property are

- unspecified

The aggregation rollup is not specified.

- local

All aggregation rollups are computed locally (in the report server) using a running aggregate (for example, RSUM). Running aggregates spread the cost of this computation as the data is retrieved. Use this option if the local computer has more idle resources than the database computer, or if you find through experiment that it is the fastest method.

- database

Aggregation rollups are computed by the underlying database software if possible. Otherwise, they are computed locally (provided local query processing is enabled). Running aggregates are used, but the cost is incurred by the database server instead of the report server. Use this option if the database computer has more idle resources than the local computer, or if you find through experiment that it is the fastest method.

- extended

All aggregation rollups are computed by the database server using an extended aggregate (for example, XSUM). Extended aggregates incur the entire cost of this computation up front. Typically, this is the fastest method, but only where the database is set up to take advantage of materialized views. For databases where OLAP functionality is supported, this is translated into the appropriate OLAP aggregate functions.

Steps

1. In the **Project Viewer**, click the data source you want to change.
2. In the **Properties** pane, in the **Rollup Processing** list box, select the type of rollup processing that you want.

Improve Performance by Setting Query Processing Type

The query processing property for data sources determines whether SQL processing is performed by the database server or if it is processed locally. For relational metadata, you can improve performance by selecting the right type of query processing.

There are two types of query processing:

- limited local

The database server does as much of the SQL processing and execution as possible. However, some reports or report sections use local SQL processing.

- database only

The database server does all the SQL processing and execution. An error appears if any reports or report sections require local SQL processing.

Although the database server can usually run the SQL and run reports much faster, local processing is sometimes necessary. For example, choose limited local processing if you want to create cross database joins, or if you want your users to use unsupported SQL99 functions.

Some complex queries require limited local processing, such as a query that must generate an At clause to avoid double-counting.

Query Processing for Dynamic Query Mode

In Dynamic Query Mode, the query processing settings have an effect only on SAP BW data sources.

- limited local

Limited Local is not supported at this time.

- database only

Little, or possibly none of the query processing is performed by the report server. Local processing only occurs if the database cannot handle the query. Consider using this value only if your report performance is unacceptable with the default setting, and becomes usable with this setting.

Be aware that results may change with this setting; test carefully to confirm that the results are still correct.

Steps

1. In the **Project Viewer**, click the data source you want to change.
2. In the **Properties** pane, in the **Query Processing** list box, click either **Limited Local** or **Database Only**.

Improving Performance by Reusing Cached Data When Running a Report

When you run a report, the query request is sent to the database and the result set is returned. After the initial report execution, you may decide to make changes to the report. Often, the report can be created without querying the database again. To take advantage of this, turn the query reuse feature on.

When query reuse is turned on and you run a report for the first time, the query is stored in the cache on the report server. Also, some data source resources may not be available until the transaction using them is released. The current default time-out is 60 minutes. If certain database activities involve modifying database objects, you must wait for the time-out period to be completed, or you can disable query reuse for reports.

The first time the report is run and the cache is created, the response time may be slightly negatively impacted. The performance improvement is realized by the report consumer on each subsequent report execution, when the response time is improved by as much as 80%. This performance improvement occurs because the report does not have to re-query the database. In addition to this, reduced queries to the database yields improved overall system performance, which positively impacts all users.

Query reuse can be set on the model or on individual reports. To specify that all reports using a particular model should use cached data, enable the **Allow Usage of Local Cache** governor on the model in IBM® Cognos® Framework Manager and republish the model. By default, this setting affects all reports that use this model, including analyses that are run as reports in IBM Cognos Viewer.

Query Reuse in IBM Cognos Viewer

If you want a report to use a different setting than the model, you can do this in IBM Cognos Report Studio. In the **Properties** pane, change the **Use Local Cache** property. Set the property to **No** if you want to always execute the query. Set the property to **Yes** if you want to use cached results. If you want the report to use the same setting as the model, change the setting to **Default**.

Changing the **Use Local Cache** property for one report does not affect other reports.

Reusing Cached Data in Query Studio

IBM Cognos Query Studio reuses cached data under various conditions. If query reuse is turned on in the model and the action can be satisfied by a subset of the cached data set, the report uses the cached data. For example, changes to the report such as adding a filter or removing a column, may change the report data, but the request can still be satisfied from a subset of the cached data.

If query reuse is turned off and the action can be satisfied from the cached data set without modifications, the report still uses the cached data. For example, changing the report format uses the previous data set even if query reuse is turned off. This is known as cursor reuse. Cursor reuse is used when the cached data can satisfy the request without modifications.

Reports that were created in Query Studio always use the same setting as that specified in the model. If the model has query reuse turned on, the report attempts to use the cached data.

Deciding Whether to Use Query Reuse in Your Environment

Before deciding whether or not to turn query reuse on, consider the following:

- If most report consumers run reports interactively but run them only once, you may not experience a high level of performance improvement by caching data.

Note: Regardless of the query reuse settings, reports that run in batch mode do not cache data.

- The size of the cache may impact scalability. For example, if a report has a large result set, the cache will also be large. This should be taken into account when sizing and configuring your server environment.

Select Function Sets

A collection of vendor-specific functions is called a function set. When you create a project that contains relational metadata, the expression editor lists the function sets for all available vendors. However, you can restrict the function sets so that they list only the vendors that you want to use in your project. You customize the function set by identifying the specific vendor for each data source defined in the project.

You can use functions that you defined in your relational data source in IBM® Cognos® Framework Manager. If you imported the user-defined functions, they are listed in Framework Manager for easy selection. If you did not import them, you can type the name of the function into an expression. If the function must be qualified, you must import them into Framework Manager.

Sometimes the vendor-specific functions were created on schemas with broad access permissions. You cannot use these functions in IBM Cognos software on a schema with restricted access permissions if both the schemas are on the same database instance.

If an unrecognized function is typed into a report, it is assumed that the function is native. For more information, see ["Native SQL" \(p. 110\)](#).

Note: When you create a project that contains SAP BW metadata, Framework Manager automatically lists only the functions that apply to SAP BW data.

Steps

1. From the **Project** menu, click **Project Function List**.
2. Select the **Set function list based on the data source type** check box.
Tip: To disable this filter, select the **Include all function sets** check box.
3. In the **Function set** page, click the appropriate data source row.

4. From the drop down list on the **Function set** field, select the function set you want to use with this data source.
5. Repeat steps 2 to 4 until finished.
6. Click **OK**.

Quality of Service

With IBM® Cognos® Framework Manager, you can query any combination of data source types, but not all data sources support functions the same way. The quality of service indicator provides you and your users with a visual clue about the behavior of individual functions when used in conjunction with the data sources in the model.

Each function specified in your data source may have a different quality of service, depending on the type of data source in use. For each query feature that does not have the same quality of service across packages, you can override the level of service and add text to describe the specific situation in that model. Your users can see the quality of service indicators and the context specific description, and use this information when determining which functions to use in reports.

The quality of service for a function is specified at the data source level and can be set for an individual function ([p. 381](#)) or for all functions in a package. The quality of service indicators are:

- not available (X)
This function is not available for any data sources in the package
- limited availability (!!)
The function is not available for some data sources in the package
- limited support (!)
The function is available for all data sources in the package but is not naturally supported for that data source. IBM Cognos software uses a local approximation for that function. Because an approximation is used, performance can be poor and the results may not be what you expect.
- unconstrained (check mark)
The function is available for all data sources

If there is more than one type of data source in the model, the quality of service values are aggregated according to the following rules:

- If the quality of service is defined as Unconstrained, Limited Support, or Limited Availability in one data source and defined as Not Available in another data source, the quality of service for that function becomes Limited Availability.
- In all other cases, the lowest common dominator is used. For example, if the quality of service is Unconstrained in one data source and Limited Support in another data source, the quality of service for that function becomes Limited Support. If the quality of service is Limited Support in one data source and Limited Availability in another data source, the quality of service is reported as Limited Availability.

Impact of Overriding the Quality of Service Indicator

IBM® Cognos® Framework Manager determines the quality of service for functions based upon the data source type. Taking into consideration the context of the model, you can override the quality of service that is determined by the product. Overriding the quality of service provides guidance to your users. It does not change the level of support for that function in your data source.

When a package is made by combining sub-packages, quality of service overrides in the parent package take precedence. If there is no parent override, the quality of service for the child packages are aggregated as described above.

Consider Your Users

Ultimately, the goal is to provide your users with enough information to satisfy their business requirements, but not enough to confuse them. If your users are unable to make decisions regarding which functions to use based on the quality of service indicators, you should consider publishing separate packages for different groups of users. If your users require access to functions whose quality of service is less than Unconstrained, you should document the restrictions of those functions when you set the quality of service.

Impacts on Performance

The quality of service indicators has no direct impact on query performance. Service indicators are intended to give you some control over which functions are available for use. You can then prevent your users from using functions that could result in long running queries or queries that fail.

It is important to note that if you use functions that are not available in your data source, IBM® Cognos® Framework Manager tries to compensate by using local processing on the report server. This may have an impact on query performance because the work is done on your report server instead of on your data source server.

In some situations, local processing may require more data to be retrieved from the data source server, which has an impact on both the data source server and the network. For example, OLAP functions are not available in a relational data source. If you attempt to use OLAP functions with a relational data source, Framework Manager uses the dimensional information in the data source to generate a local cube and run the OLAP functions against the cube. This requires retrieval of dimensional information from the data source server and extra processing on the report server.

Steps

1. From the **Project** menu, select **Project Function List**.
2. Click **Define Quality of Service**.
3. Expand the tree nodes to view the quality of service for each function.
4. To override the quality of service, click the arrow beside each function and select the quality of service indicator from the list.
5. After changing the quality of service, you can add detailed information about the function in the text box on the right.

This information becomes available to your users and can assist them in determining whether to use this function in their reports.

Tip: Click **Remove override** to set the quality of service back to the default.

6. Click **OK**.

Control and Optimize How Queries Are Run

You can modify the properties of a data source that was created using the **Metadata Wizard** in Framework Manager. The data source properties help you control and optimize the way queries are run against the database.

You cannot modify the properties of a data source that was created using the portal. These data sources can only be modified in the portal. For more information, see the IBM Cognos *Administration and Security Guide*.

Data Source Property	Description
Name	Descriptive name of the data source connection provided by the user at the time of creation
Query Processing	Determines whether SQL processing is performed by the database server or processed locally
Rollup Processing	Determines whether aggregate rollups are computed locally or in the database
Content Manager Data Source	Specifies the name of the data source as it is identified in the Content Manager. If using an XML data source, this property may be parameterized.
Catalog	Represents different information for different databases. For example, if the database is SQL Server, the element contains the name of the database; if the database is Oracle, it is not used.
Cube	Specifies the name of the cube
Schema	Represents different information for different databases. For example, for SQL Server or Oracle, the element contains the name of the owner.
Type	Specifies the type for the parent object.

Data Source Property	Description
Query Type	Specifies the type of query model that this data source understands. For example, SQL sources are relational and MDS sources are multidimensional.
Query Interface	This element contains two letters, identifying the provider type. It is maintained by the application.
Function Set ID	Defines the function set that applies to a data source . Used in the initial population of the function sets of a security view when a package is created.
Parameter Maps	References a parameterMap that represents a Oracle Essbase alias table map.

Chapter 9: Guidelines for Modeling Metadata

IBM® Cognos® Framework Manager is a metadata modeling tool that drives query generation for IBM Cognos BI. A model is a collection of metadata that includes physical information and business information for one or more data sources. IBM Cognos BI enables performance management on normalized and denormalized relational data sources as well as a variety of OLAP data sources.

"[Understanding IBM Cognos Modeling Concepts](#)" (p. 319) discusses fundamental IBM Cognos modeling concepts that you need to understand about modeling metadata for use in business reporting and analysis.

"[Building the Relational Model](#)" (p. 338) discusses building the relational model.

For information about modeling for use with dynamic query mode, see the *Dynamic Query Guide*.

To access the IBM Cognos *Guidelines for Modeling Metadata* documentation in a different language, go to *installation_location*\c10\webcontent\documentation and open the folder for the language you want. Then open ug_best.pdf.

Understanding IBM Cognos Modeling Concepts

Before you begin, there are concepts that you need to understand.

Relational modeling concepts:

- cardinality ([p. 320](#))
- determinants ([p. 322](#))
- multiple-fact, multiple-grain queries ([p. 326](#))

Model design considerations:

- relationships and determinants ([p. 329](#))
- minimized SQL ([p. 330](#))
- metadata caching ([p. 332](#))
- query subjects vs. dimensions ([p. 332](#))
- shortcuts vs. copies of query subjects ([p. 333](#))
- folders vs. namespaces ([p. 334](#))
- order of operations ([p. 334](#))
- impact of model size([p. 336](#))

Dimensional modeling concepts:

- regular dimensions ([p. 336](#))
- measure dimensions ([p. 337](#))

- scope relationships ([p. 337](#))

Relational Modeling Concepts

When modeling in IBM® Cognos® Framework Manager, it is important to understand that there is no requirement to design your data source to be a perfect star schema. Snowflaked and other forms of normalized schemas are equally acceptable as long as your data source is optimized to deliver the performance you require for your application. In general, we recommend that you create a logical model that conforms to star schema concepts. This is a requirement for IBM Cognos Analysis Studio and has also proved to be an effective way to organize data for your users.

When beginning to develop your application with a complex data source, it is recommended that you create a simplified view that represents how your users view the business and that is designed using the guidelines in this document to deliver predictable queries and results. A well-built relational model acts as the foundation of your application and provides you with a solid starting point if you choose to take advantage of dimensional capabilities in IBM Cognos software.

If you are starting with a star schema data source, less effort is required to model because the concepts employed in creating a star schema lend themselves well to building applications for query and analysis. The guidelines in this document will assist you in designing a model that will meet the needs of your application.

Cardinality

Relationships exist between two query subjects. 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; this expression usually refers to the primary and foreign keys of the underlying tables.

IBM Cognos software 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

A query that uses multiple facts from different underlying tables is split into separate queries for each underlying fact table. Each single fact query refers to its respective fact table as well as to the dimensional tables related to that fact table. Another query is used to merge these individual queries into one result set. This latter operation is generally referred to as a stitched query. You know that you have a stitched query when you see `coalesce` and a full outer join.

A stitched query also allows IBM Cognos software to properly relate data at different levels of granularity ([p. 326](#)).

Cardinality in Generated Queries

IBM Cognos software supports both minimum-maximum cardinality and optional cardinality.

In $0:1$, 0 is the minimum cardinality, 1 is the maximum cardinality.

In $1:n$, 1 is the minimum cardinality, n is the maximum cardinality.

A relationship with cardinality specified as 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 may be specified as 1:1 to 0:n. This indicates that reports will show the requested customer information even though there may not be any sales data present.

Therefore a 1 to n relationship can also be specified as:

- 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 cardinality. For example, Sales Staff (1:1) is joined to Orders (0:n).

Relationship impact:	Each Order has one and only one Sales Staff. Each Sales Staff has zero or more Order (outer join).
----------------------	---

It is important to ensure that the cardinality is correctly captured in the model because it determines the detection of fact query subjects and it is used to avoid double-counting factual data.

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

- Cardinality is applied in the context of a query.
- 1 to n cardinality implies fact data on the n side and implies 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.

Use the **Model Advisor** to see an assessment of the behavior implied by cardinality in your model.

For more information, see ["Single Fact Query" \(p. 351\)](#) and ["Multiple-fact, Multiple-grain Query on Conformed Dimensions" \(p. 352\)](#).

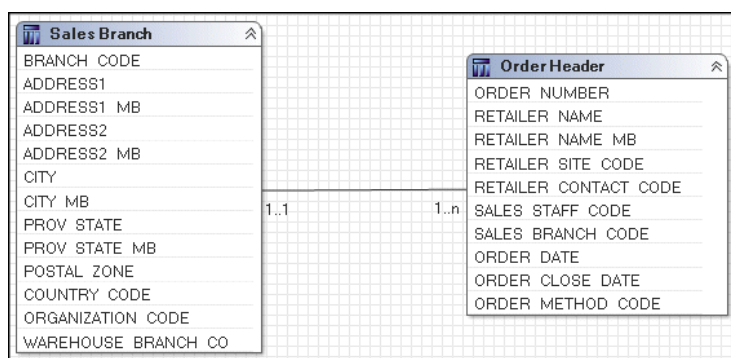
Cardinality in the Context of a Query

The role of cardinality in the context of a query is important because cardinality is used to determine when and where to split the query when generating multiple-fact queries. If dimensions and facts are incorrectly identified, stitched queries can be created unnecessarily, which is costly to performance, or the queries can be incorrectly formed, which can give incorrect results ([p. 361](#)).

The following examples show how cardinality is interpreted by IBM Cognos software.

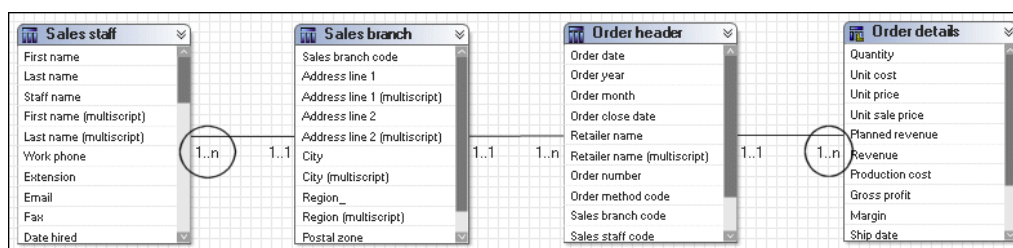
Example: Query Subjects Behaving as a Dimension and a Fact

In this example, Sales Branch behaves as a dimension relative to Order Header and Order Header behaves as a fact relative to Sales Branch.



Example: Four Query Subjects Included in a Query

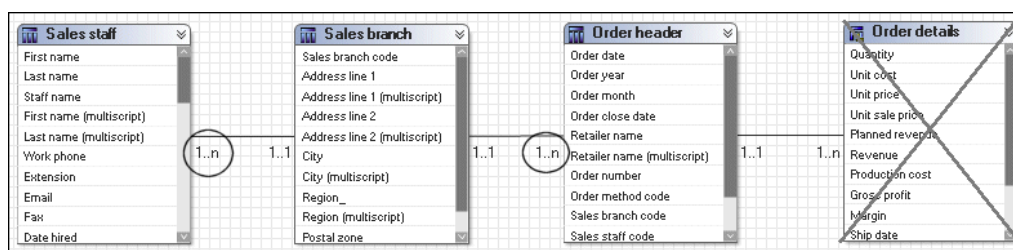
In this example, all four query subjects are included in a query. Sales staff and Order details are treated as facts. Order header and Sales branch are treated as dimensions.



The SQL generated for this query will be split, treating Sales staff and Order details as facts. The results of these two subqueries are stitched using the information retrieved from Sales branch. This gives a report that lists the Sales staff information by Sales branch next to the Order details and Order header information by Sales branch.

Example: Three Query Subjects Included in a Query

In this example, only three query subjects are included in a query. Order details is not used. Order header is now treated as a fact. Sales staff continues to be treated as a fact.



The SQL in this example also generates a stitched query, which returns a similar result as above. Note that a stitch operation retains the information from both sides of the operation by using a full outer join.

Determinants

Determinants reflect granularity by representing subsets or groups of data in a query subject and are used to ensure correct aggregation of this repeated data. Determinants are most closely related to the concept of keys and indexes in the data source and are imported based on unique key and index information in the data source. We recommend 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.

An example of a unique determinant is Day in the Time example below. An example of a non-unique determinant is Month; the key in Month is repeated for the number of days in a particular month. When you define a non-unique determinant, you should specify **Group By**. This indicates to IBM Cognos software that when the keys or attributes associated with that determinant are repeated in the data, it should apply aggregate functions and grouping to avoid double-counting. It is not recommended that you specify determinants that have both **Uniquely Identified** and **Group By** selected or have neither selected.

Year Key	Month Key	Month Name	Day Key	Day Name
2006	200601	January 06	20060101	Sunday, January 1, 2006
2006	200601	January 06	20060102	Monday, January 2, 2006

You can define three determinants for this data set as follows -- two **Group By** determinants (Year and Month) and one unique determinant (Day). The concept is similar but not identical to the concept of levels and hierarchies.

Name of the Determinant	Key	Attributes	Uniquely Identified	Group By
Year	Year Key	None	No	Yes
Month	Month Key	Month Name	No	Yes
Day	Day Key	Day Name Month Key Month Name Year Key	Yes	No

In this case, we use only one key for each determinant because each key contains enough information to identify a group within the data. Often Month is a challenge if the key does not contain enough information to clarify which year the month belongs to. In this case, however, the Month key includes the Year key and so, by itself, is enough to identify months as a sub-grouping of years.

Note: While you can create a determinant that groups months without the context of years, this is a less common choice for reporting because all data for February of all years would be grouped together instead of all data for February 2006 being grouped together.

Using Determinants with Multiple-Part Keys

In the Time dimension example above, one key was sufficient to identify each set of data for a determinant but that is not always the case.

For example, the following Geography dimension uses multiple-part key definitions for all but one determinant.

Region	Country Key	State/Province Key	City Key
North America	USA	Illinois	Springfield
North America	USA	Missouri	Springfield
North America	USA	California	Dublin
Europe	Ireland	n/a	Dublin

Similar to the example about Time, you can define three determinants for this data set as follows - two **Group By** determinants (Country and State/Province) and one unique determinant (City).

Name of the Determinant	Key	Attributes	Uniquely Identified	Group By
Country	Country Key	None	No	Yes
State/Province	State/Province Key	None	No	Yes
City	Country Key State/Province Key City Key	None	Yes	No

In this case, we used Country Key, State/Province Key, and City Key to ensure uniqueness for City. We did this because in the data we were given, some city names were repeated across states or provinces, which in turn were repeated for countries.

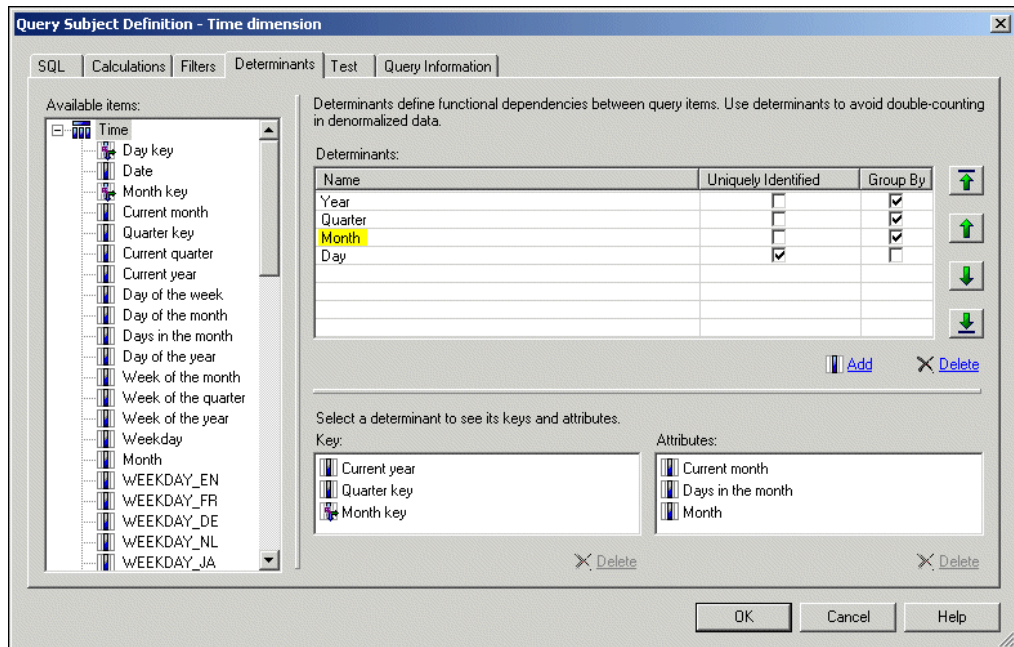
Determinants Are Evaluated in the Order In Which They Are Specified

There is no concept of a hierarchy in determinants, but there is an order of evaluation. When IBM Cognos software looks at a selection of items from a query subject, it compares them to each determinant (keys and attributes) one at a time in the order that is set in the **Determinants** tab. In this way, IBM Cognos software selects the determinant that is the best match.

In the following example, the attributes current month, days in month, and localized month names are associated to the Month key. When a query is submitted that references any one of these attributes, the Month determinant is the first determinant on which the matching criteria is satisfied. If no other attributes are required, the evaluation of determinants stops at Month and this determinant is used for the `group` and `for` clauses in the SQL.

In cases where other attributes of the dimension are also included, if those attributes have not been matched to a previous determinant, IBM Cognos software continues evaluating until it finds a match or reaches the last determinant. It is for this reason that a unique determinant has all query

items associated to it. If no other match is found, the unique key of the entire data set is used to determine how the data is grouped.



When to Use Determinants

While determinants can be used to solve a variety of problems related to data granularity, you should always 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.

For example, Time has multiple levels, and it is joined to Inventory on the Month Key and to Sales on the Day Key. For more information, see ["Multiple-fact, Multiple-grain Queries"](#) (p. 326).

- There is a need to count or perform other aggregate functions on a key or attribute that is repeated.

For example, Time 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. In SQL, that is `XMIN(Days in the month for Month_Key)`. There is also a `Group by` clause in the Cognos SQL.

There are less common cases when you need to use determinants:

- 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. Override the determinants imported from the data source that conflict with relationships created for reporting.

You cannot use multiple-segment keys when the query subject accesses blob data. With summary queries, blob data must be retrieved separately from the summary portion of the query. To do this, you need a key that uniquely identifies the row and the key must not have multiple segments.

- A join is specified that uses fewer keys than a unique determinant that is specified for a query subject.

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 relationships, there will be a conflict. 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. Modify the determinant information of the query subject if it is not appropriate to use the additional columns in the relationship.

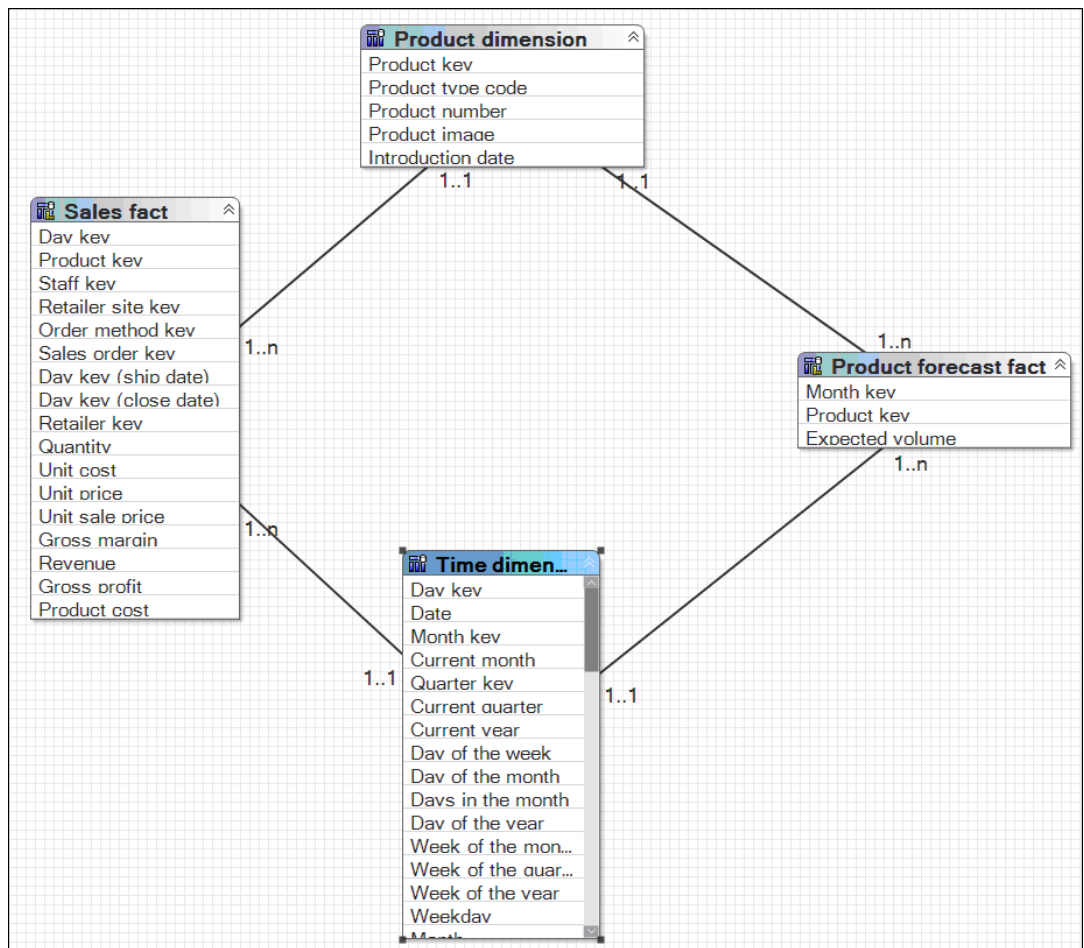
Multiple-fact, Multiple-grain Queries

Note that in this section, the term dimension is used in the conceptual sense. A query subject with cardinality of 1:1 or 0:1 behaves as a dimension. For more information, see "[Cardinality](#)" (p. 320).

Multiple-fact, multiple-grain queries in relational data sources occur when a table containing dimensional data is joined to multiple fact tables on different key columns. A dimensional query subject typically has distinct groups, or levels, of attribute data with keys that repeat. The IBM Cognos studios automatically aggregate to the lowest common level of granularity present in the report. The potential for double-counting arises when creating totals on columns that contain repeated data. When the level of granularity of the data is modeled correctly, double-counting can be avoided.

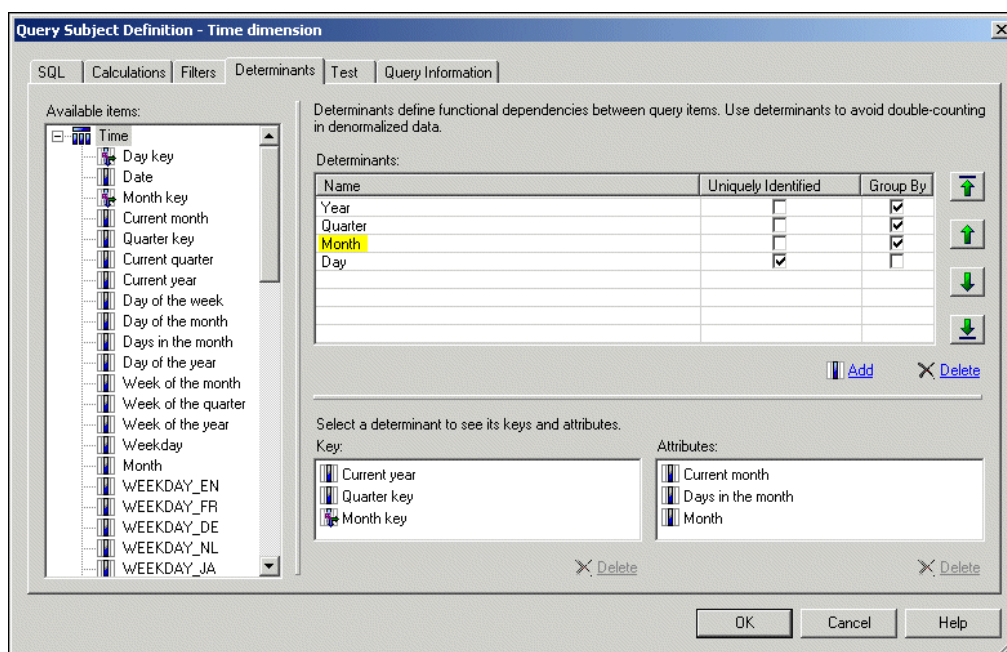
Note: You can report data at a level of granularity below the lowest common level. This causes the data of higher granularity to repeat, but the totals will not be affected if determinants are correctly applied.

This example shows two fact query subjects, Sales and Product forecast, that share two dimensional query subjects, Time and Product.



Time is the focal point of the granularity issue in this example. Sales is joined to Time on the Day key, and Product forecast is joined to Time on the Month key. Because of the different join keys, a minimum of two determinants must be clearly identified on Time. For example, the determinants for Month and Day have their keys identified. Day is the unique key for Time, Month keys are repeated for each day in the month.

For example, the determinant for Month is as follows.



The Product query subject could have at least three determinants: Product line, Product type, and Product. It has relationships to both fact tables on the Product key. There are no granularity issues with respect to the Product query subject.

By default, a report is aggregated to retrieve records from each fact table at the lowest common level of granularity. If you create a report that uses Quantity from Sales, Expected volume from Product forecast, Month from Time, and Product name from Product, the report retrieves records from each fact table at the lowest common level of granularity. In this example, it is at the month and product level.

To prevent double-counting when data exists at multiple levels of granularity, create at least two determinants for the Time query subject. For an example, see "[Determinants](#)" (p. 322).

Month	Product name	Quantity	Expected volume
April 2007	Aloe Relief	1,410	1,690
April 2007	Course Pro Umbrella	132	125
February 2007	Aloe Relief	270	245
February 2007	Course Pro Umbrella		1
February 2006	Aloe Relief	88	92

If you do not specify the determinants properly in the Time query subject, incorrect aggregation may occur. For example, Expected volume values that exist at the Month level in Product forecast is repeated for each day in the Time query subject. If determinants are not set correctly, the values for Expected volume are multiplied by the number of days in the month.

Month	Product name	Quantity	Expected volume
April 2007	Aloe Relief	1,410	50,700
April 2007	Course Pro Umbrella	132	3,750
February 2007	Aloe Relief	270	7,134
February 2007	Course Pro Umbrella		29
February 2006	Aloe Relief	88	2,576

Note the different numbers in the Expected volume column.

Model Design Considerations

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, it is important to understand the application requirements for functionality, ease of use, and performance. The design of the data source and application requirements will determine the answer to many of the questions posed in this section.

Where Should You Create Relationships and Determinants?

A frequently asked question is where to create relationships. Should relationships be created between data source query subjects, between model query subjects, or between both? The answer may vary because it depends on the complexity of the data source that you are modeling.

When working with data source query subjects, relationships and determinants belong together.

When working with model query subjects, there are side effects to using relationships and determinants that you should consider:

- The model query subject starts to function as a view, which overrides the **As View** or **Minimized** setting in the **SQL Generation** type for a query subject.

This means that the SQL stays the same no matter which items in the query subject are referenced. For more information, see ["What Is Minimized SQL?"](#) (p. 330).

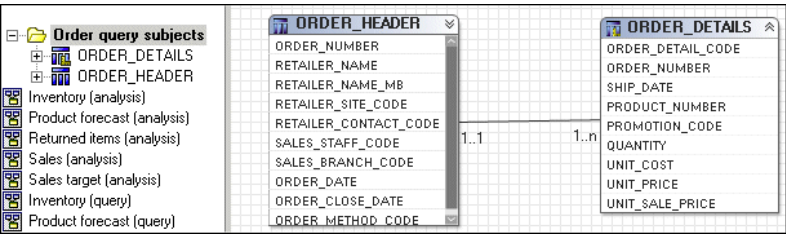
- The model query subject becomes a stand-alone object.

This means underlying relationships are no longer applied, except those between the objects that are referenced. It may be necessary to create additional relationships that were previously inferred from the metadata of the underlying query subjects.

- When a determinant is created on a model query subject, the determinant is ignored unless a relationship is also created.

Here is an example of a relationship on a model query subject that purposely overrides the **Minimized SQL** setting and simplifies the model. In this example, Order Header and Order Details are combined so that they behave as a single fact. They are placed in their own folder and all relationships to

them are deleted except the relationship between Order Header and Order Details. This is the only relationship that will matter after a model query subject is created and relationships attached to it.



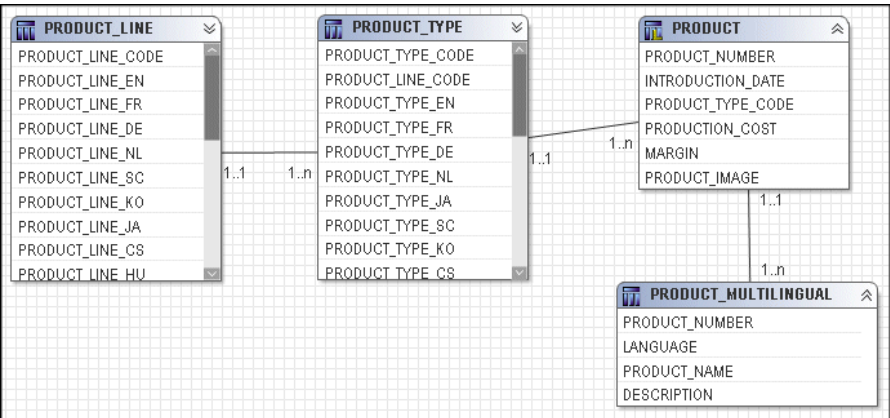
To decide where to specify relationships and determinants in the model, you must understand the impact of minimized SQL to your application.

For more information about relationships, determinants, and minimized SQL, see the **Model Advisor** topics in the IBM® Cognos® Framework Manager *User Guide*.

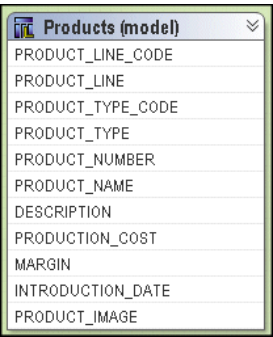
What Is Minimized SQL?

When you use minimized SQL, the generated SQL contains only the minimal set of tables and joins needed to obtain values for the selected query items.

To see an example of what minimized SQL means, you can use the following Product tables. Four query subjects, Product Line, Product Type, Product, and Product Multilingual all join to each other.



They can be combined in a model query subject.



If you test the Products model query subject as a whole, you see that four tables are referenced in the `from` clause of the query.

```

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.DESCRPTION 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_82..gos1.PRODUCT_LINE PRODUCT_LINE,
  gos1_82..gos1.PRODUCT_TYPE PRODUCT_TYPE,
  gos1_82..gos1.PRODUCT PRODUCT,
  gos1_82..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 Product name, you see that the resulting query uses only Product Multilingual, which is the table that was required. This is the effect of minimized SQL.

```

select
  PRODUCT_MULTILINGUAL.PRODUCT_NAME as Product_Name
from
  gos1_82..gos1.PRODUCT_MULTILINGUAL PRODUCT_MULTILINGUAL
where
  (PRODUCT_MULTILINGUAL."LANGUAGE" = N'EN')

```

Example: When Minimized SQL Is Important

If you are modeling a normalized data source, you may be more concerned about minimized SQL because it will reduce the number of tables used in some requests and perform better. In this case, it would be best to create relationships and determinants between the data source query subjects and then create model query subjects that do not have relationships.

There is a common misconception that if you do not have relationships between objects, you cannot create star schema groups. This is not the case. Select the model query subjects to include in the group and use the **Star Schema Grouping** wizard. Or you can create shortcuts and move them to a new namespace. There is no need to have shortcuts to the relationships; this feature is purely visual in the diagram. The effect on query generation and presentation in the studios is the same.

Example: When Minimized SQL Is Not as Important as Predictable Queries

There may be some elements in a data source that you need to encapsulate to ensure that they behave as if they were one data object. An example might be a security table that must always be joined to a fact. In the Great Outdoors Sales model, Order Header and Order Details are a set of tables that together represent a fact and you would always want them to be queried together. For an example, see ["Where Should You Create Relationships and Determinants?"](#) (p. 329).

What Is Metadata Caching?

IBM® Cognos® Framework Manager stores the metadata that is imported from the data source. However depending on governor settings and certain actions you take in the model, this metadata might not be used when preparing a query. If you enable the **Allow enhanced model portability at run time** governor, Framework Manager always queries the data source for information about the metadata before preparing a query. If you have not enabled this governor, in most cases Framework Manager accesses the metadata that has been stored in the model instead of querying the data source. The main exceptions are:

- The SQL in a data source query subject has been modified. This includes the use of macros.
- A calculation or filter has been added to a data source query subject.

Note: The generated metadata queries are well supported by most relational database management system vendors and should not have a noticeable impact on most reporting applications.

Query Subjects vs. Dimensions

Query subjects and dimensions serve separate purposes. The query subject is used to generate relational queries and may be created using star schema rules, while the dimension is used for dimensional modeling of relational sources, which introduces OLAP behavior. Because query subjects are the foundation of dimensions, a key success criterion for any dimensional model is a sound relational model.

A dimensional model is required only if you want to use IBM Cognos Analysis Studio, to enable drilling up and down in reports, or to access member functions in the studios. For many applications, there is no need for OLAP functionality. For example, your application is primarily for ad hoc query or reporting with no requirement for drilling up and down. Or you are maintaining an IBM Cognos ReportNet model. In these cases, you may choose to publish packages based on query subjects alone.

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 are planning to use your query subjects as the foundation for dimensions, you should consider the structure of the hierarchies you expect to create and ensure that you have created determinants that will support correct results when aggregating. Ensure that you have the following:

- The query subject should have a determinant specified for each level of the hierarchy in the regular dimension.
- The determinants should be specified in the same order as the levels in the regular dimension.
- If you expect to have multiple hierarchies that aggregate differently, you may need to consider creating an additional query subject with different determinants as the source for the other hierarchy.

By creating a complete relational model that delivers correct results and good performance, you will have a strong foundation for developing a dimensional model. In addition, by ensuring that a layer of model objects, either query subjects or dimensions, exists between the data source and the objects exposed to the studios, you are better able to shield your users from change.

Model Objects vs. Shortcuts

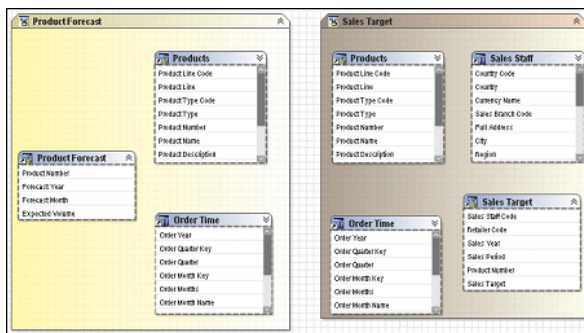
The key difference between model objects and shortcuts is that model objects give you the freedom to include or exclude items and to rename them. You may choose to use model objects instead of shortcuts if you need to limit the query items included or to change the names of items.

Shortcuts are less flexible from a presentation perspective than model objects, but they require much less maintenance because they are automatically updated when the target object is updated. If maintenance is a key concern and there is no need to customize the appearance of the query subject, use shortcuts.

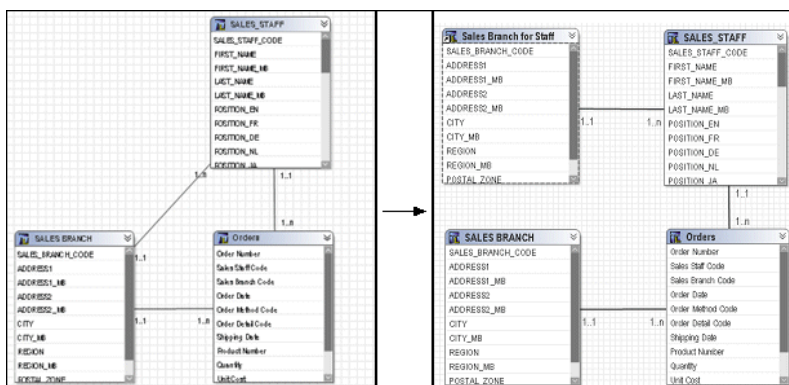
IBM® Cognos® Framework Manager has two types of shortcuts:

- regular shortcuts, which are a simple reference to the target object.
- alias shortcuts, which behave as if they were a copy of the original object with completely independent behavior. Alias shortcuts are available only for query subjects and dimensions.

Regular shortcuts are typically used as conformed dimensions with star schema groups, creating multiple references with the exact same name and appearance in multiple places. In the example below, the shortcuts created for Products and Order Time behave as references. If a query is written that brings Products from both Product Forecast and Sales Target, the query uses the definition of Products based on the original and this definition appears only once in the query.



Alias shortcuts are typically used in role-playing dimensions or shared tables. Because there is already an example in this document for role-playing dimensions, we will look at the case of shared tables. In this example, Sales Staff and Sales Branch can be treated as different hierarchies. From our knowledge of the data, we know that because staff can move between branches, we need to be able to report orders against Sales Branch and Sales Staff independently as well as together. To achieve this, we need to create an alias to Sales Branch that can be used as a level in the Sales Staff hierarchy.



With the new alias shortcut in place, it is possible to create queries that require orders by sales branch and orders by sales staff with their current branch information simultaneously.

When you open a model from a previous release, the **Shortcut Processing** governor is set to **Automatic**. When **Automatic** is used, shortcuts work the same as in previous releases, that is, a shortcut that exists in the same folder as its target behaves as an alias, or independent instance, whereas a shortcut existing elsewhere in the model behaves as a reference to the original. To take advantage of the **Treat As** property, it is recommended that you verify the model and, when repairing, change the governor to **Explicit**. The repair operation changes all shortcuts to the correct value from the **Treat As** property based on the rules followed by the **Automatic** setting, this means that there should be no change in behavior of your model unless you choose to make one or more changes to the **Treat As** properties of your shortcuts.

When you create a new model, the **Shortcut Processing** governor is always set to **Explicit**.

When the governor is set to **Explicit**, the shortcut behavior is taken from the **Treat As** property and you have complete control over how shortcuts behave without being concerned about where in the model they are located.

Folders vs. Namespaces

The most important thing to know about namespaces is that once you have begun authoring reports, any changes you make to the names of published namespaces will impact your IBM Cognos content. This is because changing the name of the namespace changes the IDs of the objects published in the metadata. Because the namespace is used as part of the object ID in IBM Cognos Framework Manager, each namespace must have a unique name in the model. Each object in a namespace must also have a unique name. Part of the strategy of star schema groups is placing shortcuts into a separate namespace, which automatically creates a unique ID for each object in the namespace. For relational databases, this allows us to use the same name for shortcuts to conformed dimensions in different star schema groups.

The next time you try to run a query, report, or analysis against the updated model, you get an error. If you need to rename the namespace that you have published, use **Analyze Publish Impact** to determine which reports are impacted.

Folders are much simpler than namespaces. They are purely for organizational purposes and do not impact object IDs or your content. You can create folders to organize objects by subject or functional area. This makes it easier for you to locate metadata, particularly in large projects.

The main drawback of folders is that they require unique names for all query subjects, dimensions, and shortcuts. Therefore, they are not ideal for containing shared objects.

Setting the Order of Operations for Model Calculations

In some cases, usually for ratio-related calculations, it is useful to perform the aggregation on the calculation terms prior to the mathematical operation.

For example, the following Order details fact contains information about each order:

Order details	
Order detail code	
Order number	
Ship date	
Product number	
Quantity	
Unit cost	
Unit price	
Unit sale price	
Revenue	→ Regular Aggregate: Sum
Product cost	→ Regular Aggregate: Sum
Gross profit	
Margin	→ Regular Aggregate: Sum

Margin is a calculation that computes the ratio of profit:

$$\text{Margin} = (\text{Revenue} - \text{Product cost}) / \text{Revenue}$$

If we run a query to show Revenue, Product cost, and Margin for each product using the Order details fact, we get the following results:

Product number	Revenue	Product cost	Margin
1	\$23,057,141	\$11,292,005	61038%
2	\$11,333,518	\$6,607,904	49606%

Notice that the value for Margin seems to be wrong. This is because of the order of operations used in computing Margin. Margin is computed as:

$$\text{Margin} = \text{sum}((\text{Revenue} - \text{Product cost}) / \text{Revenue})$$

The aggregation took place after the mathematical operation and, in this case, it produces undesired results.

To produce the desired values for Margin, we need to aggregate before the mathematical operation:

$$\text{Margin} = (\text{sum}(\text{Revenue}) - \text{sum}(\text{Product cost})) / \text{sum}(\text{Revenue})$$

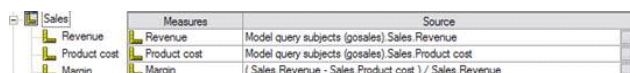
This produces the following results:

Product number	Revenue	Product cost	Margin
1	\$23,057,141	\$11,292,005	51.03%
2	\$11,333,518	\$6,607,904	41.70%

You can accomplish this in IBM® Cognos® Framework Manager by creating a stand-alone calculation for Margin and setting its **Regular Aggregate** property to **Calculated**. Each query item in the calculation's expression is aggregated as specified in its **Regular Aggregate** property. The **Regular Aggregate** properties for Revenue and Product cost are set to **Sum** and thus, when computing the calculation, sum is used to aggregate those terms.

Note: The calculated aggregation type is not supported for calculations that are embedded within query subjects. It is supported only for stand-alone calculations and for calculations that are embedded within measure dimensions and are based on measures from the same measure dimension.

For example, consider the Margin calculation that is embedded in the Sales measure dimension:



Measures	Source
Revenue	Model query subjects (gosales) Sales Revenue
Product cost	Model query subjects (gosales) Sales Product cost
Margin	(Sales Revenue - Sales Product cost) / Sales Revenue

In this example, Margin is based on the measures Product cost and Revenue that are within the same measure dimension, Sales. If the **Regular Aggregate** property for Margin is set to **Calculated**, it is rolled up as:

```
Margin = sum(Revenue - Product cost) / sum(Revenue)
```

If Margin is based on the source query items of the measures Product cost and Revenue (Sales (model).Product cost, Sales (model).Revenue), the calculated aggregation is not supported and the aggregation behaves as automatic. In this case, Margin is rolled up as:

```
Margin = sum(Revenue - Product cost) / Revenue
```

For more information about modifying how query items are aggregated, see the IBM® Cognos® Framework Manager *User Guide*.

Impact of Model Size

The size of your model may affect the efficiency of the Framework Manager application. Very large models will cause extended processing times and, in extreme cases, out-of-memory conditions. Actions such as Analyze Publish Impact, Find Report Dependencies, Publish Packages and Run Model Advisor perform optimally on models under 50 megabytes.

Dimensional Modeling Concepts

Regular and measure dimensions are used to enable an OLAP presentation of metadata, drilling up and down, and a variety of OLAP functions. You must use star schema groups (one fact with multiple dimensions) if you want to use IBM Cognos Analysis Studio with a relational data source.

When building your model, it is recommended that model regular dimensions and model measure dimensions be created based on a relational model in which star schema concepts have been applied.

While you can convert data source query subjects to data source dimensions, data source dimensions have limited functionality in comparison to query subjects or model dimensions, and they are not recommended for general use.

Regular Dimensions

Regular dimensions represent descriptive data that provides context for data modeled in measure dimensions. A regular dimension is broken into groups of information called levels. In turn, the various levels can be organized into hierarchies. For example, a product dimension can contain the levels Product Line, Product Type, and Product organized in a single hierarchy called Product. Another example is a time dimension that has the levels Year, Quarter, Month, Week, and Day, organized into two hierarchies. The one hierarchy YQMD contains the levels Year, Quarter, Month, and Day, and the other hierarchy YWD contains the levels Year, Week, and Day.

The simplest definition of a level consists of a business key and a caption, each of these referring to one query item. An instance (or row) of a level is defined as a member of that level. It is identified by a member unique name, which contains the values of the business keys of the current and higher levels. For example, [gosales].[Products].[ProductsOrg].[Product]->[All Products].

[1] . [1] . [2] identifies a member that is on the fourth level, `Product`, of the hierarchy `ProductsOrg` of the dimension `[Products]` that is in the namespace `[gosales]`. The caption for this product is `TrailChef Canteen`, which is the name shown in the metadata tree and on the report.

The level can be defined as unique if the business key of the level is sufficient to identify each set of data for a level. In the Great Outdoors Sales model, the members of the `Product` level do not require the definition of `Product` type because there are no product numbers assigned to many different product types. A level that is not defined as unique is similar to a determinant that uses multiple-part keys because keys from higher levels of granularity are required (p. 323). If members within ancestor members are not unique but the level is defined as unique, data for the non-unique members is reported as a single member. For example, if `City` is defined as unique and identified by name, data for `London, England` and `London, Canada` will be combined.

A regular dimension may also have multiple hierarchies; however, you can use only one hierarchy at a time in a query. For example, you cannot use one hierarchy in the rows of a crosstab report and another hierarchy from the same dimension in the columns. If you need both hierarchies in the same report, you must create two dimensions, one for each hierarchy.

Measure Dimensions

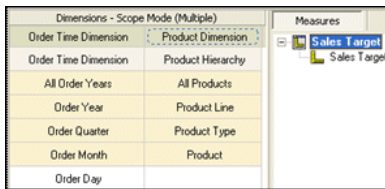
Measure dimensions represent the quantitative data described by regular dimensions. Known by many terms in various OLAP products, a measure dimension is simply the object that contains the fact data. Measure dimensions differ from fact query subjects because they do not include the foreign keys used to join a fact query subject to a dimensional query subject. This is because the measure dimension is not meant to be joined as if it were a relational data object. For query generation purposes, a measure dimension derives its relationship to a regular dimension through the underlying query subjects. Similarly the relationship to other measure dimensions is through regular dimensions that are based on query subjects built to behave as conformed dimensions. To enable multiple-fact, multiple-grain querying, you must have query subjects and determinants created appropriately before you build regular dimensions and measure dimensions.

Scope Relationships

Scope relationships exist only between measure dimensions and regular dimensions to define the level at which the measures are available for reporting. They are not the same as joins and do not impact the `Where` clause. There are no conditions or criteria set in a scope relationship to govern how a query is formed, it specifies only if a dimension can be queried with a specified fact. The absence of a scope relationship may result in an error at runtime or cause fact data to be rolled up at a high level than expected given the other items in the report.

If you set the scope relationship for the measure dimension, the same settings apply to all measures in the measure dimension. If data is reported at a different level for the measures in the measure dimension, you can set scope on a measure. You can specify the lowest level that the data can be reported on.

In this example, the `Sales Target` measure dimension has only one measure that is in scope to the `Order Month` level on the `Order Time Dimension` and to the `Product` level of the `Product Dimension`. This means that if your users try to drill beyond the month level, they will see repeated data.



Building the Relational Model

Dimensional modeling of relational data sources is available in IBM® Cognos® Framework Manager, however it depends on the existence of a sound relational model. IBM Cognos ReportNet provided some dimensional capabilities to enable multiple-fact querying and to prevent double-counting. Subsequent to IBM Cognos ReportNet, the IBM Cognos software has features designed explicitly for dimensional representation of metadata and OLAP capability with relational data sources. The concepts applied to relational modeling in IBM Cognos ReportNet have been preserved with a few changes that are documented in the Framework Manager *User Guide*.

When you create a new Framework Manager model, you will follow a common set of steps to define query generation even if you do not intend to use dimensional modeling capabilities. You must dimensionally model a relational data source when you want to use it in IBM Cognos Analysis Studio, to enable drilling up and down in reports, or to access member functions in the studios.

When you build the relational model, we recommend that you do the following:

- ☐ Define the relational modeling foundation ([p. 338](#)).
- ☐ Define the dimensional representation of the model ([p. 345](#)), if it is required.
- ☐ Organize the model ([p. 348](#)).

Defining the Relational Modeling Foundation

A model is the set of related objects required for one or more related reporting applications. A sound relational model is the foundation for a dimensional model.

We recommend that you do the following when you define the relational modeling foundation:

- ☐ Import the metadata. For information about importing, see the IBM® Cognos® Framework Manager *User Guide*.
- ☐ Verify the imported metadata ([p. 339](#)).
- ☐ Resolve ambiguous relationships ([p. 339](#)).
- ☐ Simplify the relational model using star schema concepts by analyzing cardinality for facts and dimensions and by deciding where to put relationships and determinants ([p. 329](#)).
- ☐ Add data security, if required. For information about data security, see the Framework Manager *User Guide*.

Then you can define the dimensional representation of the model ([p. 345](#)) if it is required, and organize the model for presentation ([p. 348](#)).

Verifying Imported Metadata

After importing metadata, you must check the imported metadata in these areas:

- relationships and cardinality
- determinants
- the **Usage** property for query items
- the **Regular Aggregate** property for query items

Relationships and cardinality are discussed here. For information on the **Usage** and **Regular Aggregate** properties, see the Framework Manager *User Guide*.

Analyzing the Cardinality for Facts and Dimensions

The cardinality of a relationship defines the number of rows of one table that is related to the rows of another table based on a particular set (or join) of keys. Cardinality is used by IBM Cognos software to infer which query subjects behave as facts or dimensions. The result is that IBM Cognos software can automatically resolve a common form of loop join that is caused by star schema data when you have multiple fact tables joined to a common set of dimension tables.

To ensure predictable queries, it is important to understand how cardinality is used and to correctly apply it in your model. It is recommended that you examine the underlying data source schema and address areas where cardinality incorrectly identifies facts or dimensions that could cause unpredictable query results. The **Model Advisor** feature in Framework Manager can be used to help you understand how the cardinality is interpreted.

For more information, see ["Cardinality" \(p. 320\)](#).

Resolving Ambiguous Relationships

Ambiguous relationships occur when the data represented by a query subject or dimension can be viewed in more than one context or role, or can be joined in more than one way. The most common ambiguous relationships are:

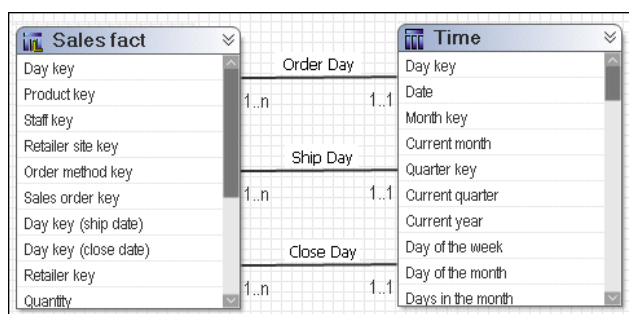
- role-playing dimensions [\(p. 339\)](#)
- loop joins [\(p. 341\)](#)
- reflexive and recursive relationships [\(p. 342\)](#)

You can use the **Model Advisor** to highlight relationships that may cause issues for query generation and resolve them in one of the ways described below. Note that there are other ways to resolve issues than the ones discussed here. The main goal is to enable clear query paths.

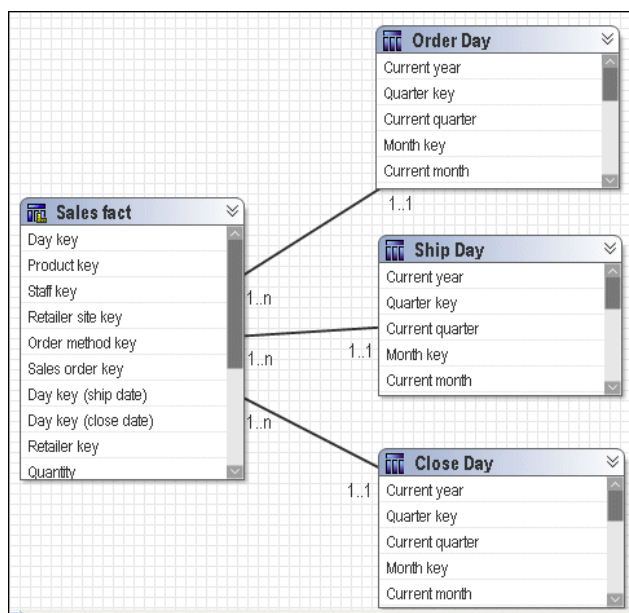
Role-Playing Dimensions

A table with multiple valid relationships between itself and another table is known as a role-playing dimension. This is most commonly seen in dimensions such as Time and Customer.

For example, the Sales fact has multiple relationships to the Time query subject on the keys Order Day, Ship Day, and Close Day.



Remove the relationships for the imported objects, fact query subjects, and role-playing dimensional query subjects. Create a model query subject for each role. Consider excluding unneeded query items to reduce the length of the metadata tree displayed to your users. Ensure that a single appropriate relationship exists between each model query subject and the fact query subject. **Note:** This will override the **Minimized SQL** setting but given a single table representation of the Time dimension, it is not considered to be problematic in this case.

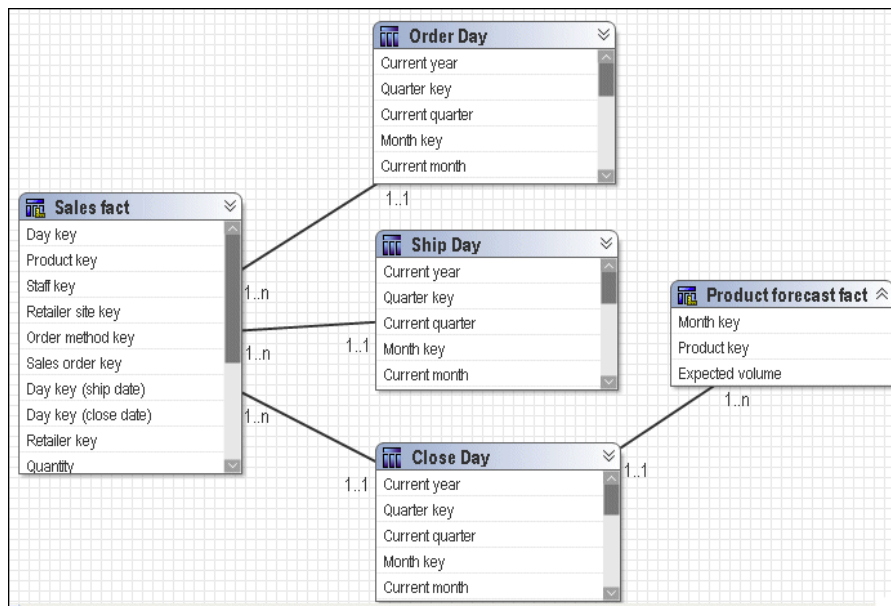


Decide how to use these roles with other facts that do not share the same concepts. For example, Product forecast fact has only one time key. You need to know your data and business to determine if all or any of the roles created for Time are applicable to Product forecast fact.

In this example, you can do one of the following:

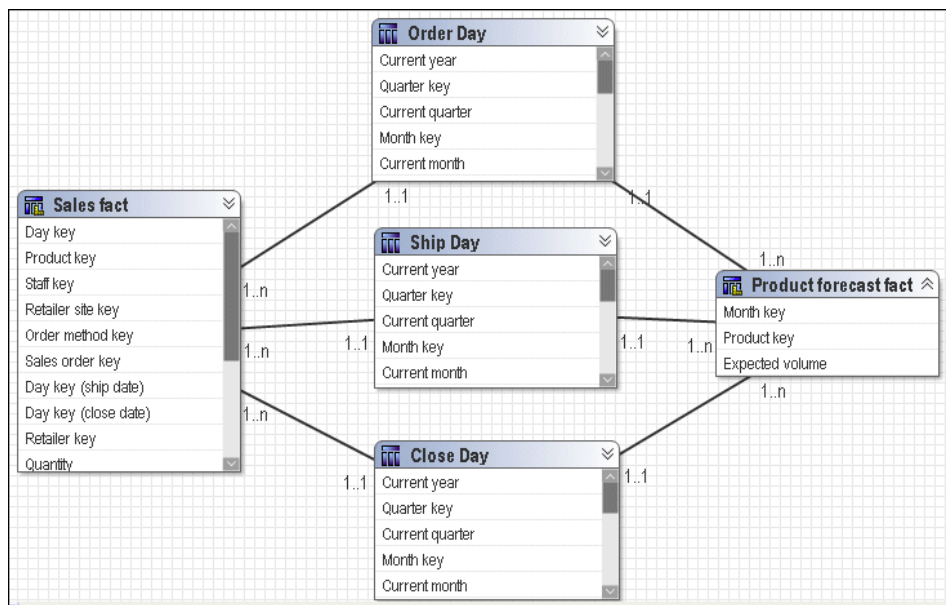
- Create an additional query subject to be the conformed time dimension and name it clearly as a conformed dimension.

Pick the most common role that you will use. You can then ensure that this version is joined to all facts requiring it. In this example, Close Day has been chosen.



- You can treat Ship Day, Order Day, and Close Day as interchangeable time query subjects with Product forecast fact.

In this case, you must create joins between each of the role-playing dimensions and Product forecast fact. You can use only one time dimension at a time when querying the Product forecast fact or your report may contain no data. For example, Month_key=Ship Month Key (200401) and Month key=Close Month Key (200312).



If modeling dimensionally, use each model query subject as the source for a regular dimension, and name the dimension and hierarchies appropriately. Ensure that there is a corresponding scope relationship specific to each role.

Loop Joins

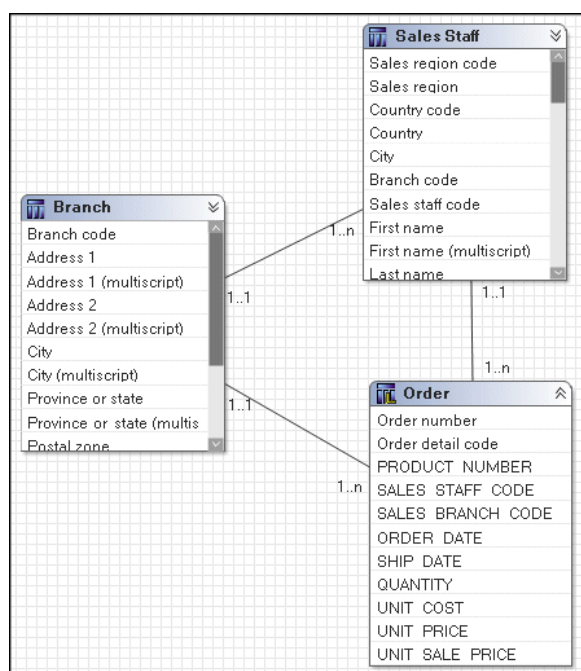
Loop joins in the model are typically a source of unpredictable behavior. This does not include star schema loop joins.

Note: When cardinality clearly identifies facts and dimensions, IBM Cognos software can automatically resolve loop joins that are caused by star schema data when you have multiple fact tables joined to a common set of dimension tables.

In the case of loop joins, ambiguously defined query subjects are the primary sign of problems. When query subjects are ambiguously defined and are part of a loop join, the joins used in a given query are decided based on a number of factors, such as the location of relationships, the number of segments in join paths, and, if all else is equal, the alphabetically first join path. This creates confusion for your users and we recommend that you model to clearly identify the join paths.

Sales Staff and Branch provide a good example of a loop join with ambiguously defined query subjects.

In this example, it is possible to join Branch directly to Order or through Sales Staff to Order. The main problem is that when Branch and Order are together, you get a different result than when the join path is Branch to Sales Staff to Order. This is because employees can move between branches so employees who moved during the year are rolled up to their current branch even if many of the sales they made are attributable to their previous branch. Because of the way this is modeled, there is no guarantee which join path will be chosen and it is likely to vary depending on which items are selected in the query.



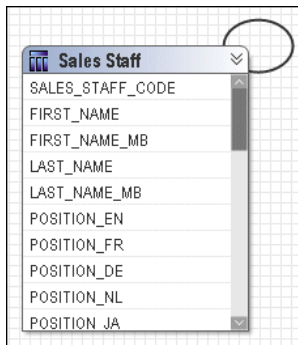
Reflexive and Recursive Relationships

Reflexive and recursive relationships imply two or more levels of granularity. IBM® Cognos® Framework Manager imports reflexive relationships but does not use them when executing queries. Reflexive relationships, which are self-joins, are shown in the model for the purpose of representation only.

To create a functioning reflexive relationship, you can either create an alias shortcut, a copy of the data source query subject, or a model query subject. You then create a relationship between the original query subject and the new one. Using a model query subject tends to be the better option for flexibility because you can specify which query items are included in the query subject. Shortcuts

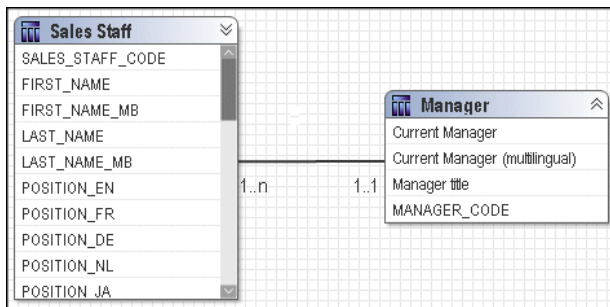
are the better solution from a maintenance perspective. For more information, see ["Model Objects vs. Shortcuts" \(p. 333\)](#).

For example, the Sales Staff query subject has a recursive relationship between Sales_Staff_Code and Manager_Code.



Create a model query subject to represent Manager. Create a relationship with a 1..1 to 1..n between Manager and Sales Staff. Then merge into a new model query subject.

For a simple two-level structure using a model query subject for Manager that is based on Sales Staff, the model looks like this:



For a recursive, balanced hierarchy, repeat this for each additional level in the hierarchy.

For a deep recursive or unbalanced hierarchy, we recommend that the hierarchy be flattened in the data source and that you model the flattened hierarchy in one regular dimension.

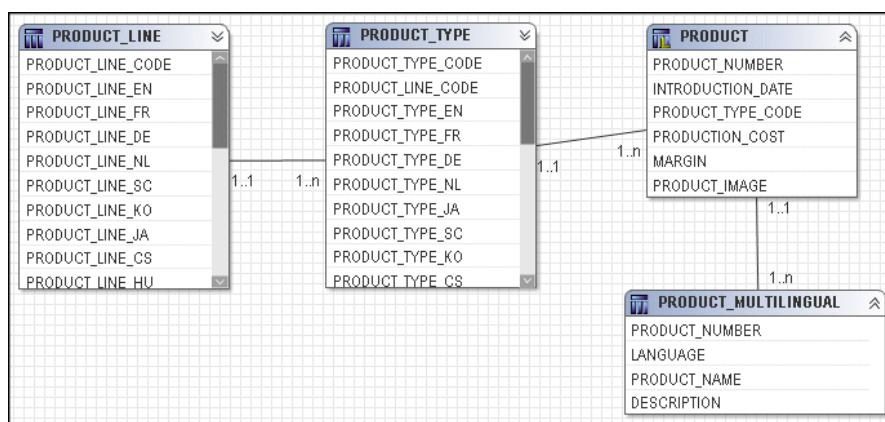
Simplifying the Relational Model

You can simplify the model by applying star schema concepts to the dimensional data and the fact data.

Modeling Query Subjects That Represent Descriptive Data

IBM Cognos dimensional modeling requires that you apply star schema principles to the logical layers of the model.

Normalized or snowflaked data sources often have several tables that describe a single business concept. For example, a normalized representation of Product may include four tables related by 1..n relationships. Each Product Line has one or more Product Types. Each Product Type has one or more Products. Products have names and descriptions in multiple languages so they exist in the Product Multilingual lookup table.

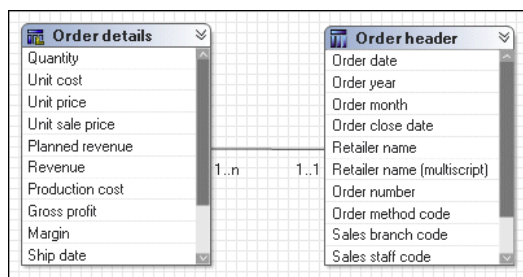


One way to simplify the model is to create one model query subject for each descriptive business concept. Your users may not know the relationship between the individual query subjects so it is helpful to group them together; in addition, having to expand each model object and select a query item requires more effort.

The next step for analysis is to create a regular dimension with a level for each query subject.

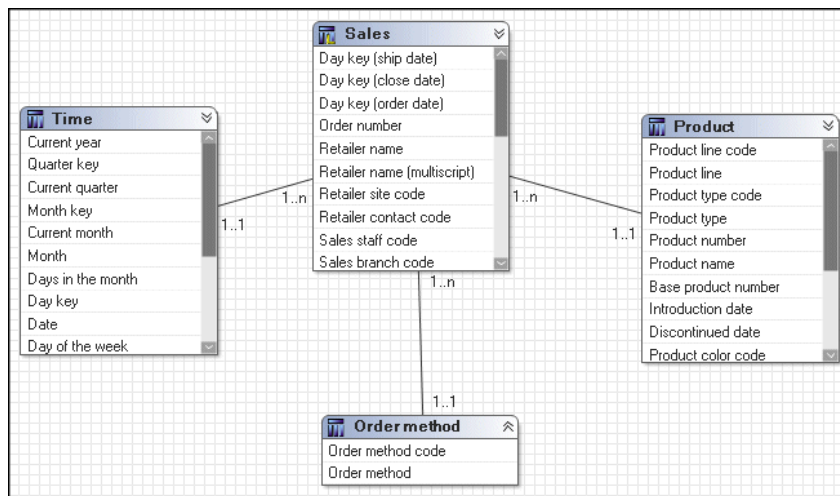
Modeling Fact Data

Data sources often have master-detail tables that contain facts. For example, when the Order header and Order details tables are used to insert and update data, the master-detail structure is beneficial. When these tables are used for reporting and analysis, you may choose to combine them into one logical business concept to simplify the model. Or you may choose to insert a dimension between them, such as Returned Items. Which solution you choose depends on your requirements.



To simplify the model in this example, apply star schema concepts to create one model query subject that combines the foreign keys of both Order header and Order details and includes all measures at the Order details level. This query subject should be joined to the same query subjects that Order header and Order details were joined to. You may choose to remove the original relationships from the two data source query subjects except for the relationship that defines the join between them. For a discussion of the pros and cons of creating relationships to model query subjects, see the examples in ["What Is Minimized SQL?"](#) (p. 330).

In the example below, Order header and Order details have been combined into a new model query subject named Sales. This query subject has been joined to Product, Time, and Order method.



The next step for analysis is to create a measure dimension based on the model query subject.

Defining the Dimensional Representation of the Model

Dimensional modeling of relational data sources is a capability made available by IBM® Cognos® Framework Manager. You can model dimensions with hierarchies and levels and have facts with multiple measures. You can then relate the dimensions to the measures by setting scope in the model.

You must dimensionally model a relational data source when you want to use it in IBM Cognos Analysis Studio, enable drilling up and down in reports, or access member functions in the studios.

We recommend that you use the relational model as the foundation layer (p. 338) and then do the following when you define the dimensional representation of the model:

- ☐ Create regular dimensions (p. 345).
- ☐ Model dimensions with multiple hierarchies (p. 346).
- ☐ Create measure dimensions (p. 347).
- ☐ Create scope relationships (p. 347).

Then you can organize the model for presentation (p. 348).

Creating Regular Dimensions

A regular dimension contains descriptive and business key information and organizes the information in a hierarchy, from the highest level of granularity to the lowest. It usually has multiple levels and each level requires a key and a caption. If you do not have a single key for your level, it is recommended that you create one in a calculation.

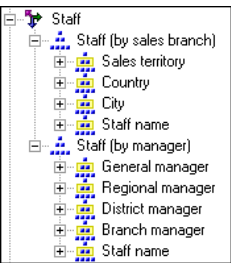
Model regular dimensions are based on data source or model query subjects that are already defined in the model. You must define a business key and a string type caption for each level. When you verify the model, the absence of business keys and caption information is detected. Instead of joining model regular dimensions to measure dimensions, create joins on the underlying query subjects and create a scope relationship between the regular dimension and the measure dimension.

Modeling Dimensions with Multiple Hierarchies

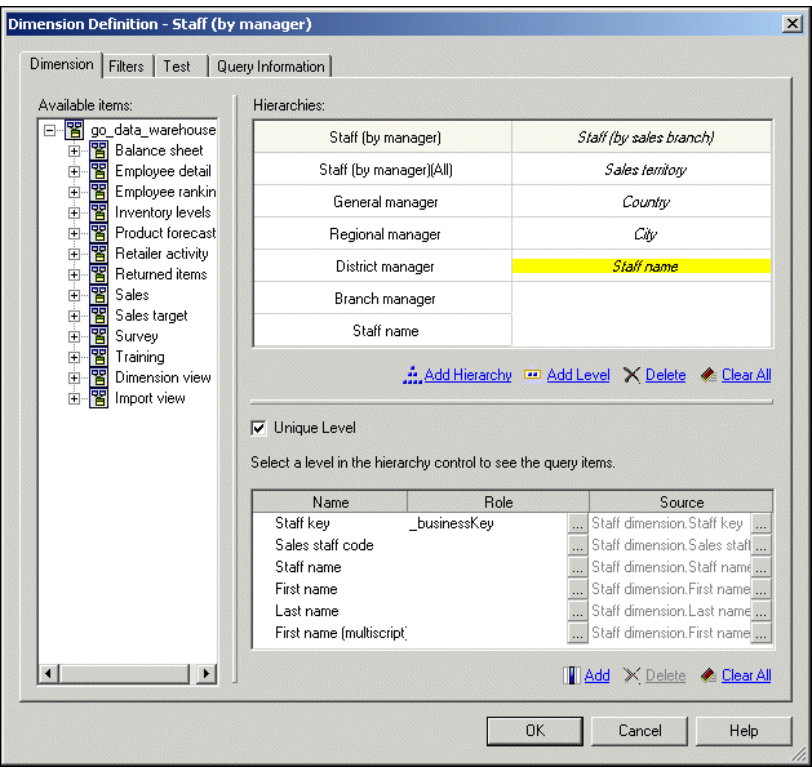
Multiple hierarchies occur when different structural views can be applied to the same data. Depending on the nature of the hierarchies and the required reports, you may need to evaluate the modeling technique applied to a particular case.

For example, sales staff can be viewed by manager or geography. In the IBM Cognos studios, these hierarchies are separate but interchangeable logical structures, which are bound to the same underlying query.

Here is sales staff as a single dimension with two hierarchies:



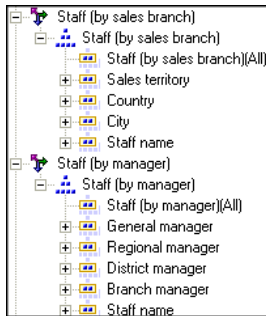
The hierarchies are defined in Framework Manager as follows.



You can specify multiple hierarchies on regular dimensions in Framework Manager. Multiple hierarchies for a regular dimension behave as views of the same query. However, you can use only one hierarchy at a time in a query. For example, you cannot use one hierarchy in the rows of a crosstab report and another hierarchy from the same dimension in the columns. If you need both hierarchies in the same report, you must create two dimensions, one for each hierarchy. In cases where you have multiple hierarchies with significantly different levels or aggregation, you may choose to model so that a separate query subject with appropriate determinants exists as the foun-

ation for that hierarchy. The only requirement is that any query subject used as the basis for a hierarchy must have a join defined to the query subject that provides the fact data.

Here are separate dimensions for each hierarchy.



Use this approach if dramatically different sets of columns are relevant for each hierarchy and it is more intuitive for your users to model the hierarchies as separate dimensions with separate and simpler queries.

Creating Measure Dimensions

A measure dimension is a collection of facts. You can create a measure dimension for one or more query subjects that have a valid relationship between them.

Model measure dimensions should be composed of only quantitative items. Because, by design, model measure dimensions do not contain keys on which to join, it is not possible to create joins to model measure dimensions. Instead of joining model measure dimensions to regular dimensions, create joins on the underlying query subjects. Then either manually create a scope relationship between them or detect scope if both dimensions are in the same namespace.

Create Scope Relationships

Scope relationships exist only between measure dimensions and regular dimensions to define the level at which the measures are available for reporting. They are not the same as joins and do not impact the `Where` clause. There are no conditions or criteria set in a scope relationship to govern how a query is formed, it specifies only if a dimension can be queried with a specified fact. The absence of a scope relationship results in an error at runtime.

If you set the scope relationship for the measure dimension, the same settings apply to all measures in the measure dimension. If data is reported at a different level for the measures in the measure dimension, you can set scope on a measure. You can specify the lowest level that the data can be reported on.

When you create a measure dimension, IBM® Cognos® Framework Manager creates a scope relationship between the measure dimension and each existing regular dimension. Framework Manager looks for a join path between the measure dimension and the regular dimensions, starting with the lowest level of detail. If there are many join paths available, the scope relationship that Framework Manager creates may not be the one that you intended. In this case, you must edit the scope relationship.

Organizing the Model

After working in the relational modeling foundation ([p. 338](#)) and creating a dimensional representation ([p. 345](#)), we recommend that you do the following to organize the model:

- ❑ Keep the metadata from the data source in a separate namespace or folder.
- ❑ Create one or more optional namespaces or folders for resolving complexities that affect querying using query subjects.

To use IBM® Cognos® Analysis Studio, there must be a namespace or folder in the model that represents the metadata with dimensional objects.

- ❑ Create one or more namespaces or folders for the augmented business view of the metadata that contains shortcuts to dimensions or query subjects.

Use business concepts to model the business view. One model can contain many business views, each suited to a different user group. You publish the business views.

- ❑ Create the star schema groups ([p. 348](#)).
- ❑ Apply object security, if required.
- ❑ Create packages and publish the metadata.

For information about the topics not covered here, see the Framework Manager *User Guide*.

Star Schema Groups

The concept of the conformed dimension is not isolated to dimensional modeling, it applies equally to query subjects.

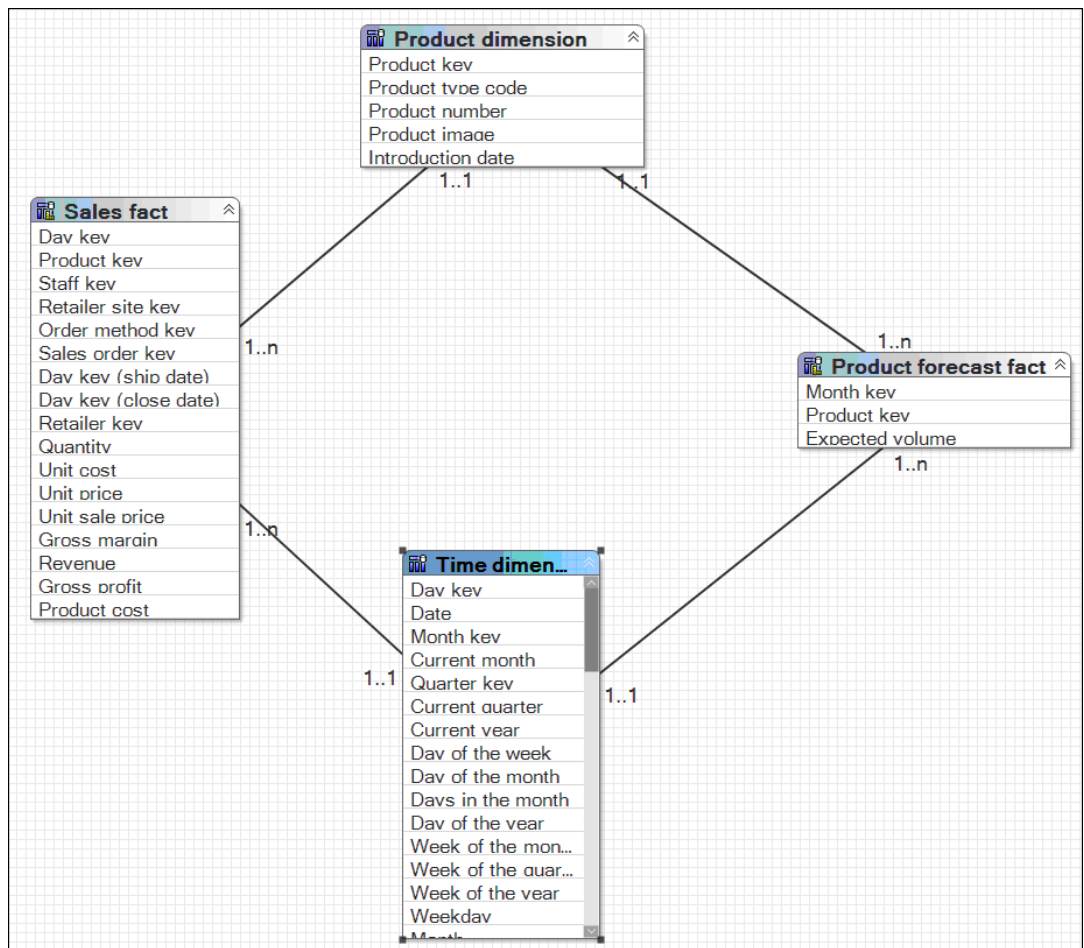
Use the **Star Schema Grouping** wizard to quickly create groups of shortcuts that will provide context for your users regarding which objects belong together. This makes the model more intuitive for your users. Star schema groups can also facilitate multiple-fact reporting by allowing the repetition of shared dimensions in different groups. This helps your users to see what different groups have in common and how they can do cross-functional, or multiple-fact, reporting. For more information, see "[Multiple-fact, Multiple-grain Queries](#)" ([p. 326](#)).

Star schema groups also provide context for queries with multiple join paths. By creating star schema groups in the business view of the model, you can clarify which join path to select when many are available. This is particularly useful for fact-less queries.

Multiple Conformed Star Schemas or Fact-less Queries

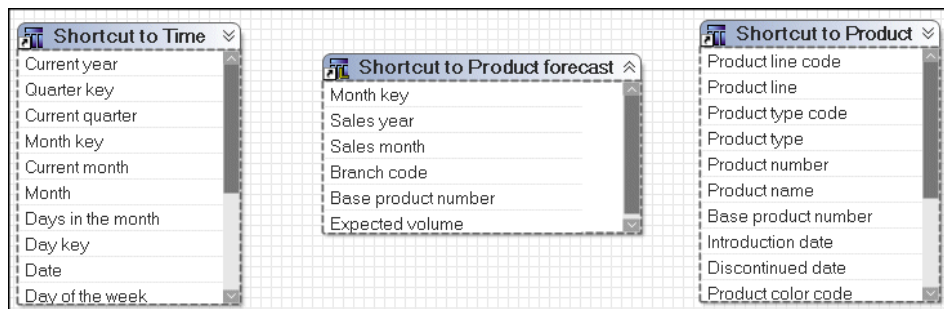
You will likely see dimensional query subjects that are joined to more than one fact query subject. Join ambiguity is an issue when you report using items from multiple dimensions or dimensional query subjects without including any items from the measure dimension or fact query subject. This is called a fact-less query.

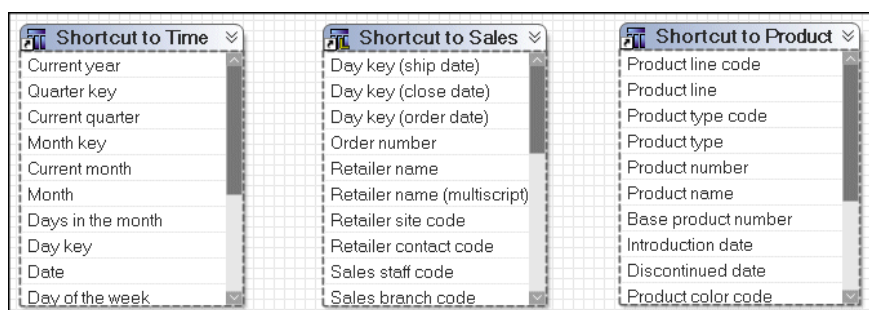
For example, Product and Time dimensions are related to the Product forecast and Sales facts.



Using these relationships, how do you write a report that uses only items from Product and Time? The business question could be which products were forecasted for sale in 2005 or which products were actually sold in 2005. Although this query involves only Product and Time, these dimensions are related through multiple facts. There is no way to guess which business question is being asked. You must set the context for the fact-less query.

In this example, we recommend that you create two namespaces, one containing shortcuts to Product, Time, and Product forecast, and another containing Product, Time, and Sales.





When you do this for all star schemas, you resolve join ambiguity by placing shortcuts to the fact and all dimensions in a single namespace. The shortcuts for conformed dimensions in each namespace are identical and are references to the original object. **Note:** The exact same rule is applied to regular dimensions and measure dimensions.

With a namespace for each star schema, it is now clear to your users which items to use. To create a report on which products were actually sold in 2005, they use Product and Year from the Sales Namespace. The only relationship that is relevant in this context is the relationship between Product, Time, and Sales, and it is used to return the data.

Chapter 10: The SQL Generated by IBM Cognos Software

The SQL generated by IBM® Cognos® software is often misunderstood. This document explains the SQL that results in common situations.

Note: The SQL examples shown in this document were edited for length and are used to highlight specific examples. These examples use the version 8.2 sample model.

To access the IBM Cognos *Guidelines for Modeling Metadata* documentation in a different language, go to *installation_location\c10\webcontent\documentation* and open the folder for the language you want. Then open *ug_best.pdf*.

Understanding Dimensional Queries

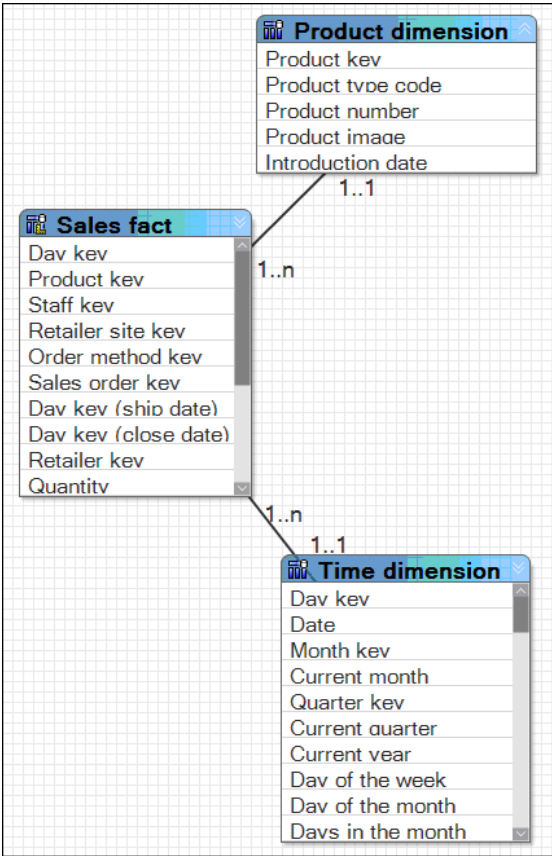
Dimensional queries are designed to enable multiple-fact querying. The basic goals of multiple-fact querying are:

- Preserve data when fact data does not perfectly align across common dimensions, such as when there are more rows in the facts than in the dimensions.
- Prevent double-counting when fact data exists at different levels of granularity by ensuring that each fact is represented in a single query with appropriate grouping. Determinants may need to be created for the underlying query subjects in some cases.

Single Fact Query

A query on a star schema group results in a single fact query.

In this example, Sales is the focus of any query written. The dimensions provide attributes and descriptions to make the data in Sales more meaningful. All relationships between dimensions and the fact are 1-n.



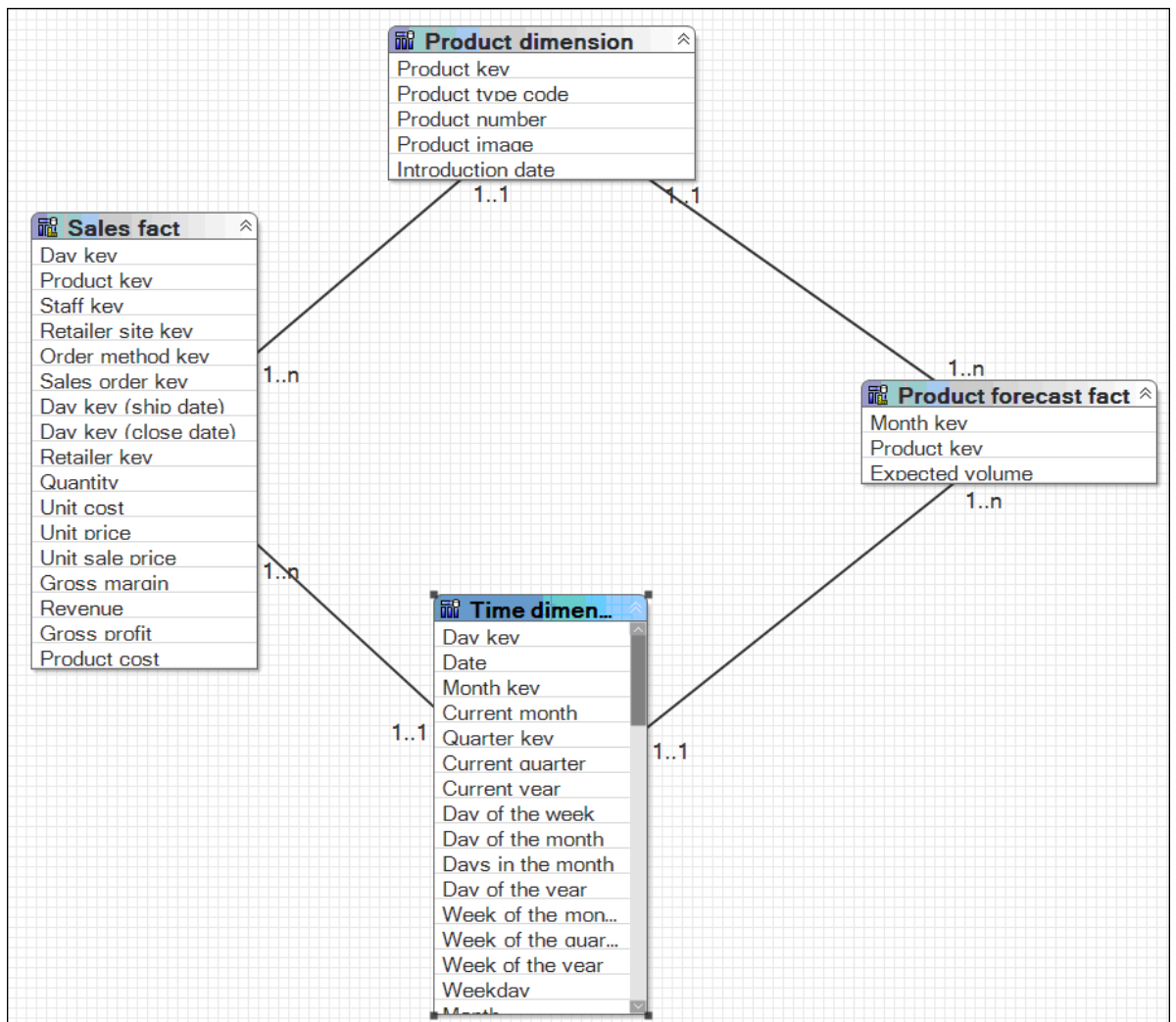
When you filter on the month and product, the result is as follows.

MONTH_NAME	PRODUCT_NAME	QUANTITY
April 2004	Aloe Relief	1,410
April 2004	Course Pro Umbrella	132
February 2004	Aloe Relief	270
February 2006	Aloe Relief	88

Multiple-fact, Multiple-grain Query on Conformed Dimensions

A query on multiple facts and conformed dimensions respects the cardinality between each fact table and its dimensions and writes SQL to return all the rows from each fact table.

For example, Sales and Product Forecast are both facts.



Note that this is a simplified representation and not an example of how this would appear in a model built using IBM® Cognos® modeling recommendations.

The Result

Individual queries on Sales and Product Forecast by Month and Product yield the following results. The data in Sales is actually stored at the day level.

MONTH_NAME	PRODUCT_NAME	EXPECTED_VOLUME
April 2004	Aloe Relief	1,690
April 2004	Course Pro Umbrella	125
February 2004	Aloe Relief	246
February 2004	Course Pro Umbrella	1
February 2006	Aloe Relief	92

A query on Sales and Product Forecast respects the cardinality between each fact table and its dimensions and writes SQL to return all the rows from each fact table. The fact tables are matched on their common keys, month and product, and, where possible, are aggregated to the lowest common level of granularity. In this case, days are rolled up to months. Nulls are often returned for this type of query because a combination of dimensional elements in one fact table may not exist in the other.

MONTH_NAME	PRODUCT_NAME	QUANTITY	EXPECTED_VOLUME
April 2004	Aloe Relief	1,410	1,690
April 2004	Course Pro Umbrella	132	125
February 2004	Aloe Relief	270	246
February 2004	Course Pro Umbrella		1
February 2006	Aloe Relief	88	92

Note that in February 2004, Course Pro Umbrellas were in the forecast but there were no actual sales. The data in Sales and Product Forecast exist at different levels of granularity. The data in Sales is at the day level, and Product Forecast is at the month level.

The SQL

The SQL generated by IBM Cognos software, known as a stitched query, is often misunderstood. A stitched query uses multiple subqueries, one for each star, brought together by a full outer join on the common keys. The goal is to preserve all dimensional members occurring on either side of the query.

The following example was edited for length and is used as an example to capture the main features of stitched queries.

```
select
  coalesce(D2.MONTH_NAME,D3.MONTH_NAME) as MONTH_NAME,
  coalesce(D2.PRODUCT_NAME,D3.PRODUCT_NAME) as PRODUCT_NAME,
  D2.EXPECTED_VOLUME as EXPECTED_VOLUME,
  D3.QUANTITY as QUANTITY
from (select TIME.MONTH_NAME as MONTH_NAME,
  PRODUCT_LOOKUP.PRODUCT_NAME as PRODUCT_NAME,
  XSUM(PRODUCT_FORECAST_FACT.EXPECTED_VOLUME for
  TIME.CURRENT_YEAR,TIME.QUARTER_KEY,TIME.MONTH_KEY,
  PRODUCT.PRODUCT_LINE_CODE, PRODUCT.PRODUCT_TYPE_CODE,
  PRODUCT.PRODUCT_KEY) as EXPECTED_VOLUME
from
  (select TIME.CURRENT_YEAR as CURRENT_YEAR,
  TIME.QUARTER_KEY as QUARTER_KEY,
  TIME.MONTH_KEY as MONTH_KEY,
  XMIN(TIME.MONTH_NAME for TIME.CURRENT_YEAR,
  TIME.QUARTER_KEY,TIME.MONTH_KEY) as MONTH_NAME
  from TIME_DIMENSION TIME
  group by TIME.MONTH_KEY) TIME
  join PRODUCT_FORECAST_FACT PRODUCT_FORECAST_FACT
  on (TIME.MONTH_KEY = PRODUCT_FORECAST_FACT.MONTH_KEY)
  join PRODUCT PRODUCT on (PRODUCT.PRODUCT_KEY =
  PRODUCT_FORECAST_FACT.PRODUCT_KEY)
where
  (PRODUCT.PRODUCT_NAME in ('Aloe Relief','Course Pro
  Umbrella')) and
  (TIME.MONTH_NAME in ('April 2004','February 2004','February
  2006'))
group by
  TIME.MONTH_NAME,
  PRODUCT_LOOKUP.PRODUCT_NAME
) D2
full outer join
(select TIME.MONTH_NAME as MONTH_NAME,
  PRODUCT_LOOKUP.PRODUCT_NAME as PRODUCT_NAME,
  XSUM(SALES_FACT.QUANTITY for TIME.CURRENT_YEAR,
  TIME.QUARTER_KEY, TIME.MONTH_KEY,
  PRODUCT.PRODUCT_LINE_CODE, PRODUCT.PRODUCT_TYPE_CODE,
  PRODUCT.PRODUCT_KEY ) as QUANTITY
from
```

```

select TIME.DAY_KEY, TIME.MONTH_KEY, TIME.QUARTER_KEY,
       TIME.CURRENT_YEAR, TIME.MONTH_EN as MONTH_NAME
from TIME_DIMENSION TIME) TIME
join SALES_FACT SALES_FACT
on (TIME.DAY_KEY = SALES_FACT.ORDER_DAY_KEY)
join PRODUCT PRODUCT on (PRODUCT.PRODUCT_KEY = SALES_FACT.PRODUCT_KEY)
where
  PRODUCT.PRODUCT_NAME in ('Aloe Relief', 'Course Pro Umbrella'))
and (TIME.MONTH_NAME in ('April 2004', 'February 2004', 'February
2006'))
group by
  TIME.MONTH_NAME,
  PRODUCT.PRODUCT_NAME
) D3
on ((D2.MONTH_NAME = D3.MONTH_NAME) and
    (D2.PRODUCT_NAME = D3.PRODUCT_NAME))

```

What Is the Coalesce Statement?

A `coalesce` statement is simply an efficient means of dealing with query items from conformed dimensions. It is used to accept the first non-null value returned from either query subject. This statement allows a full list of keys with no repetitions when doing a full outer join.

Why Is There a Full Outer Join?

A full outer join is necessary to ensure that all the data from each fact table is retrieved. An inner join gives results only if an item in inventory was sold. A right outer join gives all the sales where the items were in inventory. A left outer join gives all the items in inventory that had sales. A full outer join is the only way to learn what was in inventory and what was sold.

Modeling 1-n Relationships as 1-1 Relationships

If a 1-n relationship exists in the data but is modeled as a 1-1 relationship, SQL traps cannot be avoided because the information provided by the metadata to the IBM® Cognos® software is insufficient.

The most common problems that arise if 1-n relationships are modeled as 1-1 are the following:

- Double-counting for multiple-grain queries is not automatically prevented.

IBM Cognos software cannot detect facts and then generate a stitched query to compensate for double-counting, which can occur when dealing with hierarchical relationships and different levels of granularity across conformed dimensions.

- Multiple-fact queries are not automatically detected.

IBM Cognos software will not have sufficient information to detect a multiple-fact query. For multiple-fact queries, an inner join is performed and the loop join is eliminated by dropping the last evaluated join. Dropping a join is likely to lead to incorrect or unpredictable results depending on the dimensions and facts included in the query.

If the cardinality were modified to use only 1-1 relationships between query subjects or dimensions, the result of a query on Product Forecast and Sales with Time or Time and Product generates a single `Select` statement that drops one join to prevent a circular reference.

The example below shows that the results of this query are incorrect when compared with the results of individual queries against Sales or Product Forecast.

The results of individual queries are as follows.

MONTH_NAME	PRODUCT_NAME	QUANTITY
April 2004	Aloe Relief	1,410
April 2004	Course Pro Umbrella	132
February 2004	Aloe Relief	270
February 2006	Aloe Relief	88

MONTH_NAME	PRODUCT_NAME	EXPECTED_VOLUME
April 2004	Aloe Relief	1,690
April 2004	Course Pro Umbrella	125
February 2004	Aloe Relief	246
February 2004	Course Pro Umbrella	1
February 2006	Aloe Relief	92

When you combine these queries into a single query, the results are as follows.

MONTH_NAME	PRODUCT_NAME	QUANTITY	EXPECTED_VOLUME
April 2004	Aloe Relief	68,598	1,811,680
April 2004	Course Pro Umbrella	68,598	134,000
February 2004	Aloe Relief	29,672	105,780
February 2004	Course Pro Umbrella	29,672	430
February 2006	Aloe Relief	28,564	47,196

The SQL

Because a circular join path was detected in the model, the generated SQL did not include one of the relationships that was not necessary to complete the join path. In this example, the relationship between Time and Product Forecast was dropped.

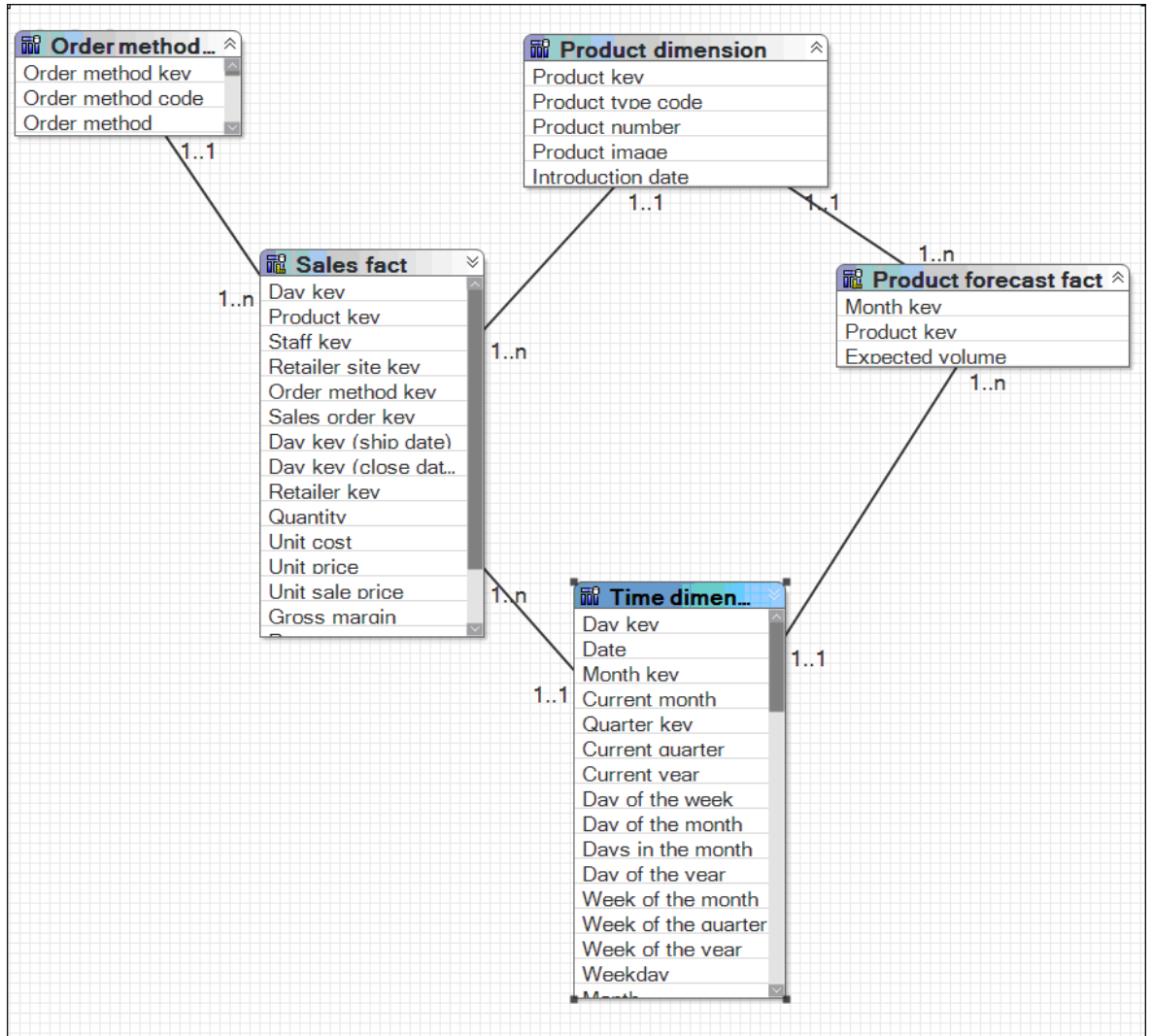
A circular join path rarely results in a query that produces useful results.

```
select
  TIME_.MONTH_NAME as MONTH_NAME,
  PRODUCT_LOOKUP.PRODUCT_NAME as PRODUCT_NAME,
  XSUM(SALES_FACT.QUANTITY for
    TIME_.CURRENT_YEAR, TIME_.QUARTER_KEY, TIME_.MONTH_KEY,
    PRODUCT.PRODUCT_LINE_CODE, PRODUCT.PRODUCT_TYPE_CODE,
    PRODUCT.PRODUCT_KEY ) as QUANTITY,
  XSUM(PRODUCT_FORECAST_FACT.EXPECTED_VOLUME for TIME_.CURRENT_YEAR,
    TIME_.QUARTER_KEY, TIME_.MONTH_KEY, PRODUCT.PRODUCT_LINE_CODE,
    PRODUCT.PRODUCT_TYPE_CODE, PRODUCT.PRODUCT_KEY ) as EXPECTED_VOLUME
from
  (select TIME.DAY_KEY, TIME.MONTH_KEY, TIME.QUARTER_KEY,
    TIME.CURRENT_YEAR, TIME.MONTH_EN as MONTH_NAME
  from TIME_DIMENSION TIME) TIME
join
  SALES_FACT on (TIME_.DAY_KEY = SALES_FACT.ORDER_DAY_KEY)
join
  PRODUCT_FORECAST_FACT on (TIME_.MONTH_KEY =
    PRODUCT_FORECAST_FACT.MONTH_KEY)
join
  PRODUCT (PRODUCT.PRODUCT_KEY = PRODUCT_FORECAST_FACT.PRODUCT_KEY)
where
  (PRODUCT.PRODUCT_NAME in ('Aloe Relief','Course Pro Umbrella'))
and
  (TIME_.MONTH_NAME in ('April 2004','February 2004','February 2006'))
```

```
group by
  TIME_.MONTH_NAME, PRODUCT.PRODUCT_NAME
```

Multiple-fact, Multiple-grain Query on Non-Conformed Dimensions

If a non-conformed dimension is added to the query, the nature of the result returned by the stitched query is changed. It is no longer possible to aggregate records to a lowest common level of granularity because one side of the query has dimensionality that is not common to the other side of the query. The result returned is really two correlated lists.



The Result

The results of individual queries on the respective star schemas look like this.

MONTH_NAME	PRODUCT_NAME	ORDER_METHOD	QUANTITY
April 2004	Aloe Relief	E-mail	114
April 2004	Aloe Relief	Fax	220
April 2004	Aloe Relief	Mail	100
April 2004	Aloe Relief	Sales visit	322
April 2004	Aloe Relief	Telephone	286
April 2004	Aloe Relief	Web	368
April 2004	Course Pro Umbrella	E-mail	22
April 2004	Course Pro Umbrella	Fax	28
April 2004	Course Pro Umbrella	Sales visit	82
February 2004	Aloe Relief	Mail	28
February 2004	Aloe Relief	Sales visit	102
February 2004	Aloe Relief	Telephone	28
February 2004	Aloe Relief	Web	112
February 2006	Aloe Relief	Sales visit	88

MONTH_NAME	PRODUCT_NAME	QUANTITY	EXPECTED_VOLUME
April 2004	Aloe Relief	1,410	1,690
April 2004	Course Pro Umbrella	132	125
February 2004	Aloe Relief	270	246
February 2004	Course Pro Umbrella		1
February 2006	Aloe Relief	88	92

Querying the same items from both star schemas yields the following result.

MONTH_NAME	PRODUCT_NAME	ORDER_METHOD	QUANTITY	EXPECTED_VOLUME
April 2004	Aloe Relief	Sales visit	322	1,690
April 2004	Aloe Relief	Telephone	286	1,690
April 2004	Aloe Relief	Web	368	1,690
April 2004	Aloe Relief	E-mail	114	1,690
April 2004	Aloe Relief	Fax	220	1,690
April 2004	Aloe Relief	Mail	100	1,690
April 2004	Course Pro Umbrella	E-mail	22	125
April 2004	Course Pro Umbrella	Fax	28	125
April 2004	Course Pro Umbrella	Sales visit	82	125
February 2004	Aloe Relief	Web	112	246
February 2004	Aloe Relief	Mail	28	246
February 2004	Aloe Relief	Sales visit	102	246
February 2004	Aloe Relief	Telephone	28	246
February 2004	Course Pro Umbrella			1
February 2006	Aloe Relief	Sales visit	88	92

In this result, the lower level of granularity for records from Sales results in more records being returned for each month and product combination. There is now a 1-n relationship between the rows returned from Product Forecast and those returned from Sales.

When you compare this to the result returned in the example of the multiple-fact, multiple grain query on conformed dimensions, you can see that more records are returned and that Expected Volume results are repeated across multiple Order Methods. Adding Order Method to the query effectively changes the relationship between Quantity data and Expected Volume data to a 1-n relationship. It is no longer possible to relate a single value from Expected Volume to one value from Quantity.

Grouping on the Month key demonstrates that the result in this example is based on the same data set as the result in the multiple-fact, multiple-grain query but with a greater degree of granularity.

The SQL

The stitched SQL generated for this example is very similar to the SQL generated in the multiple-fact, multiple-grain query (p. 352). The main difference is the addition of Order Method. Order Method is not a conformed dimension and affects only the query against the Sales Fact table.

```

select
  D2.QUANTITY as QUANTITY,
  D3.EXPECTED_VOLUME as EXPECTED_VOLUME,
  coalesce(D2.PRODUCT_NAME,D3.PRODUCT_NAME) as PRODUCT_NAME,
  coalesce(D2.MONTH_NAME,D3.MONTH_NAME) as MONTH_NAME,
  D2.ORDER_METHOD as ORDER_METHOD
from
  (select
    PRODUCT.PRODUCT_NAME as PRODUCT_NAME,
    TIME.MONTH_NAME as MONTH_NAME,
    ORDER_METHOD.ORDER_METHOD as ORDER_METHOD,
    XSUM(SALES_FACT.QUANTITY for TIME.CURRENT_YEAR,TIME.QUARTER_KEY,
    TIME.MONTH_KEY,PRODUCT.PRODUCT_LINE_CODE,PRODUCT.PRODUCT_TYPE_CODE,
    PRODUCT.PRODUCT_KEY,ORDER_METHOD_DIMENSION.ORDER_METHOD_KEY) as
QUANTITY
  from
    PRODUCT_DIMENSION PRODUCT
  join
    SALES_FACT SALES_FACT
  on (PRODUCT.PRODUCT_KEY = SALES_FACT.PRODUCT_KEY)
  join
    ORDER_METHOD_DIMENSION ORDER_METHOD
  on (ORDER_METHOD.ORDER_METHOD_KEY = SALES_FACT.ORDER_METHOD_KEY)
  join TIME_DIMENSION TIME
  on ( TIME.DAY_KEY = SALES_FACT.ORDER_DAY_KEY)
  where
    (PRODUCT.PRODUCT_NAME in ('Aloe Relief','Course Pro Umbrella'))
  and
    ( TIME.MONTH_NAME in ('April 2004','February 2004','February 2006'))
  group by
    PRODUCT.PRODUCT_NAME,
    TIME.MONTH_NAME,
    ORDER_METHOD.ORDER_METHOD
  ) D2
full outer join
(select
  PRODUCT.PRODUCT_NAME as PRODUCT_NAME,
  TIME.MONTH_NAME as MONTH_NAME,
  XSUM(PRODUCT_FORECAST_FACT.EXPECTED_VOLUME for TIME.CURRENT_YEAR,
  TIME.QUARTER_KEY,TIME.MONTH_KEY,PRODUCT.PRODUCT_LINE_CODE,
  PRODUCT.PRODUCT_TYPE_CODE,PRODUCT.PRODUCT_KEY) as EXPECTED_VOLUME
  from
    PRODUCT_DIMENSION PRODUCT
  join
    PRODUCT_FORECAST_FACT PRODUCT_FORECAST_FACT
  on (PRODUCT.PRODUCT_KEY = PRODUCT_FORECAST_FACT.PRODUCT_KEY)
  join
    (select
      TIME.CURRENT_YEAR as CURRENT_YEAR,
      TIME.QUARTER_KEY as QUARTER_KEY,
      TIME.MONTH_KEY as MONTH_KEY,
      XMIN(TIME.MONTH_NAME for TIME.CURRENT_YEAR, TIME.QUARTER_KEY,
      TIME.MONTH_KEY) as MONTH_NAME
    from
      TIME_DIMENSION TIME
    group by
      TIME.CURRENT_YEAR,
      TIME.QUARTER_KEY,
      TIME.MONTH_KEY
    ) TIME
  on (TIME.MONTH_KEY = PRODUCT_FORECAST_FACT.MONTH_KEY)
  where
    (PRODUCT.PRODUCT_NAME in ('Aloe Relief','Course Pro Umbrella'))
  and

```

```

(TIME.MONTH_NAME in ('April 2004','February 2004','February 2006'))
group by
PRODUCT.PRODUCT_NAME,
TIME.MONTH_NAME
) D3
on ((D2.PRODUCT_NAME = D3.PRODUCT_NAME) and
(D2.MONTH_NAME = D3.MONTH_NAME))

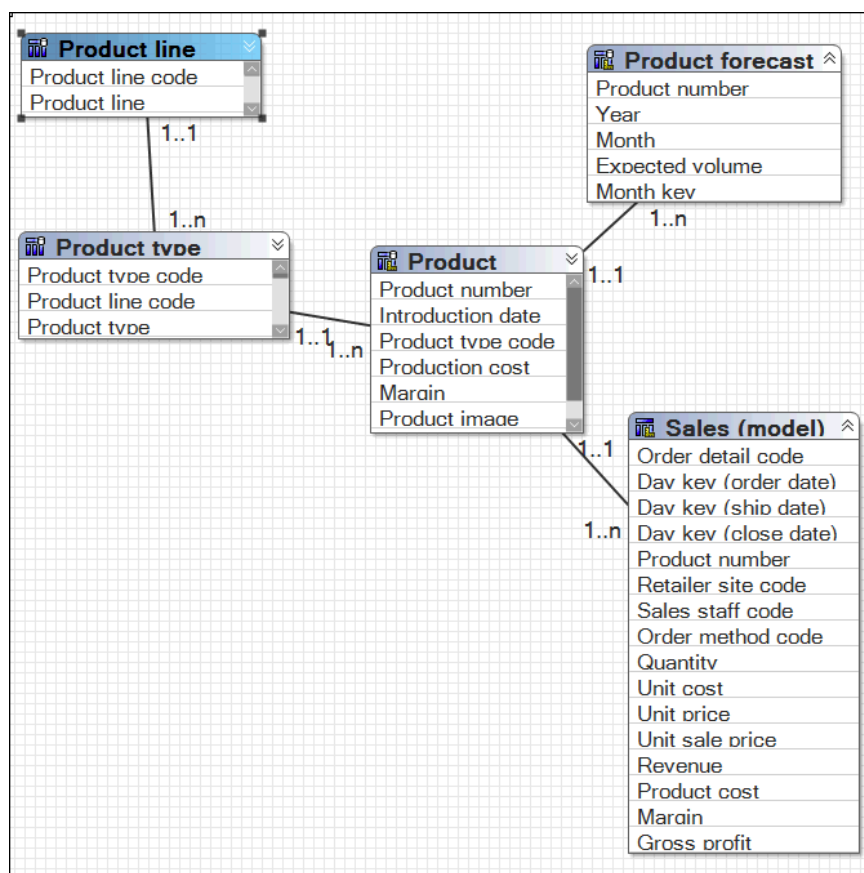
```

Resolving Ambiguously Identified Dimensions and Facts

A query subject is considered to be ambiguously defined if it participates in both n and 1 relationships to other query subjects. An ambiguously defined query subject is not always harmful from a query generation perspective. We suggest that you evaluate query subjects using the following cases. The goal of this evaluation is to prevent unnecessary query splits and to ensure that any splits that do occur are intentional and correct.

Query Subjects That Represent a Level of Hierarchy

One frequent case of an ambiguously defined query subject that is not harmful is where the query subject represents an intermediate level of a descriptive hierarchy. One example is the following Product hierarchy.



In this example, both Product type and Product could be identified as being ambiguously defined. However, this ambiguity is not detrimental to either the results generated or the performance of any query using one or more of these query subjects. You do not need to fix this query pattern because, using the rules for fact detection, only one fact is identified in any query that combines an

item from the Product forecast or Sales query subjects. It remains a best practice to collapse hierarchies into a single regular dimension when modeling for analysis purposes.

Some queries that can be written using this example include the following:

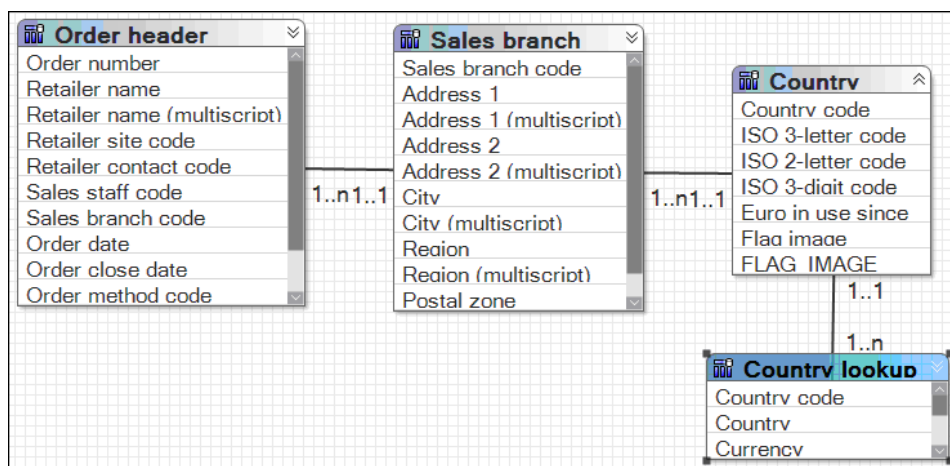
Items from these query subjects are used in a query:	Query subject that behaves as a fact in the query:
Product line and Product type	Product type
Product line, Product type, and Product	Product
Product line, Product type, Product, and Sales	Sales
Product line and Sales	Sales
Product type and Product forecast	Product forecast

Resolving Queries That Should Not Have Been Split

If queries are split and should not be split, you must resolve these queries.

Query subjects on the *n* side of all relationships are identified as facts. We can see that in the following example, Order Header and Country Multilingual are behaving as facts. In reality, the Country Multilingual query subject contains only descriptive information and seems to be a lookup table. From a dimensional or business modeling perspective, Country Multilingual is an extension of Country.

Why is it a problem to leave the model like this?



Test this model by authoring a report on the number of orders per city, per country. Using this model returns an incorrect result. The numbers are correct for the cities but some cities are shown as being in the wrong country. This is an example of an incorrectly related result.

COUNTRY	CITY	Number of Orders
Australia	Amsterdam	279
Austria	Bilbao	123
Belgium	Birmingham	164
Brazil	Boston	515
Canada	Calgary	123

Usually the first place to look when you see something like this is in the SQL.

The SQL

In this example, we see a stitched query, which makes sense if we have multiple facts in the model. A stitched query is essentially a query that attempts to stitch multiple facts together. It uses the relationships that relate the facts to each other as well as the determinants for the conformed, or common, dimensions defined in the model. A stitched query can be identified by two queries with a full outer join. The wrapper query must include a `coalesce` statement on the conformed dimensions.

Note the following problems in the SQL:

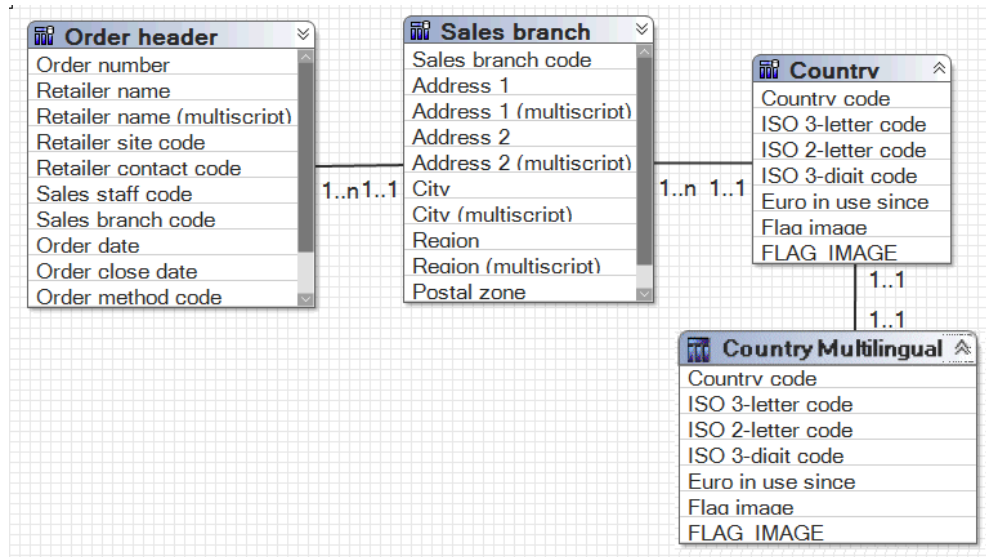
- The query has no `coalesce` statement.
- `RSUM` indicates an attempt to create a valid key.

```
select
  D3.COUNTRY as COUNTRY,
  D2.CITY as CITY,
  D2.number_of_orders as number_of_orders
from
  (select
    SALES_BRANCH.CITY as CITY,
    XCOUNT(ORDER_HEADER.ORDER_NUMBER for SALES_BRANCH.CITY) as
    number_of_orders,
    RSUM(1 at SALES_BRANCH.CITY order by SALES_BRANCH.CITY
    asc local)
    as sc
    from
      gosales.gosales.dbo.SALES_BRANCH SALES_BRANCH
    join
      gosales.gosales.dbo.ORDER_HEADER ORDER_HEADER
    on (SALES_BRANCH.SALES_BRANCH_CODE = ORDER_HEADER.SALES_BRANCH_CODE)
  group by
    SALES_BRANCH.CITY
  order by
    CITY asc
  ) D2
full outer join
  (select
    COUNTRY_MULTILINGUAL.COUNTRY as COUNTRY,
    RSUM(1 at COUNTRY_MULTILINGUAL.COUNTRY order by
    COUNTRY_MULTILINGUAL.COUNTRY asc local) as sc
  from
    gosales.gosales.dbo.COUNTRY_MULTILINGUAL COUNTRY_MULTILINGUAL
  group by
    COUNTRY_MULTILINGUAL.COUNTRY
  order by
    COUNTRY asc
  ) D3
on (D2.sc = D3.sc)
```

By looking at the stitched columns in each query, we see that they are being calculated on unrelated criteria. This explains why there is no apparent relationship between the countries and cities in the report.

So why do we see a stitched query? To answer that question, we must look at the model.

In this example, the query items used in the report came from different query subjects. Country came from Country Multilingual, City came from Sales Branch, and the Number of Orders came from a count on Order Number in the Order Header query subject.



The problem is that the query splits because the query engine sees this as a multiple-fact query. However, the split does not have a valid key on which to stitch because there is no item that both facts have in common.

There is more than one way to solve this problem but both require understanding the data.

Solution 1

You can add a filter to Country Multilingual that changes the cardinality of the relationship to 1-1.

```
Select *
from [GOSL].COUNTRY_MULTILINGUAL
Where
COUNTRY_MULTILINGUAL."LANGUAGE"='EN'
```

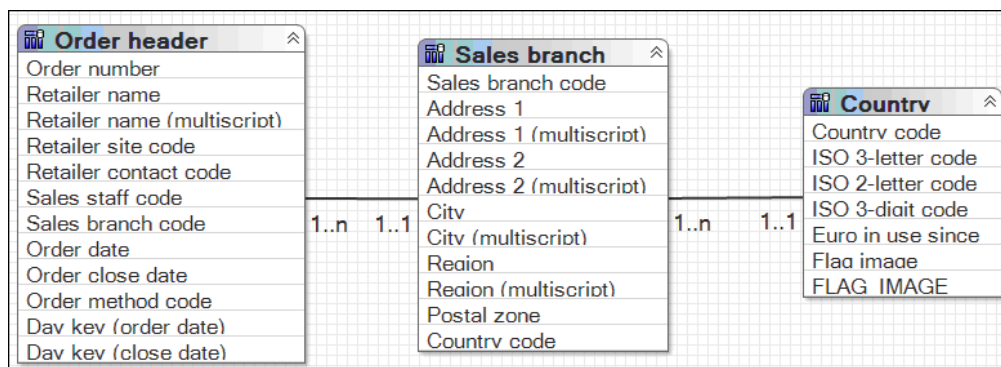
Or you can add a filter on the relationship and change the cardinality to 1-1.

```
COUNTRY.COUNTRY_CODE = COUNTRY_MULTILINGUAL.COUNTRY_CODE
and COUNTRY_MULTILINGUAL.LANGUAGE = 'EN'
```

Either choice results in a model that has a single fact in this query.

Solution 2

Simplify the model by consolidating the related query subjects. This gives the greatest benefit by simplifying the model and reducing the opportunities for error in query generation.



With either solution, the result of the query is now correct.

COUNTRY	CITY	Number of Orders
Australia	Melbourne	98
Austria	Wien	162
Belgium	Heverlee	94
Brazil	São Paulo	15
Canada	Calgary	123
Canada	Toronto	330

The SQL is no longer a stitched query.

```
select
  Country.c7 as COUNTRY,
  SALES_BRANCH.CITY as CITY,
  XCOUNT(ORDER_HEADER.ORDER_NUMBER for Country.c7,SALES_BRANCH.CITY)
  as number_of_orders
from
  (select
    COUNTRY.COUNTRY_CODE as c1,
    COUNTRY_MULTILINGUAL.COUNTRY as c7
  from
    gosales.gosales.dbo.COUNTRY COUNTRY
  join
    gosales.gosales.dbo.COUNTRY_MULTILINGUAL COUNTRY_MULTILINGUAL
  on (COUNTRY.COUNTRY_CODE = COUNTRY_MULTILINGUAL.COUNTRY_CODE)
  where COUNTRY_MULTILINGUAL.LANGUAGE='EN'
  ) Country
join
  gosales.gosales.dbo.SALES_BRANCH SALES_BRANCH
  on (SALES_BRANCH.COUNTRY_CODE = Country.c1)
join
  gosales.gosales.dbo.ORDER_HEADER ORDER_HEADER
  on (SALES_BRANCH.SALES_BRANCH_CODE = ORDER_HEADER.SALES_BRANCH_CODE)
group by
  Country.c7,
  SALES_BRANCH.CITY
```

Chapter 11: Upgrading Models

Upgrading a model prepares it to benefit from the new features in this release.

When you install the latest version of IBM® Cognos® software, most of the models created with earlier versions of the product are automatically upgraded. The packages that you published for users are automatically upgraded in the Content Manager database. These packages continue to function as they did previously, and all reports continue to run. Only upgrade the model in IBM Cognos Framework Manager if you want to modify your published metadata.

If a model does not upgrade successfully, reports that use the model will not run. You must then open the model in Framework Manager, upgrade it manually, and republish it.

The process of upgrading models differs depending on whether you are upgrading a model created in IBM Cognos ReportNet® or in earlier versions of IBM Cognos BI.

To upgrade your model, do the following:

- Verify the model before upgrading (p. 365).
- Open and upgrade the model (p. 365).
- Verify and repair the upgraded ReportNet® model (p. 367).

To find the most current product documentation, including all translated documentation, access one of the IBM Cognos Information Centers at <http://publib.boulder.ibm.com/infocenter/cogic/v1r0m0/index.jsp>.

Verifying the model before upgrading

The upgrade process does not resolve problems that existed in your model prior to the upgrade. Before you start the upgrade, verify your model and fix any reported problems in your existing environment.

When upgrading from IBM® Cognos® ReportNet® to IBM Cognos BI, ensure that the dimension information is set correctly in the ReportNet® model (p. 368).

Opening and upgrading the model

When you open an older model in IBM® Cognos® Framework Manager, you are prompted to upgrade the model.

When upgrading the model, you should:

- Understand the impact of the governors that are set during the upgrade (p. 366).
- Review the changed data types (p. 366).
- Understand the impact of the upgrade on query subjects based on SAP BW metadata (p. 367).

For more information, see ["Upgrading segmented and linked projects" \(p. 372\)](#).

Upgrade and governors

Several governors are set automatically during the upgrade process:

- The **Allow enhanced model portability at run time** governor ([p. 309](#)) is selected.

This governor is used when upgrading a ReportNet® 1.x model. It prevents rigid enforcement of data types so that an IBM® Cognos® BI model can function as a ReportNet® 1.x model until you update the data types in the metadata. After verifying that the model is upgraded successfully, clear this governor.

The status property of all data source query subjects is set to **Needs Re-evaluation**. The query engine ignores the data types in the model that are flagged as **Needs Re-evaluation** and retrieves the data type information from the data source. This may have a slightly negative impact on performance when running reports.

- The **Allow dynamic generation of dimension information** governor ([p. 310](#)) is selected.

This governor ensures consistent behavior with ReportNet® by deriving a form of dimension information from the relationships, key information, and index information in the data source.

When this governor is selected and a model contains query subjects but no dimensions, IBM Cognos BI generates queries that avoid double-counting. This also provides compatibility with ReportNet®. When you verify the model, a warning displays when this governor is selected.

- The **Shortcut Processing** governor ([p. 306](#)) is set to **Automatic**.

This governor controls how IBM Cognos BI processes shortcuts. When the governor is set to **Automatic**, the model works the same as in ReportNet®, that is, a shortcut that exists in the same folder as its target behaves as an alias or an independent instance. However, a shortcut that exists elsewhere in the model behaves as a reference to the original. When you create a new model, this governor is always set to **Explicit**.

- The **Suppress Null values for SAP BW data sources** governor ([p. 310](#)) is selected for SAP BW metadata only.

This governor controls the creation of outer joins in models based on SAP BW metadata.

Some queries can be very large because null values are not filtered out. Null suppression removes a row or column for which all of the values in the row or column are null (empty).

SAP BW performs null suppression. This reduces the amount of data transferred to the IBM Cognos studios and improves performance.

Upgrade and data types

IBM® Cognos® BI supports data types that are not available in IBM Cognos ReportNet®. This may impact how the data types are mapped during metadata import when upgrading.

Note: Mapping of the data types varies by data source vendor.

The following table shows the main differences between the data types in ReportNet® and in IBM Cognos BI.

ReportNet	IBM Cognos BI
char	nChar
decimal	numeric
varChar	nVarChar
varChar	timestampTZ
varChar	IntervalTZ

Data types are stored in the model so that IBM Cognos BI can avoid accessing the data source every time it needs to determine the data types for query items. This results in improved performance.

Some query items, mostly calculations, may appear broken after the upgrade. This may happen under the following circumstances:

- The data type of the underlying items for a calculation was changed and it is not possible to automatically assign the data type to the calculation.
- A saved calculation containing a prompt was not properly tested.

You must test the affected calculations and correct the issues.

Upgrade and query subjects that are based on SAP BW metadata

A query subject that is based on SAP BW metadata is not converted to a dimension. Instead, the upgrade process creates a hidden dimension and a new model query subject that have the same name and query items as the original query subject. The model query subject points to the newly created hidden dimension, which maintains compatibility with existing reports.

Model query subjects that do not contain levels and hierarchies are converted to measure dimensions.

After upgrading, review the model to ensure that the newly created regular and measure dimensions meet your analysis and reporting requirements.

Measures not added to an upgraded model after initial import

After upgrading a ReportNet® 1.x model to an IBM Cognos 8.3 model, measures that were added to the key figures folder for the underlying SAP BW object after the initial import are not added during the synchronization phase. The solution is to re-import the object.

This issue does not affect models in later versions of IBM Cognos BI.

Verifying and repairing the upgraded IBM Cognos ReportNet model

This section and its subsections apply only when upgrading models from IBM® Cognos® ReportNet® to IBM Cognos BI.

After updating the metadata, IBM Cognos Framework Manager prompts you to verify the model. You will see warnings for each object that contains either dimension information or one of the data types listed in the section ["Upgrade and data types" \(p. 366\)](#). When verifying large models, check one namespace at a time. You can also check an individual object.

The repair process first evaluates all selected items. This evaluation automatically resolves issues with new data types, and prompts you to repair dimension information in the model. You can select one or more check boxes and repair the items.

Verifying the model separately from the upgrade process offers a number of advantages. It allows you to:

- Open an existing model, and upgrade and publish it without any changes in query functionality.
- Take the time to reassess model requirements.
- Continue to design the production model, publish it without the dimensional information, and gradually move to dimensions and determinants.

During the upgrade process, you need to

- Convert query subjects with dimension information to either query subjects with determinants or to dimensions [\(p. 368\)](#).
- Select and repair objects [\(p. 371\)](#).

If you make changes to the model before verifying it, do not change the determinants. Doing so may result in losing the dimension information, and will not let you verify the model later.

Converting dimension information to either determinants or dimensions

In IBM® Cognos® ReportNet®, dimension information combined uniqueness with dimensional hierarchies. In IBM Cognos BI, dimension information for regular dimensions is divided between determinants and hierarchies. Determinants control uniqueness and granularity. This control is required for query subjects based on relational data sources, particularly in multiple-fact, multiple-grain queries. Hierarchies address dimensional concepts of hierarchies, levels, keys, and attributes for all data sources.

Query subjects with dimension information are not converted to determinants or to dimensions during the upgrade. The metadata that the dimension information previously specified is preserved in the model and continues to exist for the query subjects until they are repaired. You cannot change the dimension information in IBM Cognos BI, but you can upgrade this information to determinants or to dimensions [\(p. 369\)](#). Until you upgrade the query subjects, IBM Cognos BI uses the dimension information previously specified in ReportNet®.

If you make changes to the model without verifying it after the upgrade [\(p. 370\)](#), do not make changes to the determinants. This may cause you to lose the dimensional information. When reviewing the dimension information, you must understand how it is applied to the query subject, and how you will use the query subject in a model in IBM Cognos BI.

For more information, see ["Guidelines for Modeling Metadata" \(p. 319\)](#).

Mapping dimension information

You can map the dimension information in the ReportNet® model query subjects either to query subjects with determinants or to dimensions in the IBM Cognos BI model.

- **Determinants**

Use determinants to preserve existing reports while extending your application.

Determinants reflect granularity by representing subsets or groups of data in a query subject. They ensure correct aggregation of this repeated data. Determinants are imported based on unique key and index information in the data source.

Unlike the dimension information, model query subjects inherit determinants that are specified for the underlying query subjects. Specify the determinants as early as possible in the model, typically on the data source query subject. In cases where you specify different determinants for different granularity control, either create multiple instances of the data source query subject or remove determinants from the data source query subject. Then create new determinants on each model query subject that requires different granularity control.

- **Dimensions**

Use dimensions if you use your model in Analysis Studio, enable drilling up and down in reports, or access member functions in the studios. Only regular dimensions are created. During the upgrade, sufficient metadata does not exist to create measure dimensions so you must create them manually. You can then relate the dimensions to the measures by setting scope in the model.

If you upgrade query subjects to dimensions, the names of the query items in your model are changed. For example, a query item previously qualified as *namespace.query subject.query item* now is qualified as *namespace.dimension.hierarchy.level.query item*. Because this name change may make existing reports invalid, upgrade query subjects to dimensions only when you want to leverage an existing model to build a new application that requires dimensional capability.

Dimension information is mapped to determinants and dimensions as follows.

Dimension information	Determinants	Dimensions
Hierarchies	For relational data sources, the first hierarchy creates the determinant.	Hierarchies For SAP BW data sources, alternate hierarchies are upgraded to alternate hierarchies of a dimension.

Dimension information	Determinants	Dimensions
Levels	Determinants, either uniquely identified or group by	<p>Levels</p> <p>The first level of the hierarchy is automatically defined as the All level. It contains a single root member, which represents the top level of the hierarchy.</p> <p>You cannot delete or move the All level. You can change its name, description, and screen tip.</p>
Keys	<p>If Unique Key is not selected, key segments from higher levels are included in the key.</p> <p>If Unique Key is selected, only the key segment, or segments, for the level are included in the key.</p>	<p>_businessKey role</p> <p>Unique Level</p>
Alphabetically first text attribute		_memberCaption role
Attributes	<p>Attributes</p> <p>Unassociated attributes are assigned to the last determinant, which generally corresponds to the lowest level.</p>	<p>Can be manually assigned to be _memberDescription role, custom role, or no role</p>

What to review after converting

After converting query subjects with dimension information to query subjects with determinants, you must review the following settings:

- Uniquely Identified**
 When you define a unique determinant, you are indicating that the key contains enough information to identify a group within the data.
- Group By**
 You should specify **Group By** when you define a non-unique determinant. This indicates to IBM Cognos BI that, when the keys or attributes associated with that determinant are repeated in the data, it should apply aggregate functions and grouping to avoid double-counting.
- Multiple, or alternate, hierarchies that existed in ReportNet®**

If two hierarchies existed on a query subject in ReportNet®, only the first hierarchy is upgraded to a determinant for relational data sources. You must create a second query subject and manually specify the determinants for the other hierarchy.

For SAP BW data sources, alternate hierarchies are upgraded to determinants.

After converting query subjects with dimension information to dimensions, you must review the following settings:

- **_businessKey role**
This role represents the key for the level and can be assigned to only one attribute in a level.
- **Unique Level**
A unique level indicates that the keys of the levels above are not necessary to identify the members in this level.
- **_memberCaption role**
To leverage member functions in the IBM Cognos studios, you must assign a _memberCaption role to each level in a dimension. If there are no attributes for the level, the absence of a caption is highlighted when you verify the model.

All captions must have the string data type. If there is no attribute of this type available, create a calculation that is a string data type and assign the _memberCaption role to the new item. This is primarily an issue for Analysis Studio.
- **Attributes**
In general, include attributes in the dimension and associate them with the correct level. By default, they are included with no role. You can create custom roles or assign attributes to existing roles.
- **Multiple Hierarchies**
A regular dimension may have multiple hierarchies; however, you can use only one hierarchy at a time in a query. For example, you cannot use one hierarchy in the rows of a crosstab report and another hierarchy from the same dimension in the columns. If you need both hierarchies in the same report, you must create two dimensions, one for each hierarchy.

Selecting and repairing objects in the upgraded IBM Cognos ReportNet model

The repair process first evaluates all selected items. This evaluation automatically resolves issues with new data types, and prompts you to repair dimension information in the model.

The following warnings may appear when you verify and repair an upgraded model:

Warning	Description
Needs reevaluation	This message is most often related to data type changes. You can select and repair most items with this warning. Use the repair option to evaluate and upgrade specific metadata.

Warning	Description
Join expression conflicts with the determinant information defined in the query subject	Sometimes the index and key information that is specified for a query subject implies a level of granularity that does not match the relationships that are specified on a query subject.
None of the query items in this level have a caption role specified	<p>When defining levels, ensure that a business key and caption roles are specified. These roles are needed for member functions in the IBM Cognos studios and to assist in the member-oriented tree in Analysis Studio.</p> <p>All captions must have the string data type. If there is no attribute of this type available, create a calculation that is a string data type and assign the member caption role to the new item. This is primarily an issue for Analysis Studio.</p>
One or more determinants that describe the keys and attributes of the query subject should be specified	Determinants are based on key information in the data source. Determinants may not exist for a query subject upgraded from ReportNet®, especially for a model query subject. Use determinants to specify the granularity of the data in the query subject and the functional dependencies between query items. However, it is not mandatory to specify determinants for query subjects that represent a single level or fact data.

Upgrading segmented and linked projects

This information applies regardless of the product version from which you are upgrading.

For segmented projects, segments are automatically upgraded when you open the master project.

You must upgrade a linked project before upgrading any projects that use the linked project.

If you have a segmented or linked model that was created with a previous version of IBM® Cognos® Framework Manager, you must review the governor settings for each child and parent model before and after the upgrade. The governor settings for the top-level parent model are applied when publishing.

If you run a script that was created with a previous version of Framework Manager, the script bypasses the upgrade process. Review the accuracy of governor settings before and after running the script.

If a segmented project does not upgrade automatically, perform the following steps.

Steps

1. Open each segment as a separate project, starting with the lowest level segment in the hierarchy.
2. Follow the steps to upgrade the model([p. 365](#)).

Do not repair the segment.

3. After the upgrade is complete, save the project.
4. Upgrade each segment in the hierarchy, working back to the main project.
5. Repair the complete model in the master project.
6. After upgrading the master project, check in each segment, and then check in the master project.

Appendix A: Troubleshooting

You may encounter problems when working in Framework Manager. For other troubleshooting topics, see the IBM® Cognos® *Administration and Security Guide*.

Unable to Compare Two CLOBs in Oracle

If you are using Oracle and ask IBM® Cognos® BI to compare two CLOBs, such as where C2 = C3, you will see an Oracle runtime error.

To avoid this problem, use the DBMS_LOB.compare method:

```
where 0 = dmbs_lob.compare (c1, c2)
```

An Out of Memory Error with ERWin Imported Metadata

When you test query subjects based on a View table, an Out of Memory error may occur while performing the `sqlPrepareWithOptions` operation.

The solution is to create a data source query subject using the same Content Manager connection as the ERWin model.

Framework Manager Cannot Access the Gateway URI

You create a new project in Framework Manager and the following message appears:

Unable to access service at URL:

`http://hostname:80/ibmcognos/cgi-bin/cognos.cgi/`

`b_acton=xts.run&m=portal/close.xts`

Please check that your gateway URI information is configured correctly and that the service is available.

For further information please contact your service administrator.

This message appears if the gateway is not properly configured. The gateway URI must be set to the computer name where IBM® Cognos® BI is installed and reflect the type of gateway you are using. You must log on as an administrator to configure the gateway URI.

Steps to Configure the Gateway URI

1. Close Framework Manager.
2. In IBM Cognos Configuration, in the **Explorer** window, click **Environment**.
3. In the **Properties** window, in the **Gateway URI** box, type the appropriate value:
 - To use ISAPI, replace `cognos.cgi` with `cognosisapi.dll`.

- To use `apache_mod`, replace `cognos.cgi` with **`mod_cognos.dll`**.
 - To use a servlet gateway, type the following:
`http[s]://host:port/context_name/servlet/Gateway`
Note: *context_name* is the name you assigned to the ServletGateway Web application when you deployed the ServletGateway WAR file.
 - If you are not using a Web server, to use the dispatcher as the gateway, type the following:
`http[s]://host:port/p2pd/servlet/dispatch`
4. If required, change the host name portion of the **Gateway URI** from `localhost` to either the IP address of the computer or the computer name.
 5. From the **File** menu, click **Save**.
 6. From the **Actions** menu, click **Restart**.

Object Names Appear in the Wrong Language

When you import multiple languages from an SAP BW Query to a Framework Manager model, not all the object names retrieved from SAP BW appear in the correct language.

To avoid this problem, save the SAP BW Query again in each of the logon languages in Business Explorer Query Designer. The correct language texts will then show correctly in Framework Manager.

Full Outer Joins in Oracle Return Incorrect Results

When using an Oracle data source prior to version 10.2, full outer joins return incorrect data results. To avoid this problem, IBM Cognos BI processes these as local operations.

As a result of this processing, you must set the query processing to `limitedLocal` for any projects that expect explicit or implicit outer joins.

Error When Testing Query Subjects in a Model Imported from Teradata

You are using a model imported from Teradata. When you test some query subjects that contain graphic items, you see this error:

QE-DEF-0177 An error occurred while performing operation 'sqlOpenResult' status='-28'.

UDA-SQL-0114 The cursor supplied to the operation "sqlOpenResult" is inactive.

UDA-SQL-0107 A general exception has occurred during the operation "SgiCursor::doOpenResult ()".

[NCR][ODBC Teradata Driver][Teradata RDBMS] An unknown character string translation was requested.

The reason is that the GRAPHIC and VARGRAPHIC data types are not supported.

Error for Type-In SQL Query Subject

You define the following in DB2:

```
create type address as (
  number character (6),
  street varchar(35),
  city varchar(35)
)
MODE DB2SQL;
create table emp ( emp_no int, emp_address address);
Select e.emp_no, e.emp_address..street from emp e
SQL0206N "aBmtQuerySubject.2
```

When you define a type-in SQL query subject, an error appears because of the name assigned for the attribute reference in the structured type.

To resolve this problem, you have two options:

- assign a simple correlation name to the column in the original query subject, such as `Select e.emp_no, e.emp_address..street as "ABC" from emp e`
- use pass-through notation for the query subject by surrounding the column with double curly brackets (`{{ }}`)

QE-DEF-0259 Error

This error occurs if you use braces `{ }` in the wrong position in an expression. IBM® Cognos® BI expects anything between the braces `{ }` to be at the same level as a function. If you have used braces elsewhere in an expression, you will see the following error message:

QE-DEF-0259 There was a parsing error

You can also use braces to send unique syntax to the data source. For example, your database uses a keyword for a function, but this keyword is not used in IBM Cognos BI.

IBM Cognos BI does not validate the syntax you enter between braces. The syntax is simply sent to the data source.

The solution is to make sure that braces are not used in the wrong positions.

For example, you type the following in an expression:

```
[ss_ole_both].[authors_lith].[au_id] = [ss_ole_both].[authors_latin].[au_id]
{ collate Lithuanian_CI_AI}
```

You see the following error message:

QE-DEF-0259 There was a parsing error before or near position: 75, text starting at position: 5
 "le_both].[authors_lith].[au_id]=[ss_ole_both].[authors_latin].[au_id]{"

Meanwhile the following expression is valid:

```
{ Q3.au_id } = { Q4.au_id collate lithuanian_CI_AI
}
```

Externalized Key Figures Dimension Retains Old Prompt Value

You have a key figures dimension (SAP BW) that contains an optional prompt. If you externalize this dimension as a csv or tab file, the externalized file does not contain all the rows of data. This is because the prompt value is retained.

For example, you set the prompt value for the dimension when testing the dimension in Framework Manager. The prompt value is kept in the cache. Even if you clear the value of the prompt in the Prompt dialog box, externalizing the key figures dimension results in a file containing data that is filtered by the most recently used prompt.

To avoid this problem, do one of the following:

- Do not test the key figures dimension before you externalize it.
- Close the model, open it again, and externalize the key figures dimension.

Older Models Display Level Object Security

If you are using a previously-created IBM® Cognos® model, object security on a level may have been defined. Object security on a level is not supported.

The solution is to verify and repair the older model before publishing it.

Steps

1. From the **Project** menu, click **Verify Model**.
2. Select the security view that references a level and click **Repair**.

Exporting a Framework Manager Model to a CWM File Fails With Error *MILOG.TXT was not found*

Exporting a Framework model to a CWM file fails with error *MILOG.TXT was not found* when the path contains Japanese characters.

Do one of the following to solve this problem:

- Specify an export path that does not use Japanese characters.
- Change the system default language on your computer to Japanese. You can set the system default language in the Control Panel, under **Regional and Language Options** -> **Advanced**. For more information on how to do this, refer to the Windows® operating system help.

Difference in SQL for Inner Joins After Upgrading to IBM Cognos BI, Version 8.3 and Later

If you migrated from a version of the product earlier than 8.3, there can be differences in the generation of SQL used for the `INNER JOIN` syntax.

You can control the SQL syntax used for inner joins by configuring the setting for the **SQL Join Syntax** governor. The SQL join syntax generated in all versions of IBM Cognos BI produces the same result.

If you are using RDBMS materialization technology which can be implemented using either implicit or explicit syntax, you must ensure that you select the same syntax setting for the **SQL Join Syntax** governor in your model.

Full Outer Joins Not Sent to Oracle 9i and 10GR1

By default, IBM® Cognos® BI will not send full outer joins to ORACLE 9i and 10GR1 due to Oracle bug #2874433. This requires using limited local processing in IBM Cognos BI.

To enable full outer joins with Oracle, you must

- ensure that you have the required patch sets, which include the fix for bug#2874433
- modify the cogdmor.ini file to turn on full outer joins (Full_outer_join=T)

Because any manual edits to the ini settings are overwritten by the next installation, you must manually replicate them on all machines where you installed IBM Cognos BI or Framework Manager.

Unexplained Discrepancies in Number Calculations

You might find unexplained discrepancies in number calculations due to round-off errors. For example:

- You run regression tests and find differences in numbers. They are different only because of the rounding off of decimal places.
- You choose not display zeros in reports, but the zeros are displayed anyway because there are decimal places (0.00000000000000426, for example) that are rounded off to zero in reports.

Round-off problems are not specific to IBM® Cognos® software. They can occur in any environment where rounding off occurs.

Binary Round-Off Errors

Discrepancies in calculations might occur due to binary round-off errors. For example, if the number 1.1 is represented as a binary floating point number and your report format includes a large number of decimal places, the number 1.1 might actually be something like 1.09999999999997.

If your report is formatted to use only one decimal point, decimal round-off takes place, compensating for the binary round-off. So the number appears to be 1.1 when it is really 1.09999999999997. When the number is used in calculations, you might get round-off errors. For example, Microsoft® Excel calculations use binary numbers (without rounding off decimal places) but formatting in reports shows rounded off decimal places, which can create small discrepancies.

Division Round-Off Errors

Calculations that involve division typically incur round-off errors, regardless of how the numbers are represented. Examples of such calculations are Average and Percent of Base.

Design Guidelines to Minimize Round-Off Effect

The best solution is to change the underlying database schema or cube model but that may not always be possible. Another solution is to minimize the round-off effect by following these guidelines when authoring reports and creating models in FrameWork Manager and external OLAP cubes:

- Avoid storing data in floating point format whenever possible. This is especially true for currency values, which should be stored as either fixed-point decimals or as integers with a scale value such as 2.

For example, in a cube, the Revenue for Camping Equipment in 2004 is \$20,471,328.88. If revenue details are stored as floating point numbers, round-off errors might occur when revenue is calculated.

The round up errors might have slight differences, depending on the order of calculation. If revenue for Products is calculated first and revenue for Time is calculated second, you might get a different round-off error than if Time is calculated first and Products is calculated second.

Total revenue might be calculated as the number above. Or there might be slight discrepancies, for example, \$20,471,328.8800001 as opposed to \$20,471,328.88. The internal number might be slightly different than what is displayed. The number might even be for different runs of the same report, depending on the order that the OLAP engine uses for calculation.

- In reports, avoid division whenever possible. When division is unavoidable, try to do it as late as possible in the calculation process. For example, instead of `Total([Revenue]/1000)`, use `Total([Revenue])/1000`.
- When doing comparisons, add a margin to allow for round-off. For example, you may want `[Profit %]` to be a fractional value formatted as a percentage with no decimals. However, the filter `[Profit %]<>0` (or `[Profit %] NOT BETWEEN 0 and 0`) rejects zero values and may still return values that appear to be 0% after formatting.

To avoid this, filter in one of these two ways:

- `[Profit %] NOT BETWEEN -0.005 and 0.005`
- `([Profit %] <- 0.005) OR ([Profit %]> 0.005)`

Note that 0.005 is equivalent to 0.5%, which displays as either 0% or 1%, depending on floating point precision losses.

In some cases, you may prefer control round-off errors by rounding values explicitly. For example, instead of `[Profit %]`, use `round([Profit %],2)`.

- Recalculate numbers every time instead of reusing calculations that might contain rounded off decimals.

There might be additional considerations for Microsoft® Analysis Services 2005/2008, especially when comparing report results from different runs (as happens in Lifecycle Manager). Refer to Microsoft documentation for more information.

Appendix B: Using the Expression Editor

An expression is any combination of operators, constants, functions, and other components that evaluates to a single value. You build expressions to create calculation and filter definitions. A calculation is an expression that you use to create a new value from existing values contained within a data item. A filter is an expression that you use to retrieve a specific subset of records.

The expression editor shows the expression components that are supported by the data source in which the metadata is stored. For example, if you import metadata from an Oracle data source, the expression editor shows only the elements that are supported in Oracle.

If you are using an IBM® DB2® data source, note that the subtract operator is invalid if you combine the datatypes `timestamp2` and `packed decimal`.

When creating an expression that will be used in a double-byte environment, such as Japanese, the only special characters that will work are ASCII-7 and ~ -- || - \$ ¢ £ ¬.

Quality of Service Indicators

Not all data sources support functions the same way. The data modeler can set a quality of service indicator on functions to give a visual clue about the behavior of the functions. Report authors can use the quality of service indicators to determine which functions to use in a report. The quality of service indicators are:

- not available (X)

This function is not available for any data source in the package.

- limited availability (!!)

The function is not available for some data sources in the package.

- limited support (!)

The function is available for all data sources in the package but is not naturally supported for that data source. IBM® Cognos® uses a local approximation for that function. Because an approximation is used, performance can be poor and the results may not be what you expect.

- unconstrained (check mark)

The function is available for all data sources.

SAP BW Support

SAP BW does not support all operators or summaries. This can be confusing if you have imported SAP BW metadata and non-SAP BW metadata into the same model.

SAP BW does not support the following operators:

- like
- lookup

SAP BW does not support the following member summaries:

- date-time
- interval
- interval month
- interval day
- interval day to hour
- interval day to minute
- interval day to second
- interval hour
- interval hour to minute
- interval hour to second
- interval minute
- interval minute to second
- interval second
- interval year
- interval year to month
- moving
- running
- time with time zone
- timestamp with time zone

Cell values are date, number, or time. Attribute values are strings.

Searching for Values May Return Unexpected Results

In the expression editor, when searching for values for a data item, the results you obtain may contain unexpected results if the data item is not a string data type. Because users can edit the expression for a data item, IBM Cognos BI cannot determine with certainty what the data type is.

Therefore, IBM Cognos BI guesses the data type of the data item by looking at its aggregate and rollup aggregate set.

Calculation Components

You build calculations, or expressions, in the expression editor using the following components:

- operators ([p. 383](#))

- summaries ([p. 391](#))
- member summaries ([p. 402](#))
- constants ([p. 405](#))
- constructs ([p. 407](#))
- business date/time functions ([p. 408](#))
- block functions ([p. 413](#))
- macro functions ([p. 414](#))
- common functions ([p. 424](#))
- dimensional functions ([p. 431](#))
- DB2® ([p. 453](#))
- Informix ([p. 469](#))
- Microsoft® Access ([p. 475](#))
- Netezza ([p. 483](#))
- Oracle ([p. 490](#))
- Red Brick ([p. 499](#))
- Microsoft SQL Server ([p. 504](#))
- Teradata ([p. 511](#))
- SAP BW ([p. 517](#))
- Sybase ([p. 519](#))
- Postgres ([p. 527](#))
- Vertica ([p. 533](#))
- Paracel ([p. 538](#))
- MySQL ([p. 541](#))
- Greenplum ([p. 545](#))
- report functions ([p. 551](#))

Operators

Operators specify what happens to the values on either side of the operator. Operators are similar to functions, in that they manipulate data items and return a result.

(

Identifies the beginning of an expression.

Syntax

(expression)

)

Identifies the end of an expression.

Syntax

(expression)

*

Multiplies two numeric values.

Syntax

value1 * value2

,

Separates expression components.

Syntax

expression (parameter1, parameter2)

/

Divides two numeric values.

Syntax

value1 / value2

||

Concatenates, or joins, strings.

Syntax

string1 || string2

+

Adds two numeric values.

Syntax

value1 + value2

-

Subtracts two numeric values or negates a numeric value.

Syntax

value1 - value2
or
- value

<

Compares the values that are represented by "value1" against "value2" and retrieves the values that are less than "value2".

Syntax

```
value1 < value2
```

<=

Compares the values that are represented by "value1" against "value2" and retrieves the values that are less than or equal to "value2".

Syntax

```
value1 <= value2
```

<>

Compares the values that are represented by "value1" against "value2" and retrieves the values that are not equal to "value2".

Syntax

```
value1 <> value2
```

=

Compares the values that are represented by "value1" against "value2" and retrieves the values that are equal to "value2".

Syntax

```
value1 = value2
```

>

Compares the values that are represented by "value1" against "value2" and retrieves the values that are greater than "value2".

Syntax

```
value1 > value2
```

->

Separates the components in a literal member expression.

Syntax

```
[namespace].[dimension].[hierarchy].[level]->[L1]
```

>=

Compares the values that are represented by "value1" against "value2" and retrieves the values that are greater than or equal to "value2".

Syntax

```
value1 >= value2
```

and

Returns "true" if the conditions on both sides of the expression are true.

Syntax

```
argument1 and argument2
```

auto

Works with summary expressions to define the scope to be adjusted based on the grouping columns in the query. The scope is context-dependent.

Syntax

```
aggregate_function ( expression AUTO )
```

between

Determines if a value falls in a given range.

Syntax

```
expression between value1 and value2
```

Example

```
[Revenue] between 200,000 and 300,000
```

Result: Returns the number of results with revenues between 200,000 and 300,000.

Revenue	Between
----	----
\$332,986,338.06	false
\$230,110,270.55	true
\$107,099,659.94	false

case

Works with when, then, else, and end. Case identifies the beginning of a specific situation, in which when, then, and else actions are defined.

Syntax

```
case expression { when expression then expression } [ else expression ] end
```

contains

Determines if "string1" contains "string2".

Syntax

```
string1 contains string2
```

currentMeasure

Keyword that can be used as the first argument of member summary functions. This function appears in the Total Revenue by Country sample report in the GO Data Warehouse (query) package.

Syntax

```
aggregate_function ( currentMeasure within set expression )
```

default

Works with the lookup construct.

Syntax

```
lookup (....) in (....) default (....)
```

distinct

A keyword used in an aggregate expression to include only distinct occurrences of values. See also the function unique.

Syntax

```
distinct dataItem
```

Example

```
count ( distinct [OrderDetailQuantity] )
```

Result: 1704

else

Works with the if or case constructs. If the if condition or the case expression are not true, then the else expression is used. This function appears in the Top 10 Retailers for 2005 sample report in the GO Data Warehouse (analysis) package.

Syntax

```
if ( condition ) then .... else ( expression ) , or case .... else ( expression )
end
```

end

Indicates the end of a case or when construct.

Syntax

```
case .... end
```

ends with

Determines if "string1" ends with "string2".

Syntax

```
string1 ends with string2
```

for

Works with summary expressions to define the scope of the aggregation in the query.

Syntax

```
aggregate_function ( expression for expression { , expression } )
```

for all

Works with summary expressions to define the scope to be all the specified grouping columns in the query. See also the for clause.

Syntax

```
aggregate_function ( expression for ALL expression { , expression } )
```

for any

Works with summary expressions to define the scope to be adjusted based on a subset of the grouping columns in the query. Equivalent to the for clause.

Syntax

```
aggregate_function ( expression for ANY expression { , expression } )
```

for report

Works with summary expressions to set the scope to be the whole query. See also the for clause. This function appears in the Customer Returns and Satisfaction sample report in the GO Data Warehouse (analysis) package.

Syntax

```
aggregate_function ( expression for report )
```

if

Works with the then and else constructs. If defines a condition; when the if condition is true, the then expression is used. When the if condition is not true, the else expression is used. This function appears in the Top 10 Retailers for 2005 sample report in the GO Data Warehouse (analysis) package.

Syntax

```
if ( condition ) then ( expression ) else ( expression )
```

in

Determines if "expression1" exists in a given list of expressions.

Syntax

```
expression1 in ( expression_list )
```

in_range

Determines if "expression1" exists in a given list of constant values or ranges.

Syntax

```
expression1 in_range { constant : constant [ , constant : constant ] }
```

Example 1

```
[code] in_range { 5 }
```

Result: This is equivalent to `[code] = 5`.

Example 2

```
[code] in_range { 5: } 
```

Result: This is equivalent to `[code] >= 5`.

Example 3

```
[code] in_range { :5 }
```

Result: This is equivalent to `[code] <= 5`.

Example 4

```
[code] in_range { 5:10 }
```

Result: This is equivalent to `([code] >= 5 and [code] <= 10)`.

Example 5

```
[code] in_range { :5,10,20: }
```

Result: This is equivalent to `([code] <= 5 or [code] = 10 or [code] >= 20)`.

is missing

Determines if "value" is undefined in the data.

Syntax

```
value is missing
```

is null

Determines if "value" is undefined in the data.

Syntax

```
value is null
```

is not missing

Determines if "value" is defined in the data.

Syntax

```
value is not missing
```

is not null

Determines if "value" is defined in the data.

Syntax

```
value is not null
```

like

Determines if "string1" matches the pattern of "string2".

Syntax

```
string1 LIKE string2
```

lookup

Finds and replaces data with a value you specify. It is preferable to use the case construct.

Syntax

```
lookup ( name ) in ( value1 --> value2 ) default ( expression )
```

Example

```
lookup ( [Country] ) in ( 'Canada'--> ( [List Price] * 0.60), 'Australia'-->
( [List Price] * 0.80 ) ) default ( [List Price] )
```

not

Returns TRUE if "argument" is false or returns FALSE if "argument" is true.

Syntax

```
NOT argument
```

or

Returns TRUE if either of "argument1" or "argument2" are true.

Syntax

```
argument1 or argument2
```

prefilter

Performs a summary calculation before applying the summary filter.

Syntax

```
summary ([expression] prefilter)
```

rows

Counts the number of rows output by the query. Use with Count ().

Syntax

```
count ( ROWS )
```

starts with

Determines if "string1" starts with "string2".

Syntax

```
string1 starts with string2
```

then

Works with the if or case constructs. When the if condition or the when expression are true, the then expression is used. This function appears in the Top 10 Retailers for 2005 sample report in the GO Data Warehouse (analysis) package.

Syntax

```
if ( condition ) then ..., or case expression when expression then .... end
```

when

Works with the case construct. You can define conditions to occur when the when expression is true.

Syntax

```
case [expression] when ... end
```

Summaries

This list contains predefined functions that return either a single summary value for a group of related values or a different summary value for each instance of a group of related values.

aggregate

Returns a calculated value using the appropriate aggregation function, based on the aggregation type of the expression. This function appears in the Budget vs. Actual sample report in the GO Data Warehouse (analysis) package.

Syntax

```
aggregate ( expression [ auto ] )
aggregate ( expression for [ all|any ] expression { , expression } )
aggregate ( expression for report )
```

average

Returns the average value of selected data items. Distinct is an alternative expression that is compatible with earlier versions of the product.

Syntax

```
average ( [ distinct ] expression [ auto ] )
average ( [ distinct ] expression for [ all|any ] expression { , expression } )
average ( [ distinct ] expression for report )
```

Example

```
average ( Sales )
```

Result: Returns the average of all Sales values.

count

Returns the number of selected data items excluding null values. Distinct is an alternative expression that is compatible with earlier versions of the product.

Syntax

```
count ( [ distinct ] expression [ auto ] )  
count ( [ distinct ] expression for [ all|any ] expression { , expression } )  
count ( [ distinct ] expression for report )
```

Example

```
count ( Sales )
```

Result: Returns the total number of entries under Sales.

maximum

Returns the maximum value of selected data items. Distinct is an alternative expression that is compatible with earlier versions of the product.

Syntax

```
maximum ( [ distinct ] expression [ auto ] )  
maximum ( [ distinct ] expression for [ all|any ] expression { , expression } )  
maximum ( [ distinct ] expression for report )
```

Example

```
maximum ( Sales )
```

Result: Returns the maximum value out of all Sales values.

median

Returns the median value of selected data items.

Syntax

```
median ( expression [ auto ] )  
median ( expression for [ all|any ] expression { , expression } )  
median ( expression for report )
```

minimum

Returns the minimum value of selected data items. Distinct is an alternative expression that is compatible with earlier versions of the product.

Syntax

```
minimum ( [ distinct ] expression [ auto ] )  
minimum ( [ distinct ] expression for [ all|any ] expression { , expression } )  
minimum ( [ distinct ] expression for report )
```

Example

```
minimum ( Sales )
```

Result: Returns the minimum value out of all Sales values.

moving-average

Returns a moving average by row for a specified set of values of over a specified number of rows. The "<for-option>" defines the scope of the function. The "at" option defines the level of aggregation and can be used only in the context of relational datasources.

Syntax

```
moving-average ( numeric_expression , numeric_expression [ at expression
{ , expression } ] [ <for-option> ] [ prefilter ] )
moving-average ( numeric_expression , numeric_expression [ <for-option> ]
[ prefilter ] )
<for-option> ::= for expression { , expression }|for report|auto
```

Example

```
moving-average ( Qty , 3 )
```

Result: For each row, returns the quantity and a moving average of the current row and the preceding two rows.

Qty	Moving-Average (Qty, 3)
-----	-----
200	200
700	450
400	433.3333
200	433.3333
200	266.6667
500	300.0000

moving-total

Returns a moving total by row for a specified set of values over a specified number of rows. The "<for-option>" defines the scope of the function. The "at" option defines the level of aggregation and can be used only in the context of relational datasources.

Syntax

```
moving-total ( numeric_expression , numeric_expression [ at expression
{ , expression } ] [ <for-option> ] [ prefilter ] )
moving-total ( numeric_expression , numeric_expression [ <for-option> ]
[ prefilter ] )
<for-option> ::= for expression { , expression }|for report|auto
```

Example

```
moving-total ( Qty , 3 )
```

Result: For each row, returns the quantity and a moving total of the current row and the preceding two rows.

Qty	Moving-Total (Qty, 3)
-----	-----
200	200
700	900
400	1300
200	1300
200	800
500	900

percentage

Returns the percent of the total value for selected data items. The "<for-option>" defines the scope of the function. The "at" option defines the level of aggregation and can be used only in the context of relational datasources. This function appears in the Percentage Calculation (by year) interactive sample report.

Syntax

```
percentage ( numeric_expression [ at expression { , expression } ]
[ <for-option> ] [ prefilter ] )
percentage ( numeric_expression [ <for-option> ] [ prefilter ] )
<for-option> ::= for expression { , expression }|for report|auto
```

Example

```
percentage ( Sales 98 )
```

Result: Returns the percentage of the total sales for 1998 that is attributed to each sales representative.

Sales Rep	Sales 98	Percentage
-----	-----	-----
Gibbons	60646	7.11%
Flertjan	62523	7.35%
Cornel	22396	2.63%

percentile

Returns a value, on a scale of one hundred, that indicates the percent of a distribution that is equal to or below the selected data items. The "<for-option>" defines the scope of the function. The "at" option defines the level of aggregation and can be used only in the context of relational datasources.

Syntax

```
percentile ( numeric_expression [ at expression { , expression } ]
[ <for-option> ] [ prefilter ] )
percentile ( numeric_expression [ <for-option> ] [ prefilter ] )
<for-option> ::= for expression { , expression }|for report|auto
```

Example

```
percentile ( Sales 98 )
```

Result: For each row, returns the percentage of rows that are equal to or less than the quantity value of that row.

Qty	Percentile (Qty)
-----	-----
800	1
700	0.875
600	0.75
500	0.625
400	0.5
400	0.5
200	0.25
200	0.25

quantile

Returns the rank of a value within a range that you specify. It returns integers to represent any range of ranks, such as 1 (highest) to 100 (lowest). The "<for-option>" defines the scope of the function. The "at" option defines the level of aggregation and can be used only in the context of relational datasources.

Syntax

```
quantile ( numeric_expression , numeric_expression [ at expression  
{ , expression } ] [ <for-option> ] [ prefilter ] )  
quantile ( numeric_expression , numeric_expression [ <for-option> ]  
[ prefilter ] )  
<for-option> ::= for expression { , expression }|for report|auto
```

Example

```
quantile ( Qty , 4 )
```

Result: Returns the quantity, the rank of the quantity value, and the quantity values broken down into 4 quantile groups (quartiles).

Qty	Rank (Qty)	Quantile (Qty, 4)
-----	-----	-----
800	1	1
700	2	1
600	3	2
500	4	2
400	5	3
400	5	3
200	7	4
200	7	4

quartile

Returns the rank of a value, represented as integers from 1 (highest) to 4 (lowest), relative to a group of values. The "<for-option>" defines the scope of the function. The "at" option defines the level of aggregation and can be used only in the context of relational datasources.

Syntax

```
quartile ( numeric_expression [ at expression { , expression } ] [ <for-option> ]  
  [ prefilter ] )  
quartile ( numeric_expression [ <for-option> ] [ prefilter ] )  
<for-option> ::= for expression { , expression }|for report|auto
```

Example

```
quartile ( Qty )
```

Result: Returns the quantity and the quartile of the quantity value represented as integers from 1 (highest) to 4 (lowest).

Qty	Quartile (Qty)
-----	-----
450	1
400	1
350	2
300	2
250	3
200	3
150	4
100	4

rank

Returns the rank value of selected data items. The sort order is optional; descending order (DESC) is assumed by default. If two or more rows tie, then there is a gap in the sequence of ranked values

(also known as Olympic ranking). The "<for-option>" defines the scope of the function. The "at" option defines the level of aggregation and can be used only in the context of relational datasources. Distinct is an alternative expression that is compatible with earlier versions of the product. Null values are ranked last. This function appears in the Top 10 Retailers for 2005 sample report in the GO Data Warehouse (analysis) package.

Syntax

```
rank ( expression [ ASC|DESC ] { , expression [ ASC|DESC ] } [ at expression
{ , expression } ] [ <for-option> ] [ prefilter ] )
rank ( [ distinct ] expression [ ASC|DESC ] { , expression [ ASC|DESC ] }
[ <for-option> ] [ prefilter ] )
<for-option> ::= for expression { , expression }|for report|auto
```

Example

```
rank ( Sales 98 )
```

Result: For each row, returns the rank value of sales for 1998 that is attributed to each sales representative. Some numbers are skipped when a tie between rows occurs.

Sales Rep	Sales 98	Rank
Gibbons	60000	1
Flertjan	50000	2
Cornel	50000	2
Smith	48000	4

running-average

Returns the running average by row (including the current row) for a set of values. The "<for-option>" defines the scope of the function. The "at" option defines the level of aggregation and can be used only in the context of relational datasources.

Syntax

```
running-average ( numeric_expression [ at expression { , expression } ]
[ <for-option> ] [ prefilter ] )
running-average ( numeric_expression [ <for-option> ] [ prefilter ] )
<for-option> ::= for expression { , expression }|for report|auto
```

Example

```
running-average ( Qty )
```

Result: For each row, returns the quantity and a running average of the current and the previous rows.

Name	Qty	Avg	Running-Average for name
-----	-----	-----	-----
Smith	7	5	7
Smith	3	5	5
Smith	6	5	5.33
Smith	4	5	5
Wong	3	4	3
Wong	5	4	4

running-count

Returns the running count by row (including the current row) for a set of values. The "<for-option>" defines the scope of the function. The "at" option defines the level of aggregation and can be used only in the context of relational datasources.

Syntax

```
running-count ( numeric_expression [ at expression { , expression } ]
[ <for-option> ] [ prefilter ] )
running-count ( numeric_expression [ <for-option> ] [ prefilter ] )
<for-option> ::= for expression { , expression }|for report|auto
```

Example

```
running-count ( Qty )
```

Result: For each row, returns the quantity and a running count of the position of the current row.

Name	Qty	Count	Running-Count for name
-----	-----	-----	-----
Smith	7	4	1
Smith	3	4	2
Smith	6	4	3
Smith	4	4	4
Wong	3	3	1
Wong	5	3	2

running-difference

Returns a running difference by row, calculated as the difference between the value for the current row and the preceding row, (including the current row) for a set of values. The "<for-option>" defines the scope of the function. The "at" option defines the level of aggregation and can be used only in the context of relational datasources.

Syntax

```
running-difference ( numeric_expression [ at expression { , expression } ]
[ <for-option> ] [ prefilter ] )
```

```
running-difference ( numeric_expression [ <for-option> ] [ prefilter ] )
<for-option> ::= for expression { , expression }|for report|auto
```

Example

```
running-difference ( Qty )
```

Result: For each row, returns the quantity and a running difference between the value for the current row and the preceding row.

Name	Qty	Running-Difference for name
-----	-----	-----
Smith	7	NULL
Smith	3	-4
Smith	6	3
Smith	4	-2
Wong	3	-1
Wong	5	2

running-maximum

Returns the running maximum by row (including the current row) for a set of values. The "<for-option>" defines the scope of the function. The "at" option defines the level of aggregation and can be used only in the context of relational datasources.

Syntax

```
running-maximum ( numeric_expression [ at expression { , expression } ]
[ <for-option> ] [ prefilter ] )
running-maximum ( numeric_expression [ <for-option> ] [ prefilter ] )
<for-option> ::= for expression { , expression }|for report|auto
```

Example

```
running-maximum ( Qty )
```

Result: For each row, returns the quantity and a running maximum of the current and previous rows.

Name	Qty	Max	Running-Maximum (Qty) for name
-----	-----	-----	-----
Smith	2	7	2
Smith	3	7	3
Smith	6	7	6
Smith	7	7	7
Wong	3	5	3
Wong	5	5	5

running-minimum

Returns the running minimum by row (including the current row) for a set of values. The "<for-option>" defines the scope of the function. The "at" option defines the level of aggregation and can be used only in the context of relational datasources.

Syntax

```
running-minimum ( numeric_expression [ at expression { , expression } ]
[ <for-option> ] [ prefilter ] )
running-minimum ( numeric_expression [ <for-option> ] [ prefilter ] )
<for-option> ::= for expression { , expression }|for report|auto
```

Example

```
running-minimum ( Qty )
```

Result: For each row, returns the quantity and a running minimum of the current and previous rows.

Name	Qty	Min	Running-Minimum (Qty) for name
-----	-----	-----	-----
Smith	7	2	7
Smith	3	2	3
Smith	6	2	3
Smith	2	2	2
Wong	4	3	4
Wong	5	3	4

running-total

Returns a running total by row (including the current row) for a set of values. The "<for-option>" defines the scope of the function. The "at" option defines the level of aggregation and can be used only in the context of relational datasources.

Syntax

```
running-total ( numeric_expression [ at expression { , expression } ]
[ <for-option> ] [ prefilter ] )
running-total ( numeric_expression [ <for-option> ] [ prefilter ] )
<for-option> ::= for expression { , expression }|for report|auto
```

Example

```
running-total ( Qty )
```

Result: For each row, returns the quantity and a running total of the current and previous rows.

Name	Qty	Total	Running-Total (Qty) for name
-----	-----	-----	-----
Smith	2	18	2
Smith	3	18	5
Smith	6	18	11
Smith	7	18	18
Wong	3	12	3
Wong	5	12	8

standard-deviation

Returns the standard deviation of selected data items.

Syntax

```
standard-deviation ( expression [ auto ] )
standard-deviation ( expression for [ all|any ] expression { , expression } )
standard-deviation ( expression for report )
```

Example

```
standard-deviation ( ProductCost )
```

Result: Returns a value indicating the deviation between product costs and the average product cost.

standard-deviation-pop

Computes the population standard deviation and returns the square root of the population variance.

Syntax

```
standard-deviation-pop ( expression [ auto ] )
standard-deviation-pop ( expression for [ all|any ] expression
{ , expression } )
standard-deviation-pop ( expression for report )
```

Example

```
standard-deviation-pop ( ProductCost )
```

Result: Returns a value of the square root of the population variance.

total

Returns the total value of selected data items. Distinct is an alternative expression that is compatible with earlier versions of the product. This function appears in the Budget vs. Actual sample report in the GO Data Warehouse (analysis) package.

Syntax

```
total ( [ distinct ] expression [ auto ] )
total ( [ distinct ] expression for [ all|any ] expression { , expression } )
total ( [ distinct ] expression for report )
```

Example

```
total ( Sales )
```

Result: Returns the total value of all Sales values.

variance

Returns the variance of selected data items.

Syntax

```
variance ( expression [ auto ] )  
variance ( expression for [ all|any ] expression { , expression } )  
variance ( expression for report )
```

Example

```
variance ( Product Cost )
```

Result: Returns a value indicating how widely product costs vary from the average product cost.

variance-pop

Returns the population variance of a set of numbers after discarding the nulls in this set.

Syntax

```
variance-pop ( expression [ auto ] )  
variance-pop ( expression for [ all|any ] expression { , expression } )  
variance-pop ( expression for report )
```

Example

```
variance-pop ( Qty )
```

Result: For each row, returns the population variance of a set of numbers after discarding the nulls in this set.

Member Summaries

This list contains predefined functions that return either a single summary value for a set of members or a different summary value for each member of a set of members.

aggregate

Returns a calculated value using the appropriate aggregation function based on the aggregation type of the expression.

Syntax

```
aggregate ( < currentMeasure|numeric_expression > within set set_expression )  
aggregate ( < currentMeasure|numeric_expression > within < detail|aggregate >  
expression )
```

average

Returns the average value of the selected data items.

Syntax

```
average ( < currentMeasure|numeric_expression > within set set_expression )
average ( < currentMeasure|numeric_expression > within < detail|aggregate >
expression )
```

Example

```
average ( Sales )
```

Result: Returns the average of all Sales values.

count

Returns the number of selected data items excluding null values.

Syntax

```
count ( < currentMeasure|numeric_expression > within set set_expression )
count ( < currentMeasure|numeric_expression > within < detail|aggregate >
expression )
```

Example

```
count ( Sales )
```

Result: Returns the total number of entries under Sales.

maximum

Returns the maximum value of selected data items.

Syntax

```
maximum ( < currentMeasure|numeric_expression > within set set_expression )
maximum ( < currentMeasure|numeric_expression > within < detail|aggregate >
expression )
```

Example

```
maximum ( Sales )
```

Result: Returns the maximum value out of all Sales values.

median

Returns the median value of selected data items.

Syntax

```
median ( < currentMeasure|numeric_expression > within set set_expression )
median ( < currentMeasure|numeric_expression > within < detail|aggregate >
expression )
```

minimum

Returns the minimum value of selected data items.

Syntax

```
minimum ( < currentMeasure|numeric_expression > within set set_expression )
minimum ( < currentMeasure|numeric_expression > within < detail|aggregate >
expression )
```

Example

```
minimum ( Sales )
```

Result: Returns the minimum value out of all Sales values.

percentage

Returns the percent of the total value for the selected data items.

Syntax

```
percentage ( numeric_expression [ tuple member_expression { , member_expression } ] within set set_expression )
```

Example

```
percentage ( [gosales].[sales measures].[quantity] tuple [gosales].[Staff].[].[department] -> [West] within set children ( [gosales].[Staff].[].[Staff] ) )
```

percentile

Returns a value, on a scale from 0 to 100, that indicates the percent of a distribution that is equal to or below the selected data items.

Syntax

```
percentile ( numeric_expression [ tuple member_expression { , member_expression } ] within set set_expression )
```

quantile

Returns the rank of a value for the specified range. It returns integers to represent any range of ranks, such as 1 (highest) to 100 (lowest).

Syntax

```
quantile ( numeric_expression , numeric_expression [ tuple member_expression { , member_expression } ] within set set_expression )
```

quartile

Returns the rank of a value, represented as integers from 1 (highest) to 4 (lowest), relative to a group of values.

Syntax

```
quartile ( numeric_expression [ tuple member_expression { , member_expression } ] within set set_expression )
```

rank

Returns the rank value of the selected data items. The type of ranking returned (Olympic, dense, or serial) is data source dependent. The sort order is optional; DESC is assumed by default.

Syntax

```
rank ( numeric_expression [ ASC|DESC ] [ tuple member_expression { , member_expression } ] within set set_expression )
```

Example

```
rank ( [gosales].[sales measures].[quantity] tuple [gosales].[Staff].[].
[department] -> [West] within set children ( [gosales].[Staff].[].[Staff] ) )
```

standard-deviation

Returns the standard deviation of the selected data items.

Syntax

```
standard-deviation ( < currentMeasure|numeric_expression > within set
set_expression )
standard-deviation ( < currentMeasure|numeric_expression > within <
detail|aggregate > expression )
```

standard-deviation-pop

Returns the standard deviation population of the selected data items.

Syntax

```
standard-deviation-pop ( < currentMeasure|numeric_expression > within set
set_expression )
standard-deviation-pop ( < currentMeasure|numeric_expression > within <
detail|aggregate > expression )
```

total

Returns the total value of the selected data items.

Syntax

```
total ( < currentMeasure|numeric_expression > within set set_expression )
total ( < currentMeasure|numeric_expression > within < detail|aggregate >
expression )
```

variance

Returns the variance of the selected data items.

Syntax

```
variance ( < currentMeasure|numeric_expression > within set set_expression )
variance ( < currentMeasure|numeric_expression > within < detail|aggregate >
expression )
```

variance-pop

Returns the variance population of the selected data items.

Syntax

```
variance-pop ( < currentMeasure|numeric_expression > within set set_expression )
variance-pop ( < currentMeasure|numeric_expression > within < detail|aggregate
> expression )
```

Constants

A constant is a fixed value that you can use in an expression.

date

Inserts the current system date.

date-time

Inserts the current system date and time.

time with time zone

Inserts a zero time with time zone.

timestamp with time zone

Inserts an example of a timestamp with time zone.

interval

Inserts a zero interval: 000 00:00:00.000.

interval year

Inserts a zero year interval: 0 year.

interval month

Inserts a zero month interval: 0 month.

interval year to month

Inserts a zero year to month interval: 0000-00 year to month.

interval day

Inserts a zero day interval: 0 day.

interval hour

Inserts a zero hour interval: 0 hour.

interval minute

Inserts a zero minute interval: 0 minute.

interval second

Inserts a zero second interval: 0 second.

interval day to hour

Inserts a zero day to hour interval: 0 00 day to hour.

interval day to minute

Inserts a zero day to minute interval: 0 00:00 day to minute.

interval day to second

Inserts a zero day to second interval: 0 00:00:00.000000000 day to second.

interval hour to minute

Inserts a zero hour to minute interval: 00:00 hour to minute.

interval hour to second

Inserts a zero hour to second interval: 00:00:00.000000000 hour to second.

interval minute to second

Inserts a zero minute to second interval: 00:00.000000000 minute to second.

null

Inserts "null" if the expression conditions are not met.

number

Inserts the number 0, which can be replaced with a new numeric value.

string

Inserts an empty string as two single quotation marks between which you can type a string.

time

Inserts the current system time.

Constructs

This list contains constructs and templates that can be used to create an expression. Templates combine multiple functions into a group. For example, the search case template includes the case, when, else, and end functions.

if then else

This construct is the template for an if...then...else statement. This construct appears in the Top 10 Retailers for 2005 sample report in the GO Data Warehouse (analysis) package.

Syntax

```
IF ([Country] = 'Canada') THEN ([List Price] * 0.60) ELSE ([List Price])
```

in_range

This is the template for an in_range expression.

Syntax

```
[code] IN_RANGE { :30 , 40, 50, 999: }
```

Example 1

```
[code] IN_RANGE { 5 }
```

Result: This is equivalent to [code] = 5.

Example 2

```
[code] IN_RANGE { 5: }
```

Result: This is equivalent to `[code] >= 5`.

Example 3

```
[code] IN_RANGE { :5 }
```

Result: This is equivalent to `[code] <= 5`.

Example 4

```
[code] IN_RANGE { 5:10 }
```

Result: This is equivalent to `([code] >= 5 and [code] <= 10)`.

Example 5

```
[code] IN_RANGE { :5,10,20: }
```

Result: This is equivalent to `([code] <= 5 or [code] = 10 or [code] >= 20)`.

search case

This construct is the template for a search case, including the case, when, else, and end functions.

Syntax

```
CASE WHEN [Country] = 'Canada' THEN ([List Price] * 0.60) WHEN [CountryCode] >
  100 THEN [List Price] * 0.80
ELSE [List Price] END
```

simple case

This construct is the template for a simple case, including the case, when, else, and end functions.

Syntax

```
CASE [Country] WHEN 'Canada' THEN ([List Price] * 0.60) WHEN 'Australia' THEN
[List Price] * 0.80
ELSE [List Price] END
```

Business Date/Time Functions

This list contains business functions for performing date and time calculations.

_add_days

Returns the date or datetime, depending on the format of "date_expression", that results from adding "integer_expression" days to "date_expression".

Syntax

```
_add_days ( date_expression, integer_expression )
```

Example 1

```
_add_days ( 2002-04-30 , 1 )
```

Result: 2002-05-01

Example 2

```
_add_days ( 2002-04-30 12:10:10.000, 1 )
```


Result: 2002-05-01 12:10:10.000

Example 3

```
_add_days ( 2002-04-30 00:00:00.000, 1/24 )
```

Note that the second argument is not a whole number. This is supported by some database technologies and increments the time portion.

Result: 2002-04-30 01:00:00.000

_add_months

Returns the date or datetime, depending on the format of "date_expression", that results from the addition of "integer_expression" months to "date_expression".

Syntax

```
_add_months ( date_expression, integer_expression )
```

Example 1

```
_add_months ( 2002-04-30 , 1 )
```

Result: 2002-05-30

Example 2

```
_add_months ( 2002-04-30 12:10:10.000, 1 )
```

Result: 2002-05-30 12:10:10.000

_add_years

Returns the date or datetime, depending on the format of "date_expression", that results from the addition of "integer_expression" years to "date_expression".

Syntax

```
_add_years ( date_expression, integer_expression )
```

Example 1

```
_add_years ( 2002-04-30 , 1 )
```

Result: 2003-04-30

Example 2

```
_add_years ( 2002-04-30 12:10:10.000 , 1 )
```

Result: 2003-04-30 12:10:10.000

_age

Returns a number that is obtained from subtracting "date_expression" from today's date. The returned value has the form YYYYMMDD, where YYYY represents the number of years, MM represents the number of months, and DD represents the number of days.

Syntax

```
_age ( date_expression )
```

Example

```
_age ( 1990-04-30 ) (if today's date is 2003-02-05)
```

Result: 120906, meaning 12 years, 9 months, and 6 days.

_day_of_week

Returns the day of week (1 to 7), where 1 is the first day of the week as indicated by the second parameter (1 to 7, 1 being Monday and 7 being Sunday). Note that in ISO 8601 standard, a week begins with Monday being day 1.

Syntax

```
_day_of_week ( date_expression, integer )
```

Example

```
_day_of_week ( 2003-01-01 , 1 )
```

Result: 3

_day_of_year

Returns the day of year (1 to 366) in "date_expression". Also known as Julian day.

Syntax

```
_day_of_year ( date_expression )
```

Example

```
_day_of_year ( 2003-03-01 )
```

Result: 61

_days_between

Returns a positive or negative number representing the number of days between "date_expression1" and "date_expression2". If "date_expression1" < "date_expression2", then the result will be a negative number.

Syntax

```
_days_between ( date_expression1 , date_expression2 )
```

Example

```
_days_between ( 2002-04-30 , 2002-06-21 )
```

Result: -52

_days_to_end_of_month

Returns a number representing the number of days remaining in the month represented by "date_expression".

Syntax

```
_days_to_end_of_month ( date_expression )
```

Example

```
_days_to_end_of_month ( 2002-04-20 14:30:22.123 )
```

Result: 10

_first_of_month

Returns a date or datetime, depending on the argument, by converting "date_expression" to a date with the same year and month but with the day set to 1.

Syntax

```
_first_of_month ( date_expression )
```

Example 1

```
_first_of_month ( 2002-04-20 )
```

Result: 2002-04-01

Example 2

```
_first_of_month ( 2002-04-20 12:10:10.000 )
```

Result: 2002-04-01 12:10:10.000

_last_of_month

Returns a date or datetime, depending on the argument, that is the last day of the month represented by "date_expression".

Syntax

```
_last_of_month ( date_expression )
```

Example 1

```
_last_of_month ( 2002-01-14 )
```

Result: 2002-01-31

Example 2

```
_last_of_month ( 2002-01-14 12:10:10.000 )
```

Result: 2002-01-31 12:10:10.000

_make_timestamp

Returns a timestamp constructed from "integer_expression1" (the year), "integer_expression2" (the month), and "integer_expression3" (the day). The time portion defaults to 00:00:00.000 .

Syntax

```
_make_timestamp ( integer_expression1, integer_expression2, integer_expression3 )
```

Example

```
_make_timestamp ( 2002 , 01 , 14 )
```

Result: 2002-01-14 00:00:00.000

`_months_between`

Returns a positive or negative integer number representing the number of months between "date_expression1" and "date_expression2". If "date_expression1" is earlier than "date_expression2", then a negative number is returned.

Syntax

```
_months_between ( date_expression1, date_expression2 )
```

Example

```
_months_between ( 2002-04-03 , 2002-01-30 )
```

Result: 2

`_week_of_year`

Returns the number of the week of the year of "date_expression" according to the ISO 8601 standard. Week 1 of the year is the first week of the year to contain a Thursday, which is equivalent to the first week containing January 4th. A week starts on Monday (day 1) and ends on Sunday (day 7).

Syntax

```
_week_of_year ( date_expression )
```

Example

```
_week_of_year ( 2003-01-01 )
```

Result: 1

`_years_between`

Returns a positive or negative integer number representing the number of years between "date_expression1" and "date_expression2". If "date_expression1" < "date_expression2" then a negative value is returned.

Syntax

```
_years_between ( date_expression1, date_expression2 )
```

Example

```
_years_between ( 2003-01-30 , 2001-04-03 )
```

Result: 1

`_ymdint_between`

Returns a number representing the difference between "date_expression1" and "date_expression2". The returned value has the form YYYYMMDD, where YYYY represents the number of years, MM represents the number of months, and DD represents the number of days.

Syntax

```
_ymdint_between ( date_expression1 , date_expression2 )
```

Example

```
_ymdint_between ( 1990-04-30 , 2003-02-05 )
```

Result: 120906, meaning 12 years, 9 months and 6 days.

Block Functions

This list contains functions used to access members of a set, usually in the context of Analysis Studio.

_firstFromSet

Returns the first members found in the set up to "numeric_expression_maximum" + "numeric_expression_overflow". If "numeric_expression_maximum" + "numeric_expression_overflow" is exceeded, then only the maximum number of members are returned. For a set that has only a few members more than the specified numeric_expression_maximum, the numeric_expression_overflow allows the small set of extra members to be included. If the set has more members than the overflow allows, then only the numeric_expression_maximum members will be returned.

Syntax

```
_firstFromSet ( set_expression , numeric_expression_maximum , numeric_expression_overflow )
```

Example 1

```
_firstFromSet ( [great_outdoors_company].[Products].[Products].[Product line] , 2 , 8 )
```

Result: Returns the five members in the Product line set. The first two members are returned within the maximum and the following three members are returned as the overflow.

```
Camping Equipment
Golf Equipment
Mountaineering Equipment
Outdoor Protection
Personal Accessories
```

Example 2

```
_firstFromSet ( [great_outdoors_company].[Products].[Products].[Product line] , 2 , 2 )
```

Result: Camping Equipment, Golf Equipment

_remainderSet

Returns the set containing "member_expression" when the size of "set_expression" is greater than "numeric_expression"; i.e., a new member will be generated if the number of members in "set_expression" is larger than the specified "numeric_expression".

Syntax

```
_remainderSet ( member_expression, set_expression , numeric_expression )
```

Example

```
_remainderSet ( member ( aggregate ( currentMeasure within set [great_outdoors_company].[Products].[Products].[Product line] ) , 'Product Aggregate' , 'Product Aggregate' , [great_outdoors_company].[Products].[Products] ) , [great_outdoors_company].[Products].[Products].[Product line] , 1 )
```

Result: Quantity sold for Product Aggregate

Macro Functions

This list contains functions that can be used within a macro. A macro may contain one or more macro functions. A macro is delimited by a number sign (#) at the beginning and at the end. Everything between the number signs is treated as a macro expression and is executed at run time. For macro functions that accept expressions of datatype timestamp with time zone as arguments, the accepted format is 'yyyy-mm-dd hh:mm:ss[.ff]+hh:mm' where fractional seconds are optional and can be represented by 1 to 9 digits. In lieu of a space separating the date portion to the time portion, the character 'T' is also accepted. Also, in lieu of the time zone '+hh:mm', the character 'Z' is accepted and will be processed internally as '+00:00'. The macro functions that return expressions of datatype timestamp with time zone return 9 digits by default for their fractional seconds. The macro function timestampMask () can be used to trim the output if required.

+

Concatenates two strings.

Syntax

value1 + value2

Example

```
# '{ ' + $runLocale + ' }' #
```

Result: {en-us}

_add_days

Returns the timestamp with time zone (as a string) that results from adding "integer_expression" number of days to "string_expression", where "string_expression" represents a timestamp with time zone.

Syntax

```
_add_days ( string_expression , integer_expression )
```

Example 1

```
# _add_days ( '2005-11-01 12:00:00.000-05:00' , -1 ) #
```

Result: 2005-10-31 12:00:00.000000000-05:00

Example 2

```
# _add_days ( $current_timestamp , 1 ) #
```

Result: 2005-11-02 12:00:00.000000000-05:00

Example 3

```
# timestampMask ( _add_days ( $current_timestamp , 1 ) , 'yyyy-mm-dd' ) #
```

Result: 2005-11-02

_add_months

Returns the timestamp with time zone (as a string) that results from adding "integer_expression" number of months to "string_expression", where "string_expression" represents a timestamp with time zone.

Syntax

```
_add_months ( string_expression , integer_expression )
```

Example 1

```
# _add_months ( '2005-11-01 12:00:00.000-05:00' , -1 ) #
```

Result: 2005-10-01 12:00:00.000000000-05:00

Example 2

```
# _add_months ( $current_timestamp , 1 ) #
```

Result: 2005-12-01 12:00:00.000000000-05:00

Example 3

```
# timestampMask ( _add_months ( $current_timestamp , 1 ) , 'yyyy-mm-dd' ) #
```

Result: 2005-12-01

_add_years

Returns the timestamp with time zone (as a string) that results from adding "integer_expression" number of years to "string_expression", where "string_expression" represents a timestamp with time zone.

Syntax

```
_add_years ( string_expression , integer_expression )
```

Example 1

```
# _add_years ( '2005-11-01 12:00:00.000-05:00' , -1 ) #
```

Result: 2004-11-01 12:00:00.000000000-05:00

Example 2

```
# _add_years ( $current_timestamp , 1 ) #
```

Result: 2006-11-01 12:00:00.000000000-05:00

Example 3

```
# timestampMask ( _add_years ( $current_timestamp , 1 ) , 'yyyy-mm-dd' ) #
```

Result: 2006-11-01

array

Constructs an array out of the list of parameters.

Syntax

```
array ( string_expression|array_expression { , string_expression|array_expression } )
```

Example

```
# csv ( array ( 'a1' , array ( 'x1' , 'x2' ) , 'a2' ) ) #
```

Result: 'a1' , 'x1' , 'x2' , 'a2'

csv

Constructs a string from the elements of the array where the values are separated by commas. Optionally, the separator and quote strings can be specified. The default separator is a comma (,) and the default quote character is a single quote (').

Syntax

```
csv ( array_expression [ , separator_string [ , quote_string ] ] )
```

Example

```
# csv ( array ( 'a1' , 'a2' ) ) #
```

Result: 'a1', 'a2'

dq

Surrounds "string_expression" with double quotes.

Syntax

```
dq ( string_expression )
```

Example

```
# dq ( 'zero' ) #
```

Result: "zero"

getConfigurationEntry

Get an entry from the IBM® Cognos® configuration file. The force_decode_flag is optional and must be one of: 'true', '1', 1, 'false', '0', 0. The default is 'true'. When true, the value of the configuration entry will be decrypted into plain text if it is encrypted.

Syntax

```
getConfigurationEntry ( entry_string , force_decode_flag )
```

Example

```
# getConfigurationEntry ( 'serverLocale' ) #
```

Result: en

grep

Searches for and returns elements of an array that match the pattern specified in "pattern_string".

Syntax

```
grep ( pattern_string , array_expression )
```

Example

```
# csv ( grep ( 's' , array ( 'as', 'an', 'arts' ) ) ) #
```

Result: 'as', 'arts'

`_first_of_month`

Returns a timestamp with time zone (as a string) by converting the day value in "string_expression" to 1, where "string_expression" is a timestamp with time zone.

Syntax

```
_first_of_month ( string_expression )
```

Example 1

```
# _first_of_month ( '2005-11-11 12:00:00.000-05:00' ) #
```

Result: 2005-11-01 12:00:00.000000000-05:00

Example 2

```
# timestampMask ( _first_of_month ( '2005-11-11 12:00:00.000-05:00' ) , 'yyyymmdd' ) #
```

Result: 20051101

`_last_of_month`

Returns a timestamp with time zone (as a string) that is the last day of the month represented by "string_expression", where "string_expression" is a timestamp with time zone.

Syntax

```
_last_of_month ( string_expression )
```

Example 1

```
# _last_of_month ( '2005-11-11 12:00:00.000-05:00' ) #
```

Result: 2005-11-30 12:00:00.000000000-05:00

Example 2

```
# timestampMask ( _last_of_month ( '2005-11-11 12:00:00.000-05:00' ) , 'yyyy-mm-dd' ) #
```

Result: 2005-11-30

`join`

Joins the elements of an array using "separator_string".

Syntax

```
join ( separator_string , array_expression )
```

Example

```
# sq ( join ( ' || ' , array ( 'as', 'an', 'arts' ) ) ) #
```

Result: 'as || an || arts'

prompt

Prompts the user for a single value. Only "prompt_name" is required. The datatype defaults to "string" when it is not specified. The prompt is optional when "defaultText" is specified. "Text", when specified, will precede the value. "QueryItem" can be specified to take advantage of the prompt information properties of "queryItem". "Trailing_text", when specified, will be appended to the value.

Syntax

```
prompt ( prompt_name , datatype , defaultText , text , queryItem , trailing_text )
```

Example 1

```
select . . . where COUNTRY_MULTILINGUAL.COUNTRY_CODE > #prompt ( 'Starting  
CountryCode' , 'integer' , '10' ) #
```

Result: select . . . where COUNTRY_MULTILINGUAL.COUNTRY_CODE > 10

Example 2

```
[gosales].[COUNTRY].[COUNTRY] = # prompt  
( 'countryPrompt' , 'string' , '''Canada''' ) #
```

Result: [gosales].[COUNTRY].[COUNTRY] = 'Canada'

Notes

- The "defaultText" parameter must be specified such that it is literally valid in the context of the macro since no formatting takes place on this value. The default string "'Canada'" in Example 2 is specified as a string using single quotes, in which the embedded single quotes are doubled up, thus 3 quotes. This results in the string being properly displayed within single quotes in the expression. As a general rule for the string datatype, "defaultText" should always be specified like this, except in the context of a stored procedure parameter. For "defaultText" of types 'date' or 'datetime', a special format should be used in the context of SQL. Examples of these formats are 'DATE "2001-12-25"' and 'DATETIME "2001-12-25 12:00:00"'. In all other contexts, use the date/datetime without the keyword and escaped single quotes (e.g., '2001-12-25').

promptmany

Prompts the user for one or more values. Only "prompt_name" is required. The datatype defaults to string when it is not specified. The prompt is optional when "defaultText" is specified. "Text", when specified, will precede the list of values. "QueryItem" can be specified to take advantage of the prompt information properties of "queryItem". "Trailing_text", when specified, will be appended to the list of values.

Syntax

```
promptmany ( prompt_name , datatype , defaultText , text , queryItem , trailing_text )
```

Example 1

```
select . . . where COUNTRY_MULTILINGUAL.COUNTRY in ( # promptmany
( 'CountryName' ) # )
```

Result: select . . . where COUNTRY_MULTILINGUAL.COUNTRY_CODE in ('Canada' , 'The Netherlands' , 'Russia')

Example 2

```
select . . . from gosales.gosales.dbo.COUNTRY_MULTILINGUAL COUNTRY_
MULTILINGUAL , gosales.gosales.dbo.COUNTRY XX where COUNTRY_MULTILINGUAL.COUNTRY_
CODE = XX.COUNTRY_CODE # promptmany ( 'Selected CountryCodes' , 'integer' , '
' , ' ' and COUNTRY_MULTILINGUAL.COUNTRY_CODE in ( ' , ' ' , ' ) ' ) #
```

Result: select . . . from gosales.gosales.dbo.COUNTRY_MULTILINGUAL COUNTRY_MULTI-
LINGUAL , gosales.gosales.dbo.COUNTRY XX where COUNTRY_MULTILINGUAL.COUNTRY_
CODE = XX.COUNTRY_CODE and COUNTRY_MULTILINGUAL.COUNTRY_CODE in
('Canada' , 'The Netherlands' , 'Russia')

sb

Surrounds "string_expression" with square brackets.

Syntax

```
sb ( string_expression )
```

Example

```
# sb ( 'abc' ) #
```

Result: [abc]

sq

Surrounds "string_expression" with single quotes.

Syntax

```
sq ( string_expression )
```

Example

```
# sq ( 'zero' ) #
```

Result: 'zero'

sort

Sorts the elements of the array in alphabetical order. Duplicates are retained.

Syntax

```
sort ( array_expression )
```

Example

```
# csv ( sort ( array ( 's3', 'a', 'x' ) ) ) #
```

Result: 'a', 's3', 'x'

split

Splits a string or string elements of the array into separate elements.

Syntax

```
split ( pattern_string, string_expression|array_expression )
```

Example 1

```
# csv ( split ( '::', 'ab=c::de=f::gh=i' ) ) #
```

Result: 'ab=c', 'de=f', 'gh=i'

Example 2

```
# csv ( split ( '=', split ( '::', 'ab=c::de=f::gh=i' ) ) ) #
```

Result: 'ab', 'c', 'de', 'f', 'gh', 'i'

substitute

Searches for a pattern in a string or in the string elements of an array and substitutes the first occurrence of "pattern_string" with "replacement_string".

Syntax

```
substitute ( pattern_string, replacement_string, string_expression|array_expression )
```

Example 1

```
# sq ( substitute ( '^cn=', '***', 'cn=help' ) ) #
```

Result: '***help'

Example 2

```
# csv ( substitute ( '^cn=', '***', array ( 'cn=help' , 'acn=5' ) ) ) #
```

Result: '***help', 'acn=5'

Example 3

```
# csv ( substitute ( 'cn=', '', array ( 'cn=help' , 'acn=5' ) ) ) #
```

Result: 'help', 'a5'

timestampMask

Returns "string_expression1", representing a timestamp with time zone, trimmed to the format specified in "string_expression2". The format in "string_expression2" must be one of the following: 'yyyy', 'mm', 'dd', 'yyyy-mm', 'yyyymm', 'yyyy-mm-dd', 'yyyymmdd', 'yyyy-mm-dd hh:mm:ss', 'yyyy-mm-dd hh:mm:ss+hh:mm', 'yyyy-mm-dd hh:mm:ss.ff3', 'yyyy-mm-dd hh:mm:ss.ff3+hh:mm', 'yyyy-mm-ddThh:mm:ss', 'yyyy-mm-ddThh:mm:ss+hh:mm', 'yyyy-mm-ddThh:mm:ss.ff3+hh:mm', or 'yyyy-mm-ddThh:mm:ss.ff3+hh:mm'. The macro functions that return a string representation

of a timestamp with time zone show a precision of 9 digits for the fractional part of the seconds by default. The format options allow this to be trimmed down to a precision of 3 or 0.

Syntax

```
timestampMask ( string_expression1 , string_expression2 )
```

Example 1

```
# timestampMask ( $current_timestamp , 'yyyy-dd-mm' ) #
```

Result: 2005-11-01

Example 2

```
# timestampMask ( '2005-11-01 12:00:00.000-05:00' , 'yyyy-mm-dd  
hh:mm:ss+hh:mm' ) #
```

Result: 2005-11-01 12:00:00-05:00

Example 3

```
# timestampMask ( '2005-11-01 12:00:00.123456789-05:00' , 'yyyy-mm-ddThh:mm:  
ss+hh:mm.ff3+hh:mm' ) #
```

Result: 2005-11-01T12:00:00.123-05:00

toLocal

Returns the string representing a timestamp with time zone resulting from adjusting "string_expression" to the time zone of the operating system. Note that the macro function timestampMask () can be used to trim the output.

Syntax

```
toLocal ( string_expression )
```

Example 1

```
# toLocal ( '2005-11-01 17:00:00.000-00:00' ) # where OS local time zone is  
-05:00
```

Result: 2005-11-01 12:00:00.000000000-05:00

Example 2

```
# timestampMask ( toLocal ( '2005-11-01 17:00:00.000-00:00' ) , 'yyyy-mm-dd  
hh:mm:ss+hh:mm' ) # where OS local time zone is -05:00
```

Result: 2005-11-01 12:00:00-05:00

Example 3

```
# toLocal ( '2005-11-01 13:30:00.000-03:30' ) # where OS local time zone is  
-05:00
```

Result: 2005-11-01 12:00:00.000000000-05:00

toUTC

Returns the string representing a timestamp with time zone resulting from adjusting "string_expression" to the zero-point reference UTC time zone, also known as GMT time. Note that the macro function timestampMask () can be used to trim the output.

Syntax

```
toUTC ( string_expression )
```

Example 1

```
# toUTC ( '2005-11-01 12:00:00.000-05:00' ) #
```

Result: 2005-11-01 17:00:00.000000000-00:00

Example 2

```
# timestampMask( toUTC ( '2005-11-01 12:00:00.000-05:00' ) , 'yyyy-mm-dd hh:mm:ss.ff3+hh:mm' ) #
```

Result: 2005-11-01 17:00:00.000-00:00

Example 3

```
# toUTC ( $current_timestamp ) #
```

Result: 2005-11-01 17:00:00.000000000-00:00

unique

Removes duplicate entries from the array. The order of the elements is retained.

Syntax

```
unique ( array_expression )
```

Example

```
# csv ( unique ( array ( 's3', 'a', 's3', 'x' ) ) ) #
```

Result: 's3', 'a', 'x'

urlencode

URL-encodes the passed argument. This function is useful when specifying XML connection strings.

Syntax

```
urlencode ( prompt ( 'userValue' ) )
```

Example

```
urlencode ( prompt ( 'some_val' ) )
```

Result: %27testValue%27

CSVIdentityName

Uses the identity information of the current authenticated user to look up values in the specified parameter map. Each individual piece of the user's identity (account name, group names, role names) is used as a key into the map. The unique list of values that is retrieved from the parameter map is then returned as a string, where each value is surrounded by single quotes and where multiple values are separated by commas.

Syntax

```
CSVIdentityName ( %parameter_map_name [ , separator_string ] )
```

Example

```
# CSVIdentityName ( %security_clearance_level_map ) #
```

Result: 'level_500', 'level_501', 'level_700'

CSVIdentityNameList

Returns the pieces of the user's identity (account name, group names, role names) as a list of strings. The unique list of values is returned as a string, where each value is surrounded by single quotes and where multiple values are separated by commas.

Syntax

```
CSVIdentityNameList ( [ separator_string ] )
```

Example

```
# CSVIdentityNameList ( ) #
```

Result: 'Everyone', 'Report Administrators', 'Query User'

CAMPassport

Returns the Cognos® Access Manager passport.

Syntax

```
CAMPassport ( )
```

Example

```
# CAMPassport ( ) #
```

Result: 111:98812d62-4fd4-037b-4354-26414cf7ebef:3677162321

CAMIDList

Returns the pieces of the user's Cognos® Access Manager ID (CAMID), such as account name, group names, or role names, as a list of values separated by commas.

Syntax

```
CAMIDList ( [ separator_string ] )
```

Example

```
#CAMIDList ( ) #
```

Result: CAMID ("::Everyone"), CAMID (":Authors"), CAMID (":Query Users"), CAMID (":Consumers"), CAMID (":Metrics Authors")

CAMIDListForType

Returns an array of the user's Cognos® Access Manager IDs (CAMIDs) based on the identity type (account, group, or role). CAMIDListForType can be used with the macro functions csv or join.

Syntax

```
CAMIDListForType ( identity type )
```

Example

```
[qs].[userRole] in ( # csv ( CAMIDListForType ( 'role' ) ) # )
```

Result: [qs].[userRole] in ('CAMID ("::System Administrators") ', 'CAMID (":Authors")')

Common Functions

abs

Returns the absolute value of "numeric_expression". Negative values are returned as positive values.

Syntax

```
abs ( numeric_expression )
```

Example 1

```
abs ( 15 )
```

Result: 15

Example 2

```
abs ( -15 )
```

Result: 15

cast

Converts "expression" to a specified data type. Some data types allow for a length and precision to be specified. Make sure that the target is of the appropriate type and size. The following can be used for "datatype_specification": character, varchar, char, numeric, decimal, integer, smallint, real, float, date, time, timestamp, time with time zone, timestamp with time zone, and interval. When type casting to an interval type, one of the following interval qualifiers must be specified: year, month, or year to month for the year-to-month interval datatype; day, hour, minute, second, day to hour, day to minute, day to second, hour to minute, hour to second, or minute to second for the day-to-second interval datatype. Notes: When you convert a value of type timestamp to type date, the time portion of the timestamp value is ignored. When you convert a value of type timestamp to type time, the date portion of the timestamp is ignored. When you convert a value of type date to type timestamp, the time components of the timestamp are set to zero. When you convert a value of type time to type timestamp, the date component is set to the current system date. It is invalid to convert one interval datatype to the other (for instance because the number of days in a month is variable). Note that you can specify the number of digits for the leading qualifier only, i.e. YEAR(4) TO MONTH, DAY(5). Errors will be reported if the target type and size are not compatible with the source type and size.

Syntax

```
cast ( expression , datatype_specification )
```

Example 1

```
cast ( '123' , integer )
```

Result: 123

Example 2

```
cast ( 12345 , varchar ( 10 ) )
```

Result: a string containing 12345

ceil

Returns the smallest integer that is greater than or equal to "numeric_expression".

Syntax

```
ceil ( numeric_expression )
```

ceiling

Returns the smallest integer that is greater than or equal to "numeric_expression".

Syntax

```
ceiling ( numeric_expression )
```

Example 1

```
ceiling ( 4.22 )
```

Result: 5

Example 2

```
ceiling ( -1.23 )
```

Result: -1

char_length

Returns the number of logical characters in "string_expression". The number of logical characters can be distinct from the number of bytes in some East Asian locales.

Syntax

```
char_length ( string_expression )
```

Example

```
char_length ( 'Canada' )
```

Result: 6

character_length

Returns the number of characters in "string_expression".

Syntax

```
character_length ( string_expression )
```

Example

```
character_length ( 'Canada' )
```

Result: 6

coalesce

Returns the first non-null argument (or null if all arguments are null). Requires two or more arguments in "expression_list".

Syntax

```
coalesce ( expression_list )
```

Example

```
coalesce ( [Unit price], [Unit sale price] )
```

Result: Returns the unit price, or the unit sale price if the unit price is null.

current_date

Returns a date value representing the current date of the computer that the database software runs on.

Syntax

```
current_date
```

Example

```
current_date
```

Result: 2003-03-04

current_time

Returns a time with time zone value, representing the current time of the computer that runs the database software if the database supports this function. Otherwise, it represents the current time of the computer that runs IBM® Cognos® BI software.

Syntax

```
current_time
```

Example

```
current_time
```

Result: 16:33:11+05:00

current_timestamp

Returns a datetime with time zone value, representing the current time of the computer that runs the database software if the database supports this function. Otherwise, it represents the current time of the computer that runs IBM® Cognos® BI software.

Syntax

```
current_timestamp
```

Example

```
current_timestamp
```

Result: 2003-03-03 16:40:15.535000+05:00

exp

Returns 'e' raised to the power of "numeric_expression". The constant 'e' is the base of the natural logarithm.

Syntax

```
exp ( numeric_expression )
```

Example

```
exp ( 2 )
```

Result: 7.389056

extract

Returns an integer representing the value of datepart (year, month, day, hour, minute, second) in "datetime_expression".

Syntax

```
extract ( datepart , datetime_expression )
```

Example 1

```
extract ( year , 2003-03-03 16:40:15.535 )
```

Result: 2003

Example 2

```
extract ( hour , 2003-03-03 16:40:15.535 )
```

Result: 16

floor

Returns the largest integer that is less than or equal to "numeric_expression".

Syntax

```
floor ( numeric_expression )
```

Example 1

```
floor ( 3.22 )
```

Result: 3

Example 2

```
floor ( -1.23 )
```

Result: -2

ln

Returns the natural logarithm of "numeric_expression".

Syntax

```
ln ( numeric_expression )
```

Example

```
ln ( 4 )
```

Result: 1.38629

localtime

Returns a time value, representing the current time of the computer that runs the database software.

Syntax

```
localtime
```

Example

```
localtime
```

Result: 16:33:11

localtimestamp

Returns a datetime value, representing the current timestamp of the computer that runs the database software.

Syntax

```
localtimestamp
```

Example

```
localtimestamp
```

Result: 2003-03-03 16:40:15.535000

lower

Returns "string_expression" with all uppercase characters shifted to lowercase.

Syntax

```
lower ( string_expression )
```

Example

```
lower ( 'ABCDEF' )
```

Result: abcdef

mod

Returns the remainder (modulus) of "integer_expression1" divided by "integer_expression2". "Integer_expression2" must not be zero or an exception condition is raised.

Syntax

```
mod ( integer_expression1, integer_expression2 )
```

Example

```
mod ( 20 , 3 )
```

Result: 2

nullif

Returns null if "expression1" equals "expression2", otherwise returns "expression1".

Syntax

```
nullif ( expression1, expression2 )
```

octet_length

Returns the number of bytes in "string_expression".

Syntax

```
octet_length ( string_expression )
```

Example 1

```
octet_length ( 'ABCDEF' )
```

Result: 6

Example 2

```
octet_length ( '' )
```

Result: 0

position

Returns the integer value representing the starting position of "string_expression1" in "string_expression2" or 0 when the "string_expression1" is not found.

Syntax

```
position ( string_expression1 , string_expression2 )
```

Example 1

```
position ( 'C' , 'ABCDEF' )
```

Result: 3

Example 2

```
position ( 'H' , 'ABCDEF' )
```

Result: 0

power

Returns "numeric_expression1" raised to the power "numeric_expression2". If "numeric_expression1" is negative, then "numeric_expression2" must result in an integer value.

Syntax

```
power ( numeric_expression1 , numeric_expression2 )
```

Example

```
power ( 3 , 2 )
```

Result: 9

_round

Returns "numeric_expression" rounded to "integer_expression" places to the right of the decimal point. Notes: "integer_expression" must be a non-negative integer. Rounding takes place before data formatting is applied.

Syntax

```
_round ( numeric_expression , integer_expression )
```

Example

```
_round ( 1220.42369, 2 )
```

Result: 1220.42

sqrt

Returns the square root of "numeric_expression". "Numeric_expression" must be non-negative.

Syntax

```
sqrt ( numeric_expression )
```

Example

```
sqrt ( 9 )
```

Result: 3

substring

Returns the substring of "string_expression" that starts at position "integer_expression1" for "integer_expression2" characters or to the end of "string_expression" if "integer_expression2" is omitted. The first character in "string_expression" is at position 1.

Syntax

```
substring ( string_expression , integer_expression1 [ , integer_expression2 ] )
```

Example

```
substring ( 'abcdefg' , 3 , 2 )
```

Result: cd

trim

Returns "string_expression" trimmed of leading and trailing blanks or trimmed of a certain character specified in "match_character_expression". "Both" is implicit when the first argument is not stated and blank is implicit when the second argument is not stated.

Syntax

```
trim ( [ [ trailing|leading|both ] [ match_character_expression ] , ] string_expression )
```

Example 1

```
trim ( trailing 'A' , 'ABCDEFA' )
```

Result: ABCDEF

Example 2

```
trim ( both , ' ABCDEF ' )
```

Result: ABCDEF

upper

Returns "string_expression" with all lowercase characters converted to uppercase.

Syntax

```
upper ( string_expression )
```

Example

```
upper ( 'abcdef' )
```

Result: ABCDEF

Dimensional Functions**ancestor**

Returns the ancestor of "member" at "level" or at "integer" number of levels above "member".
Note: The result is not guaranteed to be consistent when there is more than one such ancestor.

Syntax

```
ancestor ( member, level|integer )
```

Example 1

```
ancestor ( [TrailChef Water Bag] , 1 )
```

Result: Cooking Gear

Example 2

```
ancestor ( [TrailChef Water Bag] , 2 )
```

Result: Camping Equipment

Example 3

```
ancestor ( [TrailChef Water Bag] , [great_outdoors_company].[Products].  
[Products].[Product type] )
```

Result: Cooking Gear

ancestors

Returns all the ancestors of "member" at "level" or "index" distance above the member. (Most data sources support only one ancestor at a specified level. If the data source supports more than one ancestor, the result is a member set.)

Syntax

```
ancestors ( member , level|index )
```

Example 1

```
ancestors ( [TrailChef Water Bag] , 1 )
```

Result: Cooking Gear

Example 2

```
ancestors ( [TrailChef Water Bag] , 2 )
```

Result: Camping Equipment

Example 3

```
ancestors ( [TrailChef Water Bag] , [great_outdoors_company].[Products].  
[Products].[Product type] )
```

Result: Cooking Gear

bottomCount

Sorts a set according to the value of "numeric_expression" evaluated at each of the members of "set_expression" and returns the bottom "index_expression" members.

Syntax

```
bottomCount ( set_expression , index_expression , numeric_expression )
```

Example

```
bottomCount ( [great_outdoors_company].[Products].[Products].[Product  
line] , 2 , [Revenue] )
```

Result: Returns the bottom two members of the set sorted by revenue.

Product line	Revenue
-----	-----
Outdoor Protection	\$3,171,114.92
Mountaineering Equip- ment	\$20,891,350.60

bottomPercent

Sorts "numeric_expression2", evaluated at the corresponding members of "set_expression", and picks up the bottommost elements whose cumulative total is equal to or less than "numeric_expression1" percent of the total.

Syntax

```
bottomPercent ( set_expression , numeric_expression1 , numeric_expression2 )
```

Example

```
bottomPercent ( set ( [Camping Equipment] , [Golf Equipment] , [Mountaineering  
Equipment] ) , 40 , [2006] )
```

Result: For the set of Camping Equipment, Golf Equipment, and Mountaineering Equipment, returns the members whose percentage total are less than or equal to 40% for 2006.

bottomSum

Sorts "numeric_expression2", evaluated at the corresponding member of "set_expression", and picks up the bottommost elements whose cumulative total is equal to or less than "numeric_expression1".

Syntax

```
bottomSum ( set_expression , numeric_expression1 , numeric_expression2 )
```

Example

```
bottomSum ( members ( [great_outdoors_company].[Products].[Products].[Product
line] ) , 6000000 , tuple ( [2006] , [great_outdoors_company].[Measures].[Gross
profit] ) ) )
```

caption

Returns the caption values of "level", "member", or "set_expression". The caption is the string display name for an element and does not necessarily match the unique identifier used to generate the business key or member unique name (MUN) for the element. The caption is not necessarily unique; for example, the caption for a month may return the month name without further year details to make the value unique.

Syntax

```
caption ( level|member|set_expression )
```

Example 1

```
caption ( [TrailChef Water Bag] )
```

Result: TrailChef Water Bag

Example 2

```
caption ( [great_outdoors_company].[Products].[Products].[Product line] )
```

Result: Returns the caption values of the Product line set.

```
Camping Equipment
Mountaineering Equipment
Personal Accessories
Outdoor Protection
Golf Equipment
```

children

Returns the set of children of a specified member.

Syntax

```
children ( member )
```

Example

```
children ( [Camping Equipment] )
```

Result: Returns the set of children for Camping Equipment.

```
Cooking Gear
Tents
Sleeping Bags
```

Packs
Lanterns

closingPeriod

Returns the last sibling member among the descendants of a member at "level". This function is typically used with a time dimension.

Syntax

```
closingPeriod ( level [ , member ] )
```

Example 1

```
closingPeriod ( [great_outdoors_company].[Years].[Years].[Month] )
```

Result: 2006/Dec

Example 2

```
closingPeriod ( [great_outdoors_company].[Years].[Years].[Year] )
```

Result: 2006

Example 3

```
closingPeriod ( [great_outdoors_company].[Years].[Years].[Month] , [2006 Q 4] )
```

Result: 2006/Dec

cousin

Returns the child member of "member2" with the same relative position as "member1" to its parent. This function appears in the Revenue by GO Subsidiary 2005 sample report in the GO Data Warehouse (analysis) package.

Syntax

```
cousin ( member1 , member2 )
```

Example 1

```
cousin ( [Irons] , [Camping Equipment] )
```

Result: Cooking Gear

Example 2

```
cousin ( [Putters] , [Camping Equipment] )
```

Result: Sleeping Bags

completeTuple

Identifies a cell location (intersection) based on the specified members, each of which must be from a different dimension. However, completeTuple () implicitly includes the default member from all dimensions not otherwise specified in the arguments, rather than the current member. CompleteTuple will use the default measure rather than the currentMeasure in the query if the measure is not defined in the completetuple function. This function appears in the Planned Headcount sample report in the GO Data Warehouse (analysis) package.

Syntax

```
completeTuple ( member { , member } )
```

Example 1

```
completeTuple ( [Mountaineering Equipment] , [Fax] )
```

Result: The `completeTuple` does not pick up the `currentMember` by default as the tuple function does. The values in the first column are identical across each year because the default member of the Years dimension, the root member, is used rather than the current member. Likewise, the first column displays Revenue rather than Quantity Sold because the Revenue measure is the default from the Measures dimension. `CompleteTuple` will use the default measure rather than the current-Measure in the query if the measure is not defined in the `completetuple` function.

Quantity Sold	Mountaineering Sales by Fax
-----	-----
2004	\$1,220,329.38
2005	\$1,220,329.38
2006	\$1,220,329.38

Example 2

```
completeTuple ( [Mountaineering Equipment] , [Fax] , [Quantity  
sold] , currentMember ( [great_outdoors_company].[Years].[Years] ) )
```

Result: The `completeTuple` function uses the `currentMember` of the Years dimension and the Quantity sold measure.

Quantity Sold	Mountaineering Sales by Fax
-----	-----
2004	0
2005	8,746
2006	7,860

currentMember

Returns the current member of the hierarchy during an iteration. If "hierarchy" is not present in the context in which the expression is being evaluated, its default member is assumed. This function appears in the Rolling and Moving Averages interactive sample report.

Syntax

```
currentMember ( hierarchy )
```

defaultMember

Returns the default member of "hierarchy".

Syntax

```
defaultMember ( hierarchy )
```

Example 1

```
defaultMember ( [great_outdoors_company].[Products].[Products] )
```

Result: Products

Example 2

```
defaultMember ( [great_outdoors_company].[Years].[Years] )
```

Result: Year

Example 3

```
defaultMember ( hierarchy ( [great_outdoors_company].[Measures].[Quantity  
sold] ) )
```

Result: Revenue

descendants

Returns the set of descendants of "member" or "set_expression" at "level" (qualified name) or "distance" (integer 0..n) from the root. Multiple options may be specified (separated by a space) to determine which members are returned. self: Only the members at the specified level are included in the final set (this is the default behaviour in the absence of any options). before: If there are any intermediate levels between the member's level and the one specified, members from those levels are included. If the level specified is the same as the member upon which the function is applied, the member is included in the final set. beforewithmember: If there are any intermediate levels between the member's level and the one specified, members from those levels are included. The member upon which the function is applied is also included in the final set. after: If other levels exist after the specified level, members from those levels are included in the final set. This function appears in the Sales Commissions for Central Europe sample report in the GO Data Warehouse (analysis) package.

Syntax

```
descendants ( member|set_expression , level|distance [ , { self|before|  
beforewithmember|after } ] )
```

Example 1

```
descendants ( [great_outdoors_company].[Products].[Products].[Products] , [great_  
outdoors_company].[Products].[Products].[Product type] )
```

Result: Returns the set of descendants of the Products set at the Product type level. Note: [great_outdoors_company].[Products].[Products].[Products] is the root member of the Products hierarchy.

```
Cooking Gear
Sleeping Bags
Packs
Tents
...
Eyewear
Knives
Watches
```

Example 2

```
descendants ( [great_outdoors_company].[Products].[Products].[Products] , 1 )
```

Result: Returns the set of descendants of the Products set at the first level.

```
Camping Equipment
Golf Equipment
Mountaineering Equipment
Outdoor Protection
Personal Accessories
```

Example 3

```
descendants ( [great_outdoors_company].[Products].[Products].
[Products] , 3 , before )
```

Result: Returns the descendants of the Products set before the third level.

```
Camping Equipment
Cooking Gear
Sleeping Bags
Packs
Tents
...
Eyewear
Knives
Watches
```

Example 4

```
descendants ( [great_outdoors_company].[Products].[Products].
[Products] , 2 , self before )
```

Result: Returns the set of descendants of the Products set before and including the second level.

```
Camping Equipment
Cooking Gear
Sleeping Bags
Packs
Tents
...
Eyewear
Knives
Watches
```

except

Returns the members of "set_expression1" that are not also in "set_expression2". Duplicates are retained only if the optional keyword all is supplied as the third argument.

Syntax

```
except ( set_expression1 , set_expression2 [ , all ] )
```

Example

```
except ( set ( [Camping Equipment] , [Mountaineering Equipment] ) , set
( [Camping Equipment] , [Golf Equipment] ) )
```

Result: Mountaineering Equipment

filter

Returns the set resulting from filtering a specified set based on the Boolean condition. Each member is included in the result if and only if the corresponding value of "Boolean_expression" is true.

Syntax

```
filter ( set_expression , Boolean_expression )
```

Example

```
filter ( [Product line] , [Gross margin] > .30 )
```

Result: Mountaineering Equipment

firstChild

Returns the first child of "member".

Syntax

```
firstChild ( member )
```

Example 1

```
firstChild ( [By Product Lines] )
```

Result: Camping Equipment

Example 2

```
firstChild ( [Camping Equipment] )
```

Result: Cooking Gear

firstSibling

Returns the first child of the parent of "member".

Syntax

```
firstSibling ( member )
```

Example 1

```
firstSibling ( [Outdoor Protection] )
```

Result: Camping Equipment

Example 2

```
firstSibling ( [Camping Equipment] )
```

Result: Camping Equipment

_format

Associates a format with the expression. The format_keyword can be PERCENTAGE_0, PERCENTAGE_1, or PERCENTAGE_2. PERCENTAGE_1 returns a percentage with one digit to the right of the decimal point, PERCENTAGE_2 returns a percentage with two digits to the right of the decimal point, and PERCENTAGE_3 returns a percentage value out of one with three digits to the right of the decimal point (for example, 0.965).

Syntax

```
_format ( expression , format_keyword )
```

Example

```
_format ( [Unit Sale Price] / [Unit Price] , PERCENTAGE_2 )
```

Result: 75.12%

emptySet

Returns an empty member set for "hierarchy". This is most often used as a placeholder during development or with dynamic report design (either with the IBM® Cognos® Software Development Kit or via report design). By creating a data item that contains the emptyset function, it is possible to build complex expressions that can later be revised by redefining the emptyset data item.

Syntax

```
emptySet ( hierarchy )
```

Example

```
except ( [great_outdoors_company].[Products].[Products].[Product line] , emptyset  
  ( [great_outdoors_company].[Products].[Products] ) )
```

Result: Returns the Product line set and an empty set for the Products set.

```
Camping Equipment  
Golf Equipment  
Mountaineering Equipment  
Outdoor Protection  
Personal Accessories
```

generate

Evaluates "set_expression2" for each member of "set_expression1" and joins the resulting sets by union. The result retains duplicates only when the optional keyword "all" is supplied as the third argument.

Syntax

```
generate ( set_expression1 , set_expression2 [ , all ] )
```

Example

```
generate ( [Product line] , topCount ( descendants ( currentMember ( [great_  
outdoors_company].[Products].[Products] ) , [great_outdoors_company].[Products].  
[Products].[Product name] ) , 2 , [Revenue] ) )
```

Result: Returns the top two products by revenue for each product line.

head

Returns the first "index_expression" elements of "set_expression". The default for "index_expression" is 1.

Syntax

```
head ( set_expression [ , index_expression ] )
```

Example 1

```
head ( members ( [great_outdoors_company].[Products].[Products].[Product  
line] ) )
```

Result: Camping Equipment

Example 2

```
head ( members ( [great_outdoors_company].[Products].[Products].[Product  
line] ) , 2 )
```

Result: Returns the top two members of the Product line set.

```
Camping Equipment
Mountaineering Equipment
```

hierarchize

Orders the members of "set_expression" in a hierarchy. Members in a level are sorted in their natural order. This is the default ordering of the members along a dimension when no other sort conditions are specified.

Syntax

```
hierarchize ( set_expression )
```

Example

```
hierarchize ( set ( [Golf Equipment] , [Mountaineering Equipment] , [Camping Equipment] ) )
```

Result: Returns Camping Equipment, Golf Equipment, Mountaineering Equipment.

hierarchy

Returns the hierarchy that contains "level", "member", or "set_expression".

Syntax

```
hierarchy ( level|member|set_expression )
```

Example 1

```
hierarchy ( [Cooking Gear] )
```

Result: Returns every member in the hierarchy that contains Cooking Gear.

```
Products
Camping Equipment
Cooking Gear
TrailChef Water Bag
TrailChef Canteen
...
Mountain Man Extreme
Mountain Man Deluxe
```

Example 2

```
hierarchy ( [great_outdoors_company].[Products].[Products].[Product line] )
```

Result: Returns every member in the hierarchy that contains the Product line.

```
Products
Camping Equipment
Cooking Gear
TrailChef Water Bag
TrailChef Canteen
...
Mountain Man Extreme
Mountain Man Deluxe
```

item

Returns a member from the "index" location within "set_expression". The index into the set is zero based.

Syntax

```
item ( set_expression , index )
```

Example

```
item ( children ( [Camping Equipment] ) , 2 )
```

Result: Sleeping Bags

intersect

Returns the intersection of "set_expression1" and "set_expression2". The result retains duplicates only when the optional keyword "all" is supplied as the third argument.

Syntax

```
intersect ( set_expression1 , set_expression2 [ , all ] )
```

Example

```
intersect ( set ( [Camping Equipment] , [Mountaineering Equipment] ) , set  
( [Camping Equipment] , [Outdoor Protection] , ) , all )
```

Result: Camping Equipment

lag

Returns the sibling member that is "index_expression" number of positions prior to "member".

Syntax

```
lag ( member , index_expression )
```

Example 1

```
lag ( [Tents] , 1 )
```

Result: Cooking Gear

Example 2

```
lag ( [Tents] , -2 )
```

Result: Packs

lastChild

Returns the last child of a specified member.

Syntax

```
lastChild ( member )
```

Example 1

```
lastChild ( Cooking Gear )
```

Result: TrailChef Utensils

Example 2

```
lastChild ( [By Product Line] )
```

Result: Golf Equipment

lastPeriods

Returns the set of members from the same level that ends with "member". The number of members returned is the absolute value of "integer_expression". If "integer_expression" is negative, members following and including the specified member are returned. Typically used with a time dimension. This function appears in the Rolling and Moving Averages interactive sample report.

Syntax

```
lastPeriods ( integer_expression , member )
```

Example 1

```
lastPeriods ( 2 , [2006 Q 4] )
```

Result: Returns the last two members from the level that ends with 2006 Q 4.

```
2006 Q 3  
2006 Q 4
```

Example 2

```
lastPeriods ( -3 , [2006 Q 4] )
```

Result: Returns the last three members from the level that starts with 2006 Q 4.

```
2006 Q 4  
2007 Q 1  
2007 Q 2
```

lastSibling

Returns the last child of the parent of a specified member.

Syntax

```
lastSibling ( member )
```

Example

```
lastSibling ( [Camping Equipment] )
```

Result: Golf Equipment

lead

Returns the sibling member that is "index_expression" number of positions after "member". If "index_expression" is negative, returns the sibling member that is "index_expression" number of positions before "member".

Syntax

```
lead ( member , index_expression )
```

Example 1

```
lead ( [Outdoor Protection] , 1 )
```

Result: Personal Accessories

Example 2

```
lead ( [Outdoor Protection] , -2 )
```

Result: Golf Equipment

level

Returns the level of "member".

Syntax

```
level ( member )
```

Example

```
level ( [Golf Equipment] )
```

Result: Returns the members on the Golf Equipment level.

```
Camping Equipment
Mountaineering Equipment
Personal Accessories
Outdoor Protection
Golf Equipment
```

levels

Returns the level in "hierarchy" whose distance from the root is specified by "index".

Syntax

```
levels ( hierarchy , index )
```

Example 1

```
levels ( [great_outdoors_company].[Products].[Products] , 2 )
```

Result: Returns the members two levels from the root Products hierarchy.

```
Cooking Gear
Sleeping Bags
Packs
Tents
...
Irons
Putters
Woods
Golf Accessories
```

Example 2

```
levels ( [great_outdoors_company].[Products].[Products] , 1 )
```

Result: Returns the members one level from the root Products hierarchy.

```
Camping Equipment
Mountaineering Equipment
Personal Accessories
Outdoor Protection
Golf Equipment
```

linkMember

Returns the corresponding member in "level" or "hierarchy" (of the same dimension). For level-based hierarchies, a level must be specified as the second argument, and for parent-child hierarchies, a hierarchy must be specified. An exception is thrown when the second parameter does not resolve

to a hierarchy of the member's dimension. Note that calculated members are not supported as the first argument.

Syntax

```
linkMember ( member , level|hierarchy )
```

members

Returns the set of members in "hierarchy" or "level". In the case of a hierarchy, the order of the members in the result is not guaranteed. If a predictable order is required, an explicit ordering function (such as `hierarchize`) must be used.

Syntax

```
members ( hierarchy|level )
```

Example 1

```
members ( [great_outdoors_company].[Years].[Years] )
```

Result: Returns the members in Years.

Example 2

```
members ( [great_outdoors_company].[Products].[Products].[Product line] )
```

Result: Returns the members in Product line.

nextMember

Returns the next member in the "member" level.

Syntax

```
nextMember ( member )
```

Example

```
nextMember ( [Outdoor Protection] )
```

Result: Golf Equipment

openingPeriod

Returns the first sibling member among the descendants of a member at "level". This function is typically used with a time dimension.

Syntax

```
openingPeriod ( level [ , member ] )
```

Example 1

```
openingPeriod ( [great_outdoors_company].[Years].[Years].[Month] )
```

Result: 2004/Jan

Example 2

```
openingPeriod ( [great_outdoors_company].[Years].[Years].[Year] )
```

Result: 2004

Example 3

```
openingPeriod ( [great_outdoors_company].[Years].[Years].[Month] , [2006 Q 4] )
```

Result: 2006/Oct

order

Arranges the members of "set_expression" according to their "value_expression" and the third parameter. ASC and DESC arrange members in ascending or descending order, respectively, according to their position in the set hierarchy. Then the children of each member are arranged according to "value_expression". BASC and BDESC arrange members in the set without regard to the hierarchy. In the absence of an explicit specification, ASC is the default.

Syntax

```
order ( set_expression , value_expression [ , ASC|DESC|BASC|BDESC ] )
```

Example 1

```
order ( members ( [Great Outdoors Company].[Product].[Product].[Product type] ) , [Quantity sold] , BASC )
```

Result: Returns the quantity sold for each product type in no particular order.

Product Line	Quantity
-----	-----
Woods	13,924
Irons	14,244
Safety	22,332
...	...
Sunscreen	215,432
Insect Repellents	270,074
Lanterns	345,096

Example 2

```
order ( members ( [Great Outdoors Company].[Product].[Product].[Product type] ) , [Quantity sold] , ASC )
```

Result: Returns the quantity sold for each product type in ascending order.

Product Line	Quantity
-----	-----
Woods	13,924
Irons	14,244
Putters	23,244
...	...
Tents	130,664
Cooking Gear	198,676
Lanterns	345,096

ordinal

Returns the zero-based ordinal value (distance from the root level) of "level".

Syntax

```
ordinal ( level )
```

Example 1

```
ordinal ( [great_outdoors_company].[Products].[Products].[Product line] )
```

Result: 1

Example 2

```
ordinal ( [great_outdoors_company].[Products].[Products].[Product type] )
```

Result: 2

parallelPeriod

Returns a member from a prior period in the same relative position as "member". This function is similar to the cousin function, but is more closely related to time series. It takes the ancestor of "member" at "level" (called "ancestor") and the sibling of "ancestor" that lags by "integer_expression" positions, and returns the parallel period of "member" among the descendants of that sibling. When unspecified, "integer_expression" defaults to 1 and "member" defaults to the current member.

Syntax

```
parallelPeriod ( level [ , integer_expression [ , member ] ] )
```

Example 1

```
parallelPeriod ( [great_outdoors_company].[Years].[Years].[Quarter] , -1 , [2006/Aug] )
```

Result: 2006/Nov

Example 2

```
parallelPeriod ( [great_outdoors_company].[Years].[Years].[Quarter] , 1 , [2006/Aug] )
```

Result: 2006/May

Example 3

```
parallelPeriod ( [great_outdoors_company].[Years].[Years].[Year] , 2 , [2006/Aug] )
```

Result: 2004/Aug

parent

Returns the member that is the parent of "member" or "measure".

Syntax

```
parent ( member|measure )
```

Example

```
parent ( [Cooking Gear] )
```

Result: Camping Equipment

periodsToDate

Returns a set of sibling members from the same level as "member", as constrained by "level". It locates the ancestor of "member" at "level" and returns that ancestor's descendants at the same level as "member" (up to and including "member"). Typically used with a time dimension. This function appears in the Rolling and Moving Averages interactive sample report.

Syntax

```
periodsToDate ( level , member )
```

Example

```
periodsToDate ( [great_outdoors_company].[Years].[Years].[Year] , [2004/Mar] )
```

Result: Returns values for [2004/Jan], [2004/Feb], [2004/Mar]

prevMember

Returns the member that immediately precedes "member" in the same level. This function appears in the Sales Growth Year Over Year sample report in the GO Data Warehouse (analysis) package.

Syntax

```
prevMember ( member )
```

Example 1

```
prevMember ( [Outdoor Protection] )
```

Result: Personal Accessories

Example 2

```
prevMember ( [2005] )
```

Result: 2004

member

Defines a member based on "value_expression" in "hierarchy". "String1" identifies the member created by this function. It must be unique in the query and different from any other member in the same hierarchy. "String2" is the caption of the member; if it is absent, the caption is empty. To ensure predictable results, it is recommended that you supply the "hierarchy". Note: All calculations used as grouping items whose sibling items are other calculations or member sets should be explicitly assigned to a hierarchy using this function. The results are not predictable otherwise. The only exception is where the calculation involves only members of the same hierarchy as the siblings. In this case, the calculation is assumed to belong to that hierarchy.

Syntax

```
member ( value_expression [ , string1 [ , string2 [ , hierarchy ] ] ] )
```

Example

```
member ( total ( currentMeasure within set filter ( [great_outdoors_company].
[Products].[Products].[Product name] , caption ( [great_outdoors_company].
[Products].[Products].[Product name] ) starts with 'B' ) ) , 'BProducts' , 'B
Products' , [great_outdoors_company].[Products].[Products] )
```

Result: Returns the quantity sold and revenue for all products that start with the letter B.

nestedSet

Returns the set of members of "set_expression2" evaluated in the context of the current member of "set_expression1".

Syntax

```
nestedSet ( set_expression1 , set_expression2 )
```

Example

```
nestedSet ( members ( [Product line] ) , topCount ( descendants ( currentMember
( [great_outdoors_company].[Products].[Products] ) , [great_outdoors_company].
[Products].[Products].[Product name] ) , 2 , [Revenue] ) )
```

Result: Returns the top two products by revenue for each product line.

set

Returns the list of members defined in the expression. The members must belong to the same hierarchy.

Syntax

```
set ( member { , member } )
```

Example

```
set ( [Golf Equipment] , [Irons] , [TrailChef Cup] )
```

Result: Returns Golf Equipment, Irons, and TrailChef Cup.

siblings

Returns the children of the parent of the specified member.

Syntax

```
siblings ( member )
```

Example

```
siblings ( [Golf Equipment] )
```

Result: Returns the siblings of Golf Equipment.

```
Camping Equipment
Golf Equipment
Mountaineering Equipment
Outdoor Protection
Personal Accessories
```

tail

Returns the last "index_expression" elements of "set expression". The default for "index_expression" is 1.

Syntax

```
tail ( set_expression [ , index_expression ] )
```

Example 1

```
tail (members ( [great_outdoors_company].[Products].[Products].[Product
line] ) )
```

Result: Returns the last member of the Product line set.

```
Personal Accessories
```

Example 2

```
tail ( members ( [great_outdoors_company].[Products].[Products].[Product
line] ) , 2 )
```

Result: Returns the last two members of the Product line set.

```
Outdoor Protection
Personal Accessories
```

topCount

Sorts a set according to the values of "numeric_expression" evaluated at each of the members of "set_expression" and returns the top "index_expression" members.

Syntax

```
topCount ( set_expression , index_expression , numeric_expression )
```

Example

```
topCount ( [great_outdoors_company].[Products].[Products].[Product line] , 2 ,
[Revenue] )
```

Result: Returns the top two revenues for the Product line set.

Product line	Revenue
-----	-----
Camping Equipment	\$89,713,990.92
Personal Accessories	\$31,894,465.86

topPercent

Sorts "numeric_expression2", evaluated at the corresponding members of "set_expression", and picks up the topmost elements whose cumulative total is at least "numeric_expression1" percent of the total.

Syntax

```
topPercent ( set_expression , numeric_expression1 , numeric_expression2 )
```

Example

```
topPercent ( set ( [Camping Equipment] , [Golf Equipment] , [Mountaineering Equipment] ) , 40 , [2006] )
```

Result: For the set of Camping Equipment, Golf Equipment, and Mountaineering Equipment, returns the members whose percentage totals are greater than or equal to 40% for 2006.

topSum

Sorts "numeric_expression2", evaluated at the corresponding members of "set_expression", and picks up the topmost elements whose cumulative total is at least "numeric_expression1".

Syntax

```
topSum ( set_expression , numeric_expression1 , numeric_expression2 )
```

Example

```
topSum ( children ( [Products] ) , 16000000 , tuple ( [2006] , [great_outdoors_company].[Measures].[Gross profit] ) )
```

tuple

Identifies a cell location (intersection) based on the specified members, each of which must be from a different dimension. This function implicitly includes the current member from all dimensions that are not otherwise specified in the arguments. The current member of any dimension not specified in the evaluating context is assumed to be the default member of that dimension. The value of this cell can be obtained with the "value" function.

Syntax

```
tuple ( member { , member } )
```

Example

```
tuple ( [Mountaineering Equipment] , [Fax] )
```

Result: Returns the Mountaineering Equipment sales by fax.

union

Returns data for "set_expression1" and "set_expression2". The result retains duplicates only when the optional keyword "all" is supplied as the third argument.

Syntax

```
union ( set_expression1 , set_expression2 [ , all ] )
```

Example 1

```
union ( set ( [Camping Equipment] , [Golf Equipment] ) , set ( [Golf  
Equipment] , [Mountaineering Equipment] ) )
```

Result: Returns data for both sets as one new set, showing the Golf Equipment column only once.

Example 2

```
union ( set ( [Camping Equipment] , [Golf Equipment] ) , set ( [Golf  
Equipment] , [Mountaineering Equipment] ) , all )
```

Result: Returns data for both sets as one new set, showing the Golf Equipment column twice.

roleValue

Returns the value of the attribute that is associated with the role whose name is specified by "string" within the specified context. "Member" or "set_expression" is optional only in a number of limited circumstances, where it can be derived from another context. Applications can be made portable across different data sources and models by accessing attributes by role rather than by query item ID. For dimensionally-modeled relational (DMR) data sources, assignment of roles is the modeler's responsibility. Intrinsic roles that are defined for members of all data source types include: '_businessKey', '_memberCaption', '_memberDescription', '_memberUniqueName'. Additional roles can be defined in Framework Manager for each level in a hierarchy. For example, a Product type level may have an attribute column called "Type Shipping Container", and the Product level may have a "Product Shipping Container" attribute. Each of these could be assigned a custom role in Framework Manager called "Container". The property could then be referenced independently of the actual column name by using the roleValue function.

Syntax

```
roleValue ( string [ , member|set_expression ] )
```

Example 1

```
roleValue ( '_memberCaption' , [Sales].[Product].[Product].[Product line] ->  
[all].[1] )
```

Result: Camping Equipment

Example 2

```
roleValue ( '_businessKey' , [great_outdoors_company].[Years].[Years].[Year] )
```

Result: Returns the value of the attribute that is associated with the business key role.

```
("2004-01-01","2004-12-31")  
("2005-01-01","2005-12-31")  
("2006-01-01","2006-12-31")
```

Example 3

```
roleValue ( '_memberUniqueName' , [great_outdoors_company].[Years].[Years].  
[Year] )
```

Result: Returns the value of the attribute that is associated with the MUN role.

```
[great_outdoors_company].[Years].[Years].[Year] ->:[PC].[Years (Root)].[20040101-  
20041231]  
[great_outdoors_company].[Years].[Years].[Year] ->:[PC].[Years (Root)].[20050101-  
20051231]  
[great_outdoors_company].[Years].[Years].[Year] ->:[PC].[Years (Root)].[20060101-  
20061231]
```

rootMember

Returns the root member of a single-root hierarchy. This function appears in the Promotion Success sample report in the GO Data Warehouse (analysis) package.

Syntax

```
rootMember ( hierarchy )
```

rootMembers

Returns the root members of a hierarchy.

Syntax

```
rootMembers ( hierarchy )
```

Example

```
rootMembers ( [great_outdoors_company].[Years].[Years] )
```

Result: By Time

subset

Returns a subset of members in "set_expression" starting at "index_expression1" from the beginning. If the count "index_expression2" is specified, that many members are returned (if available). Otherwise, all remaining members are returned.

Syntax

```
subset ( set_expression, index_expression1 [ , index_expression2 ] )
```

Example 1

```
subset ( members ( [great_outdoors_company].[Products].[Products].[Product  
line] ) , 2 )
```

Result: Returns the members of the Product line set starting at the second member.

```
Mountaineering Equipment  
Outdoor Protection  
Personal Accessories
```

Example 2

```
subset ( members ( [great_outdoors_company].[Products].[Products].[Product  
line] ) , 2 , 2 )
```

Result: Returns two members of the Product line set starting at the second member.

Mountaineering Equipment
Outdoor Protection

unique

Removes all duplicates from "set_expression". The remaining members retain their original order.

Syntax

```
unique ( set_expression )
```

value

Returns the value of the cell identified by "tuple". Note that the default member of the Measures dimension is the Default Measure.

Syntax

```
value ( tuple )
```

Example 1

```
value ( tuple ( [great_outdoors_company].[Years].[Years].[Year] ->:[PC].[Years  
(Root)].[20040101-20041231] , [great_outdoors_company].[Measures].[Revenue] ) )
```

Result: \$34,750,563.50

Example 2

```
value ( tuple ( [2004] , [Camping Equipment] , [Revenue] ) )
```

Result: \$20,471,328.88

DB2

ascii

Returns the ASCII code value of the leftmost character of the argument as an integer.

Syntax

```
ascii ( string_expression )
```

Example

```
ascii ( a )
```

Result: Returns 65, the ASCII code value of "a".

ceiling

Returns the smallest integer greater than or equal to "numeric_expression".

Syntax

```
ceiling ( numeric_expression )
```

Example

```
ceiling ( 0.75 )
```

Result: Returns 0.8.

char

Returns a string representation of a date/time value or a decimal number.

Syntax

```
char ( expression )
```

chr

Returns the character that has the ASCII code value specified by "integer_expression". "Integer_expression" should be between 0 and 255.

Syntax

```
chr ( integer_expression )
```

Example

```
chr ( 65 )
```

Result: Returns a, the character for the ASCII code value of 65.

concat

Returns a string that is the result of concatenating "string_expression1" with "string_expression2".

Syntax

```
concat ( string_expression1, string_expression2 )
```

Example

```
concat ( [Sales target (query)].[Sales staff].[First name], [Sales target  
(query)].[Sales staff].[Last name] )
```

Result: Returns the first name and last name; e.g., Bob Smith.

date

Returns a date from a single input value. "Expression" can be a string or integer representation of a date.

Syntax

```
date ( expression )
```

Example

```
date ( '1998-01-08' )
```

Result: Returns 8 January 1998.

day

Returns the day of the month (1-31) from "date_expression". "Date_expression" can be a date value or a string representation of a date.

Syntax

```
day ( date_expression )
```

Example

```
day ( '1998-01-08' )
```

Result: Returns 8.

dayname

Returns a character string containing the data source-specific name of the day (for example, Sunday through Saturday or Sun. through Sat. for a data source that uses English, or Sonntag through Samstag for a data source that uses German) for the day portion of "date_expression".

"Date_expression" can be a date value or a string representation of a date.

Syntax

```
dayname ( date_expression )
```

Example

```
dayname ( '1998-01-08' )
```

Result: Returns Thursday.

dayofweek

Returns the day of the week in "date_expression" as an integer in the range 1 to 7, where 1 represents Sunday. "date_expression" can be a date value or a string representation of a date.

Syntax

```
dayofweek ( date_expression )
```

Example

```
dayofweek ( '1998-01-08' )
```

Result: Returns 5.

dayofweek_iso

Returns the day of the week in "date_expression" as an integer in the range 1 to 7, where 1 represents Monday. "date_expression" can be a date value or a string representation of a date.

Syntax

```
dayofweek_iso ( date_expression )
```

Example

```
dayofweek_iso ( '1998-01-08' )
```

Result: Returns 4.

dayofyear

Returns the day of the year in "date_expression" as an integer in the range 1 to 366.

"Date_expression" can be a date value or a string representation of a date.

Syntax

```
dayofyear ( date_expression )
```

Example

```
dayofyear ( current_date )
```

Result: Returns the day of the year for the current date; e.g., if it was January 28, the expression would return 28.

days

Returns an integer representation of a date. "Expression" can be a date value or a string representation of a date.

Syntax

```
days ( expression )
```

dec

Returns the decimal representation of "string_expression1" with precision "numeric_expression1", scale "numeric_expression2", and decimal character "string_expression2". "String_expression1" must be formatted as an SQL Integer or Decimal constant.

Syntax

```
dec ( string_expression1 [ , numeric_expression1 [ , numeric_expression2  
[ , string_expression2 ] ] ] )
```

decimal

Returns the decimal representation of "string_expression1" with precision "numeric_expression1", scale "numeric_expression2" and decimal character "string_expression2". "String_expression1" must be formatted as an SQL Integer or Decimal constant.

Syntax

```
decimal ( string_expression1 [ , numeric_expression1 [ , numeric_expression2  
[ , string_expression2 ] ] ] )
```

difference

Returns an integer value representing the difference between the values returned by the data source-specific soundex function for "string_expression1" and "string_expression2". The value returned ranges from 0 to 4, with 4 indicating the best match. Note that 4 does not mean that the strings are equal.

Syntax

```
difference ( string_expression1 , string_expression2 )
```

Example 1

```
difference ([Sales target (query)].[Sales staff].[First name],[Sales (query)].  
[Retailers].[Contact first name])
```

Result: 0

Example 2

```
difference ([Sales target (query)].[Sales staff].[First name],[Sales target  
(query)].[Sales staff].[First name])
```


Result: 4

digits

Returns the character string representation of a non-floating point number.

Syntax

```
digits ( numeric_expression )
```

double

Returns the floating-point representation of an expression. "Expression" can either be a numeric or string expression.

Syntax

```
double ( expression )
```

event_mon_state

Returns the operational state of a particular state monitor.

Syntax

```
event_mon_state ( string_expression )
```

float

Returns the floating-point representation of a number.

Syntax

```
float ( numeric_expression )
```

hex

Returns the hexadecimal representation of a value.

Syntax

```
hex ( expression )
```

hour

Returns the hour, an integer from 0 (midnight) to 23 (11:00 pm), from "time_expression". "Time_expression" can be a time value or a string representation of a time.

Syntax

```
hour ( time_expression )
```

Example

```
hour ( 01:22:45 )
```

Result: Returns 1.

insert

Returns a string where "integer_expression2" characters have been deleted from "string_expression1" beginning at "integer_expression1" and where "string_expression2" has been inserted into "string_expression1" at its start. The first character in the string is at position 1.

Syntax

```
insert ( string_expression1, integer_expression1, integer_expression2, string_expression2 )
```

integer

Returns the integer representation of an expression. "Expression" can be a numeric value or a string representation of a number.

Syntax

```
integer ( expression )
```

Example

```
integer ( 84.95 )
```

Result: 85

int

Returns the integer representation of an expression. "Expression" can be a numeric value or a string representation of a number.

Syntax

```
int ( expression )
```

Example

```
int ( 84.95 )
```

Result: 85

julian_day

Returns an integer value representing the number of days from January 1, 4712 BC (the start of the Julian date calendar) to the date value specified in "expression". "Expression" can be a date value or a string representation of a date.

Syntax

```
julian_day ( expression )
```

Example

```
julian_day ( '2009-06-29' )
```

Result: 2455012.22130739595741034

lcase

Returns "string_expression" with all uppercase characters shifted to lowercase.

Syntax

```
lcase ( string_expression )
```

Example

```
lcase ( [Sales (query)].[Sales staff].[Last name] )
```

Result: Returns last names with no uppercase letters.

left

Returns the leftmost "integer_expression" characters of "string_expression".

Syntax

```
left ( string_expression, integer_expression )
```

Example

```
left ( [Sales (query)].[Sales staff].[Last name] , 3 )
```

Result: Returns the first three characters of each last name.

length

Returns the length of the operand in bytes. Exception: double byte string types return the length in characters.

Syntax

```
length ( expression )
```

Example

```
length ( [Sales (query)].[Sales staff].[Record start date] )
```

Result: Returns 4; dates always return a value of 4.

locate

Returns the starting position of the first occurrence of "string_expression1" within "string_expression2". The search starts at position start "integer_expression" of "string_expression2". The first character in a string is at position 1. If "string_expression1" is not found, zero is returned.

Syntax

```
locate ( string_expression1, string_expression2 [ , integer_expression ] )
```

Example

```
locate ( A, [Sales (query)].[Sales staff].[Last name] , 2 )
```

Result: Returns the position of the character A in the last names starting at the second character of the last name.

long_varchar

Returns a long string.

Syntax

```
long_varchar ( string_expression )
```

ltrim

Returns "string_expression" with leading spaces removed.

Syntax

```
ltrim ( string_expression )
```

Example

```
ltrim ( [Sales (query)].[Sales staff].[Last name] )
```

Result: Returns last names with any leading spaces removed.

microsecond

Returns the microsecond (time-unit) part of a value. "Expression" can be a timestamp or a string representation of a timestamp.

Syntax

```
microsecond ( expression )
```

Example

```
microsecond ( 01:45:34.056 )
```

Result: Returns 056.

midnight_seconds

Returns an integer value in the range 0 to 86400 representing the number of seconds between midnight and time value specified in the argument. "Expression" can be a time value, a timestamp or a string representation of a time.

Syntax

```
midnight_seconds ( expression )
```

Example

```
midnight_seconds ( 01:45:34.056 )
```

Result: Returns 6334.

minute

Returns the minute (an integer from 0-59) from "time_expression". "Time_expression" can be a time value, a timestamp, or a string representation of a time.

Syntax

```
minute ( time_expression )
```

Example

```
minute ( 01:45:34.056 )
```

Result: Returns 45.

month

Returns the month (an integer from 1-12) from "date_expression".

Syntax

```
month ( date_expression )
```

Example

```
month ( 2005-11-01 )
```

Result: Returns 11.

monthname

Returns a character string containing the data source-specific name of the month (for example, January through December or Jan. through Dec. for an English data source, or Januar through Dezember for a German data source) for the month portion of "date_expression".

Syntax

```
monthname ( date_expression )
```

Example

```
monthname ( 2005-11-01 )
```

Result: November

quarter

Returns the quarter in "date_expression" as a number in the range 1 to 4, where 1 represents January 1 through March 31.

Syntax

```
quarter ( date_expression )
```

Example

```
quarter ( 2005-11-01 )
```

Result: Returns 4.

radians

Returns the number of radians converted from "numeric_expression" degrees.

Syntax

```
radians ( numeric_expression )
```

repeat

Returns a string consisting of "string_expression" repeated "integer_expression" times.

Syntax

```
repeat ( string_expression, integer_expression )
```

Example

```
repeat ( XYZ, 3 )
```

Result: Returns XYZXYZXYZ.

replace

Replaces all occurrences of "string_expression2" in "string_expression1" with "string_expression3".

Syntax

```
replace ( string_expression1, string_expression2, string_expression3 )
```

Example

```
replace ( [Sales (query)].[Sales staff].[Position code], A, a )
```

Result: Returns position codes with all occurrences of "A" replaced by "a".

right

Returns the rightmost "integer_expression" characters of "string_expression".

Syntax

```
right ( string_expression, integer_expression )
```

Example

```
right ( [Sales (query)].[Sales staff].[Position code], 3 )
```

Result: Returns the rightmost 3 characters of each position code.

round

Returns "numeric_expression" rounded to "integer_expression" places to the right of the decimal point. If "integer_expression" is negative, "numeric_expression" is rounded to the nearest absolute value "integer_expression" places to the left of the decimal point. Rounding takes place before data formatting is applied.

Syntax

```
round ( numeric_expression, integer_expression )
```

Example

```
round ( 3.14159265, 3 )
```

Result: Returns 3.142.

rtrim

Returns "string_expression" with trailing spaces removed.

Syntax

```
rtrim ( string_expression )
```

Example

```
rtrim ( [Sales (query)].[Sales staff].[Last name] )
```

Result: Returns last names with any spaces at the end of the name removed.

second

Returns the second (an integer from 0-59) from "time_expression".

Syntax

```
second ( time_expression )
```

Example

```
second ( 01:45:34.056 )
```

Result: Returns 34.

sign

Returns an indicator of the sign of "numeric_expression": +1 if "numeric_expression" is positive, 0 if zero, or -1 if negative.

Syntax

```
sign ( numeric_expression )
```

Example

```
sign ( [Revenue] )
```

Result: Returns + for positive values and - for negative values.

smallint

Returns the small integer representation of a number.

Syntax

```
smallint ( expression )
```

soundex

Returns a 4 character string code obtained by systematically abbreviating words and names in "string_expression" according to phonetics. Can be used to determine if two strings sound the same. For example, does sound-of ('SMITH') = sound-of ('SMYTH').

Syntax

```
soundex ( string_expression )
```

space

Returns a string consisting of "integer_expression" spaces.

Syntax

```
space ( integer_expression )
```

Example

```
space ( 5 )
```

Result: Returns 5 spaces.

substr

Returns the substring of "string_expression" that starts at position "integer_expression1" for "integer_expression2" characters. The first character in "string_expression" is at position 1.

Syntax

```
substr ( string_expression , integer_expression1 [ , integer_expression2 ] )
```

Example

```
substr ( [Sales (query)].[Sales staff].[Position code], 3 , 5 )
```

Result: Returns characters 3 to 7 of the position codes.

table_name

Returns an unqualified name of a table or view based on the object name in "string_expression1" and the schema name given in "string_expression2". It is used to resolve aliases.

Syntax

```
table_name ( string_expression1 [ , string_expression2 ] )
```

table_schema

Returns the schema name portion of the two-part table or view name based on the object name in "string_expression1" and the schema name in "string_expression2". It is used to resolve aliases.

Syntax

```
table_schema ( string_expression1 [ , string_expression2 ] )
```

time

Returns a time from a value.

Syntax

```
time ( expression )
```

timestamp

Returns a timestamp from a value or a pair of values. "Expression1" must represent a date value, and "expression2" must represent a time value.

Syntax

```
timestamp ( expression1 [ , expression2 ] )
```

Example

```
timestamp ( 11 November 2005 , 12:00:00.000000 )
```

Result: Returns 2005-11-11-12:00:00.000000.

timestamp_iso

Returns a datetime in the ISO format (yyyy-mm-dd hh:mm:ss.nnnnnnn) converted from the IBM format (yyyy-mm-dd-hh.mm.ss.nnnnnnn). If "expression" is a time, it inserts the value of the CURRENT DATE for the date elements and zero for the fractional time element.

Syntax

```
timestamp_iso ( expression )
```

Example

```
timestamp_iso ( 11 November 2005 , 12:00:00.000000 )
```

Result: Returns 2005-11-11 12:00:00.000000.

timestampdiff

Returns an estimated number of intervals of type "expression1" based on the difference between two timestamps. "Expression2" is the result of subtracting two timestamp types and converting the result to CHAR. Valid values of "expression1" are: 1 Fractions of a second; 2 Seconds; 4 Minutes; 8 Hours; 16 Days; 32 Weeks; 64 Months; 128 Quarters; 256 Years.

Syntax

```
timestampdiff ( expression1, expression2 )
```

to_char

Returns the string representation of a timestamp with the format of "string_expression".

Syntax

```
to_char ( timestamp_expression , string_expression )
```

translate

Returns "string_expression1" in which characters from "string_expression3" are translated to the equivalent characters in "string_expression2". "String_expression4" is a single character that is used to pad "string_expression2" if it is shorter than "string_expression3". If only "string_expression1" is present, then this function translates it to uppercase characters.

Syntax

```
translate ( string_expression1 [ , string_expression2, string_expression3  
[ , string_expression4 ] ] )
```

Example 1

```
translate ( 'abcdefg' )
```

Result: Returns ABCDEFG.

Example 2

```
translate ( 'mnlop' , n, m , - )
```

Result: Returns n-nlop.

trunc

Returns "numeric_expression1" truncated to "numeric_expression2" places to the right of the decimal point. If "numeric_expression2" is negative, "numeric_expression1" is truncated to the absolute value of "numeric_expression2" places to the left of the decimal point.

Syntax

```
trunc ( numeric_expression1, numeric_expression2 )
```

Example

```
trunc ( 3.14159265, 3 )
```

Result: Returns 3.141.

truncate

Returns "numeric_expression1" truncated to "numeric_expression2" places to the right of the decimal point. If "numeric_expression2" is negative, "numeric_expression1" is truncated to the absolute value of "numeric_expression2" places to the left of the decimal point.

Syntax

```
truncate ( numeric_expression1, numeric_expression2 )
```

Example

```
truncate ( 3141.59265, -3 )
```

Result: Returns 3.

ucase

Returns "string_expression" with all lowercase characters shifted to uppercase.

Syntax

```
ucase ( string_expression )
```

Example

```
ucase ( XY896Zbcd789 )
```

Result: Returns XY896ZBCED789.

value

Returns the first non-null argument (or null if all arguments are null). The Value function takes two or more arguments.

Syntax

```
value ( expression_list )
```

Example

```
value ( [Unit cost], [Unit price], [Unit sale price] )
```

Result: Returns the first non-null value.

varchar

Returns a VARCHAR representation of expression, with length numeric_expression.

Syntax

```
varchar ( expression [ , numeric_expression ] )
```

week

Returns the week of the year in "date_expression" as an integer value in the range 1 to 53.

Syntax

```
week ( date_expression )
```

Example

```
week ( 11 November 2005 )
```

Result: Returns 45.

year

Returns the year from "date_expression".

Syntax

```
year ( date_expression )
```

Example

```
year ( 11 November 2005 )
```

Result: Returns 2005.

DB2 Math**log**

Returns the natural logarithm of "numeric_expression".

Syntax

```
log ( numeric_expression )
```

log10

Returns the base ten logarithm of "numeric_expression".

Syntax

```
log10 ( numeric_expression )
```

rand

Generates a random number using "integer_expression" as a seed value.

Syntax

```
rand ( integer_expression )
```

DB2 Trigonometry**acos**

Returns the arccosine of "numeric_expression" in radians. The arccosine is the angle whose cosine is "numeric_expression".

Syntax

```
acos ( numeric_expression )
```

asin

Returns the arcsine of "numeric_expression" in radians. The arcsine is the angle whose sine is "numeric_expression".

Syntax

```
asin ( numeric_expression )
```

atan

Returns the arctangent of "numeric_expression" in radians. The arctangent is the angle whose tangent is "numeric_expression".

Syntax

```
atan ( numeric_expression )
```

atanh

Returns the hyperbolic arctangent of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

Syntax

```
atanh ( numeric_expression )
```

atan2

Returns the arctangent of the x and y coordinates specified by "numeric_expression1" and "numeric_expression2", respectively, in radians. The arctangent is the angle whose tangent is "numeric_expression2" / "numeric_expression1".

Syntax

```
atan2 ( numeric_expression1 , numeric_expression2 )
```

cos

Returns the cosine of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

Syntax

```
cos ( numeric_expression )
```

cosh

Returns the hyperbolic cosine of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

Syntax

```
cosh ( numeric_expression )
```

cot

Returns the cotangent of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

Syntax

```
cot ( numeric_expression )
```

degrees

Returns "numeric_expression" radians converted to degrees.

Syntax

```
degrees ( numeric_expression )
```

sin

Returns the sine of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

Syntax

```
sin ( numeric_expression )
```

sinh

Returns the hyperbolic sine of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

Syntax

```
sinh ( numeric_expression )
```

tan

Returns the tangent of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

Syntax

```
tan ( numeric_expression )
```

tanh

Returns the hyperbolic tangent of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

Syntax

```
tanh ( numeric_expression )
```

Informix**cardinality**

Returns the number of elements in a collection column (SET, MULTiset, LIST).

Syntax

```
cardinality ( string_expression )
```

char_length

Returns the number of logical characters in "string_expression". The number of logical characters can be distinct from the number of bytes in some East Asian locales.

Syntax

```
char_length ( string_expression )
```

concat

Returns a string that is the result of concatenating, or joining, "string_expression1" to "string_expression2".

Syntax

```
concat ( string_expression1 , string_expression2 )
```

Example

```
concat ( [Sales (query)].[Sales staff].[First name], [Sales (query)].[Sales  
staff].[Last name] )
```

Result: Returns the first name and last name; e.g., Bob Smith.

date

Returns the date value of "string_expression", "date_expression", or "integer_expression".

Syntax

```
date ( string_expression|date_expression|integer_expression )
```

day

Returns an integer that represents the day of the month (1-31).

Syntax

```
day ( date_expression )
```

extend

Adjusts the precision of a datetime or date expression. The expression cannot be a quoted string representation of a date value. If you do not specify first and last qualifiers, the default qualifiers are year to fraction (3). If the expression contains fields that are not specified by the qualifiers, the unwanted fields are discarded. If the first qualifier specifies a larger (more significant) field than what exists in the expression, the new fields are filled in with values returned by the current function. If the last qualifier specifies a smaller (less significant) field than what exists in the expression, the new fields are filled in with constant values. A missing month or day field is filled in with 1, and missing hour to fraction fields are filled in with 0.

Syntax

```
extend ( date_expression , ' { ' year to second ' } ' )
```

Example

```
extend ( some_date_column , { year to second } )
```

hex

Returns the hexadecimal encoding of "integer_expression".

Syntax

```
hex ( integer_expression )
```

initcap

Returns "string_expression" with the first letter of each word in uppercase and all other letters in lowercase. A word begins after any character other than a letter. Thus, in addition to a blank space, symbols such as commas, periods, and colons can introduce a new word.

Syntax

```
initcap ( string_expression )
```

length

Returns the number of bytes in "string_expression", not including any trailing blank spaces. For byte or text "string_expression", length returns the full number of bytes, including any trailing blank spaces.

Syntax

```
length ( string_expression )
```

lpad

Returns "string_expression1" left-padded by "string_expression2" to the total number of characters specified by "integer_expression". The sequence of "string_expression2" occurs as many times as necessary to make the return string the length specified by "integer_expression".

Syntax

```
lpad ( string_expression1 , integer_expression , string_expression2 )
```

mdy

Returns a type date value with three expressions that evaluate to integers that represent the month (integer_expression1), day (integer_expression2), and year (integer_expression3).

Syntax

```
mdy ( integer_expression1 , integer_expression2 , integer_expression3 )
```

month

Returns an integer corresponding to the month portion of "date_expression".

Syntax

```
month ( date_expression )
```

nvl

Returns the value of "expression1" if "expression1" is not NULL. If "expression1" is NULL, then returns the value of "expression2".

Syntax

```
nvl ( expression1 , expression2 )
```

Example

```
nvl ( [Unit sale price] , [Unit price] )
```

Result: Returns the unit sale price, or returns the unit price if the unit sale price is NULL.

octet_length

Returns the number of bytes in "string_expression", including any trailing spaces.

Syntax

```
octet_length ( string_expression )
```

replace

Returns "string_expression1" in which every occurrence of "string_expression2" is replaced by "string_expression3". If you omit the "string_expression3" option, every occurrence of "string_expression2" is omitted from the return string.

Syntax

```
replace ( string_expression1 , string_expression2 [ , string_expression3 ] )
```

Example

```
replace ( [Sales (query)].[Products].[Product line code] , - )
```

Result: Returns all product line codes without the character "-"

round

Returns the rounded value of "numeric_expression". If you omit "integer_expression", the value is rounded to zero digits or to the units place. The digit range of 32 (+ and -) refers to the entire decimal value. Rounding takes place before data formatting is applied.

Syntax

```
round ( numeric_expression [ , integer_expression ] )
```

Example

```
round (125, -1)
```

Result: 130

rpad

Returns "string_expression1" right-padded by "string_expression2" to the total number of characters specified by "integer_expression". The sequence of "string_expression2" occurs as many times as necessary to make the return string the length specified by "integer_expression".

Syntax

```
rpadd ( string_expression1 , integer_expression , string_expression2 )
```

substr

Returns the substring of "string_expression" that starts at position "integer_expression1" for "integer_expression2" characters. The first character in "string_expression" is at position 1. If you omit "integer_expression2", returns the substring of "string_expression" that starts at position "integer_expression1" and ends at the end of "string_expression".

Syntax

```
substr ( string_expression , integer_expression1 [ , integer_expression2 ] )
```

Example

```
substr ( [Sales (query)].[Sales staff].[Position code], 3 , 5 )
```

Result: Returns characters 3 to 7 of the position codes.

to_char

Returns the character string "date_expression" with the specified "string_expression" formatting. You can use this function only with built-in data types.

Syntax

```
to_char ( date_expression , string_expression )
```

to_date

Returns "string_expression1" as a date according to the date format you specify in "string_expression2". If "string_expression1" is NULL, then a NULL value is returned.

Syntax

```
to_date ( string_expression1 , string_expression2 )
```

trunc

Returns the truncated value of "numeric_expression". If you omit "integer_expression", then "numeric_expression" is truncated to zero digits or to the unit's place. The digit limitation of 32 (+ and -) refers to the entire decimal value.

Syntax

```
trunc ( numeric_expression [ , integer_expression ] )
```

weekday

Returns an integer that represents the day of the week of "date_expression". Zero (0) represents Sunday, one (1) represents Monday, and so on.

Syntax

```
weekday ( date_expression )
```

year

Returns a four-digit integer that represents the year of "date_expression".

Syntax

```
year ( date_expression )
```

Informix Math

log10

Returns the logarithm of "numeric_expression" to base 10.

Syntax

```
log10 ( numeric_expression )
```

logn

Returns the natural logarithm of "numeric_expression".

Syntax

```
logn ( numeric_expression )
```

root

Returns the root value of "numeric_expression1". Requires at least one numeric argument (the radians argument). If only "numeric_expression1" is supplied, 2 is used as a default value for "numeric_expression2". Zero cannot be used as the value of "numeric_expression2".

Syntax

```
root ( numeric_expression1 [ , numeric_expression2 ] )
```

Informix Trigonometry

acos

Returns the arccosine of "numeric_expression" in radians. The arccosine is the angle whose cosine is "numeric_expression".

Syntax

```
acos ( numeric_expression )
```

asin

Returns the arcsine of "numeric_expression" in radians. The arcsine is the angle whose sine is "numeric_expression".

Syntax

```
asin ( numeric_expression )
```

atan

Returns the arctangent of "numeric_expression" in radians. The arctangent is the angle whose tangent is "numeric_expression".

Syntax

```
atan ( numeric_expression )
```

atan2

Returns the arctangent of the x and y coordinates specified by "numeric_expression1" and "numeric_expression2", respectively, in radians. The arctangent is the angle whose tangent is "numeric_expression1".

Syntax

```
atan2 ( numeric_expression1 , numeric_expression2 )
```

cos

Returns the cosine of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

Syntax

```
cos ( numeric_expression )
```

sin

Returns the sine of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

Syntax

```
sin ( numeric_expression )
```

tan

Returns the tangent of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

Syntax

```
tan ( numeric_expression )
```

MS Access**ascii**

Returns the ascii code value of the leftmost character of "string_expression".

Syntax

```
ascii ( string_expression )
```

ceiling

Returns the smallest integer greater than or equal to "numeric_expression".

Syntax

```
ceiling ( numeric_expression )
```

chr

Returns the character that has the ASCII code value specified by "integer_expression". "Integer_expression" should be between 0 and 255.

Syntax

```
chr ( integer_expression )
```

concat

Returns a string that is the result of concatenating, or joining, "string_expression1" to "string_expression2".

Syntax

```
concat ( string_expression1 , string_expression2 )
```

Example

```
concat ( [Sales (query)].[Sales staff].[First name], [Sales (query)].[Sales  
staff].[Last name] )
```

Result: Returns the first name and last name; e.g., Bob Smith.

curdate

Returns a date value representing the current date of the computer that the database software runs on.

Syntax

```
curdate ()
```

curtime

Returns a time value representing the current time of the computer that the database software runs on.

Syntax

```
curtime ()
```

dayname

Returns a character string containing the data source-specific name of the day (for example, Sunday through Saturday or Sun. through Sat. for an English data source, or Sonntag through Samstag for a German data source) for the day portion of "date_expression".

Syntax

```
dayname ( date_expression )
```

dayofmonth

Returns the day of the month (1-31) from "date_expression". Returns the days field (a signed integer) from "interval_expression".

Syntax

```
dayofmonth ( date_expression|interval_expression )
```

dayofweek

Returns the day of the week in "date_expression" as an integer (1-7), where 1 represents Monday.

Syntax

```
dayofweek ( date_expression )
```

dayofyear

Returns the day of the year in "date_expression" as an integer (1-366).

Syntax

```
dayofyear ( date_expression )
```

hour

Returns the hour from "time_expression" as an integer from 0 (midnight) to 23 (11:00 pm).

Syntax

```
hour ( time_expression )
```

instr

Searches "string_expression1" for the first occurrence of "string_expression2" and returns an integer specifying the position of "string_expression2". "Integer_expression1" sets the starting position for the search. If "integer_expression1" is omitted, the search begins at the first character position of "string_expression1". "Integer_expression2" specifies the type of string comparison. "Integer_expression1" is required if "integer_expression2" is specified.

Syntax

```
instr ( [ integer_expression1 , ] string_expression1 , string_expression2  
[ , integer_expression2 ] )
```

lcase

Returns "string_expression" with all uppercase characters converted to lowercase.

Syntax

```
lcase ( string_expression )
```

left

Returns the leftmost "integer_expression" characters of "string_expression".

Syntax

```
left ( string_expression , integer_expression )
```

Example

```
left ( [Sales (query)].[Sales staff].[Last name] , 3 )
```

Result: Returns the first three characters of each last name.

length

Returns the number of characters in "string_expression", excluding trailing blanks and the string termination character.

Syntax

```
length ( string_expression )
```

locate

Returns the starting position of the first occurrence of "string_expression1" within "string_expression2". The search starts at position "integer_expression" of "string_expression2". The first character in a string is at position 1. If "string_expression1" is not found, then zero is returned.

Syntax

```
locate ( string_expression1 , string_expression2 [ , integer_expression ] )
```

ltrim

Returns "string_expression" with leading spaces removed.

Syntax

```
ltrim ( string_expression )
```

minute

Returns the minute (an integer from 0-59) from "time_expression".

Syntax

```
minute ( time_expression )
```

month

Returns the month (an integer from 1-12) from "date_expression".

Syntax

```
month ( date_expression )
```

monthname

Returns a character string containing the data source-specific name of the month (for example, January through December or Jan. through Dec. for an English data source, or Januar through Dezember for a German data source) for the month portion of "date_expression".

Syntax

```
monthname ( date_expression )
```

Example

```
monthname ( 2005-11-01 )
```

Result: November

now

Returns a datetime value representing the current date and time of the computer that the database software runs on.

Syntax

```
now ( )
```

position

Returns the starting position of "string_expression1" in "string_expression2". The first character in a string is at position 1.

Syntax

```
position ( string_expression1 , string_expression2 )
```

quarter

Returns the quarter in "date_expression" as a number (1-4), where 1 represents January 1 through March 31.

Syntax

```
quarter ( date_expression )
```

right

Returns the rightmost "integer_expression" characters of "string_expression".

Syntax

```
right ( string_expression , integer_expression )
```

round

Returns "numeric_expression" rounded to the nearest value "integer_expression" places right of the decimal point. If "integer_expression" is negative, "numeric_expression" is rounded to the nearest absolute value "integer_expression" places to the left of the decimal point. Rounding takes place before data formatting is applied.

Syntax

```
round ( numeric_expression , integer_expression )
```

Example

```
round (125, -1)
```

Result: 130

rtrim

Returns "string_expression" with trailing spaces removed.

Syntax

```
rtrim ( string_expression )
```

Example

```
rtrim ( [Sales (query)].[Sales staff].[Last name] )
```

Result: Returns last names with any spaces at the end of the name removed.

sign

Returns an indicator of the sign of "numeric_expression", +1 if positive, 0 if zero, or -1 if negative.

Syntax

```
sign ( numeric_expression )
```

space

Returns a string consisting of "integer_expression" spaces.

Syntax

```
space ( integer_expression )
```

substr

Returns the substring of "string_expression" that starts at position "integer_expression1" for "integer_expression2" characters. The first character in "string_expression" is at position 1.

Syntax

```
substr ( string_expression , integer_expression1 , integer_expression2 )
```

Example

```
substr ( [Sales (query)].[Sales staff].[Position code], 3 , 5 )
```

Result: Returns characters 3 to 7 of the position codes.

substring

Returns the substring of "string_expression" that starts at position "integer_expression1" for "integer_expression2" characters. The first character in "string_expression" is at position 1.

Syntax

```
substring ( string_expression , integer_expression1 , integer_expression2 )
```

Example

```
substring ( [Sales (query)].[Sales staff].[Position code], 3 , 5 )
```

Result: Returns characters 3 to 7 of the position codes.

truncate

Returns "string_expression" with trailing spaces removed.

Syntax

```
truncate ( string_expression )
```

ucase

Returns "string_expression" with all lowercase characters converted to uppercase.

Syntax

```
ucase ( string_expression )
```

week

Returns the week of the year in "date_expression" as an integer value (1-53), where 1 represents the first week of the year.

Syntax

```
week ( date_expression )
```

year

Returns the year from "date_expression".

Syntax

```
year ( date_expression )
```

MS Access Cast**cast_decimal**

Returns the value of "expression" cast as a decimal.

Syntax

```
cast_decimal ( expression )
```

cast_float

Returns the value of "expression" cast as a float.

Syntax

```
cast_float ( expression )
```

cast_integer

Returns the value of "expression" cast as an integer.

Syntax

```
cast_integer ( expression )
```

Example

```
cast_integer ( 84.95 )
```

Result: 84

cast_numeric

Returns "string_expression" cast as a numeric value.

Syntax

```
cast_numeric ( string_expression )
```

cast_real

Returns the value of "expression" cast as a real value.

Syntax

```
cast_real ( expression )
```

cast_smallint

Returns "expression" cast as a small integer.

Syntax

```
cast_smallint ( expression )
```

cast_varchar

Returns the value of "expression" cast as a variable character field.

Syntax

```
cast_varchar ( expression )
```

MS Access Math

log

Returns the natural logarithm of "numeric_expression".

Syntax

```
log ( numeric_expression )
```

rand

Generates a random number using "integer_expression" as a seed value.

Syntax

```
rand ( integer_expression )
```

MS Access Trigonometry

atan

Returns the arctangent of "numeric_expression" in radians. The arctangent is the angle whose tangent is "numeric_expression".

Syntax

```
atan ( numeric_expression )
```

cos

Returns the cosine of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

Syntax

```
cos ( numeric_expression )
```

sin

Returns the sine of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

Syntax

```
sin ( numeric_expression )
```

tan

Returns the tangent of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

Syntax

```
tan ( numeric_expression )
```

Netezza**ascii**

Returns a number representing the ASCII code value of the leftmost character of "string_expression"; for example, `ascii ('A')` is 65.

Syntax

```
ascii ( string_expression )
```

chr

Returns the character that has the ASCII code value specified by "integer_expression". "Integer_expression" should be between 0 and 255.

Syntax

```
chr ( integer_expression )
```

decode

Compares "expr" to each search value one by one. If "expr" is equal to a search, then it returns the corresponding result. If no match is found, it returns "default". If "default" is omitted, it returns null.

Syntax

```
decode ( expr , search , result [ , search , result ] ... [ , default ] )
```

initcap

Returns "string_expression", with the first letter of each word in uppercase, all other letters in lowercase. Words are delimited by white space or characters that are not alphanumeric.

Syntax

```
initcap ( string_expression )
```

instr

Searches "string_expression1" starting at position "integer_expression1" for the "integer_expression2" occurrence of "string_expression2". If "integer_expression1" is negative then the search is backwards from the end of "string_expression1". Returns an integer indicating the position of "string_expression2".

Syntax

```
instr ( string_expression1 , string_expression2 [ , integer_expression1 [ , integer_expression2 ] ] )
```

lpad

Returns "string_expression1" padded to length "integer_expression" with occurrences of "string_expression2". If "string_expression1" is longer than "integer_expression", the appropriate portion of "string_expression1" is returned.

Syntax

```
lpad ( string_expression1 , integer_expression [ , string_expression2 ] )
```

ltrim

Returns "string_expression1", with leading characters removed up to the first character not in "string_expression2"; for example, ltrim ('xyxXxyAB' , 'xy') returns XxyAB.

Syntax

```
ltrim ( string_expression1 [ , string_expression2 ] )
```

months_between

Returns the number of months from "date_expression1" to "date_expression2". If "date_expression1" is later than "date_expression2" then the result will be a positive number. The days and time portions of the difference are ignored, i.e., the months are not rounded, except if "date_expression1" and "date_expression2" are the last days of a month.

Syntax

```
months_between ( date_expression1 , date_expression2 )
```

next_day

Returns the datetime of the first weekday named by "string_expression" that is later than "datetime_expression". The return value has the same hours, minutes, and seconds as "datetime_expression".

Syntax

```
next_day ( datetime_expression , string_expression )
```

nvl

Returns "expression" if not null, otherwise returns "constant". Valid for "numeric_expression", "string_expression", "date_expression", and "time_expression".

Syntax

```
nvl ( expression , constant )
```

round

Returns "numeric_expression" rounded to the nearest value "integer_expression" places right of the decimal point. If "integer_expression" is negative, "numeric_expression" is rounded to the nearest absolute value "integer_expression" places to the left of the decimal point; for example, round (125, -1) rounds to 130.

Syntax

```
round ( numeric_expression [ , integer_expression ] )
```

rpadd

Returns "string_expression1" right-padded to length "integer_expression" with occurrences of "string_expression2". If "string_expression1" is longer than "integer_expression", the appropriate portion of "string_expression1" is returned. If "string_expression2" is not specified, then spaces are used.

Syntax

```
rpadd ( string_expression1 , integer_expression [ , string_expression2 ] )
```

rtrim

Returns "string_expression1", with final characters removed after the last character not in "string_expression2"; for example, rtrim ('ABxXxyx' , 'xy') returns ABxX. If "string_expression2" is not specified, the final space characters are removed.

Syntax

```
rtrim ( string_expression1 [ , string_expression2 ] )
```

substr

Returns the substring of "string_expression" that starts at position "integer_expression1". The first character in "string_expression" is at position 1. "Integer_expression2" can be used to select fewer characters; by default it selects characters to the end of the string.

Syntax

```
substr ( string_expression , integer_expression1 [ , integer_expression2 ] )
```

{current_db}

Syntax

```
{current_db}
```

{current_user}

Syntax

```
{current_user}
```

{session_user}

Syntax

```
{session_user}
```

to_char

Returns the string representation of "expression" with the format of "string_expression". "Expression" can be either a date value or a numeric value.

Syntax

```
to_char ( expression [ , string_expression ] )
```

to_date

Converts "string_expression1" to a datetime value as specified by the format "string_expression2".

Syntax

```
to_date ( string_expression1 , string_expression2 )
```

to_number

Converts "string_expression1" to a numeric value as specified by the format "string_expression2".

Syntax

```
to_number ( string_expression1 , string_expression2 )
```

translate

Returns "string_expression1", with all occurrences of each character in "string_expression2" replaced by its corresponding character in "string_expression3".

Syntax

```
translate ( string_expression1 , string_expression2 , string_expression3 )
```

date_trunc

Truncates "date_expression1" to a value as specified by the format "string_expression1".

Syntax

```
date_trunc ( string_expression1 , date_expression1 )
```

trunc

Truncates digits from "numeric_expression1" using "numeric_expression2" as the precision.

Syntax

```
trunc ( numeric_expression1 [ , numeric_expression2 ] )
```

version

Returns the "string_expression1" value of the database version.

Syntax

```
version ( )
```

Netezza Math**log**

Returns the logarithm of "numeric_expression2" to the base "numeric_expression1".

Syntax

```
log ( numeric_expression1 , numeric_expression2 )
```

Netezza Trigonometry**acos**

Returns the arccosine of "numeric_expression" in radians. The arccosine is the angle whose cosine is "numeric_expression".

Syntax

```
acos ( numeric_expression )
```

asin

Returns the arcsine of "numeric_expression" in radians. The arcsine is the angle whose sine is "numeric_expression".

Syntax

```
asin ( numeric_expression )
```

atan

Returns the arctangent of "numeric_expression" in radians. The arctangent is the angle whose tangent is "numeric_expression".

Syntax

```
atan ( numeric_expression )
```

atan2

Returns the arctangent of the x and y coordinates specified by "numeric_expression1" and "numeric_expression2", respectively, in radians. The arctangent is the angle whose tangent is "numeric_expression2" / "numeric_expression1".

Syntax

```
atan2 ( numeric_expression1 , numeric_expression2 )
```

cos

Returns the cosine of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

Syntax

```
cos ( numeric_expression )
```

degrees

Returns the degrees where "numeric_expression" is an angle expressed in radians.

Syntax

```
degrees ( numeric_expression )
```

radians

Returns the radians where "numeric_expression" is an angle expressed in degrees.

Syntax

```
radians ( numeric_expression )
```

sin

Returns the sine of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

Syntax

```
sin ( numeric_expression )
```

tan

Returns the tangent of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

Syntax

```
tan ( numeric_expression )
```

Netezza Fuzzy

le_dst

Returns a value indicating how different the two input strings are, calculated according to the Levenshtein edit distance algorithm.

Syntax

```
le_dst ( string_expression1 , string_expression2 )
```

dle_dst

Returns a value indicating how different the two input strings are, calculated according to the Damerau-Levenshtein distance algorithm

Syntax

```
dle_dst ( string_expression1 , string_expression2 )
```

Netezza Phonetic**nysiis**

Returns a Soundex representation of "string_expression" using the New York State Identification and Intelligence System (NYSIIS) variation of Soundex.

Syntax

```
nysiis ( string_expression )
```

dbl_mp

Returns a composite 32-bit value of "string_expression".

Syntax

```
dbl_mp ( string_expression )
```

pri_mp

Returns the 4 character primary metaphone string from "numeric_expression" returned by dbl_mp.

Syntax

```
pri_mp ( numeric_expression )
```

sec_mp

Returns the 4 character secondary metaphone string from "numeric_expression" returned by dbl_mp.

Syntax

```
sec_mp ( numeric_expression )
```

score_mp

Returns a score for how closely "numeric_expression" and "numeric_expression2" match.

Syntax

```
score_mp ( numeric_expression , numeric_expression2 , numeric_expression3 ,  
numeric_expression4 , numeric_expression5 , numeric_expression6 )
```

Oracle

add_months

Returns the datetime resulting from adding "integer_expression" months to "date_expression".

Syntax

```
add_months ( date_expression , integer_expression )
```

ascii

Returns a number representing the ASCII code value of the leftmost character of "string_expression".

Syntax

```
ascii ( string_expression )
```

Example

```
ascii ( 'A' )
```

Result: Returns '65'

ceil

Returns the smallest integer greater than or equal to "numeric_expression".

Syntax

```
ceil ( numeric_expression )
```

char_length

Returns the number of logical characters in "string_expression". The number of logical characters can be distinct from the number of bytes in some East Asian locales.

Syntax

```
char_length ( string_expression )
```

chr

Returns the character that has the ASCII code value specified by "integer_expression". "Integer_expression" should be between 0 and 255.

Syntax

```
chr ( integer_expression )
```

concat

Returns a string that is the result of concatenating, or joining, "string_expression1" to "string_expression2".

Syntax

```
concat ( string_expression1 , string_expression2 )
```

Example

```
concat ( [Sales (query)].[Sales staff].[First name], [Sales (query)].[Sales
staff].[Last name] )
```

Result: Returns the first name and last name; e.g., Bob Smith.

decode

Compares "expression" to each search value one by one. If "expression" is equal to a search, then it returns the corresponding result. If no match is found, it returns "default", or if "default" is omitted, it returns null.

Syntax

```
decode ( expression , search , result [ , search , result ]... [ , default ] )
```

dump

Returns internal representation of "expression" with the format of "numeric_expression1" starting from position "numeric_expression2" for "numeric_expression3" characters.

Syntax

```
dump ( expression [ , numeric_expression1 [ , numeric_expression2 [ , numeric_
expression3 ] ] ] )
```

greatest

Returns the greatest value in "expression_list".

Syntax

```
greatest ( expression_list )
```

initcap

Returns "string_expression" with the first letter of each word in uppercase and all other letters in lowercase. Words are delimited by white space or characters that are not alphanumeric.

Syntax

```
initcap ( string_expression )
```

instr

Searches "string_expression1" starting at position "integer_expression1" for the "integer_expression2" occurrence of "string_expression2". If "integer_expression1" is negative, then the search occurs backwards from the end of "string_expression1". Returns an integer indicating the position of "string_expression2".

Syntax

```
instr ( string_expression1, string_expression2 [ , integer_expression1
[ , integer_expression2 ] ] )
```

instrb

Searches "string_expression1" starting at position "integer_expression1" for the "integer_expression2" occurrence of "string_expression2". If "integer_expression1" is negative, then the search

occurs backwards from the end of "string_expression1". Returns the position (byte number) where "string_expression2" was found.

Syntax

```
instrb ( string_expression1, string_expression2 [ , integer_expression1  
[ , integer_expression2 ] ] )
```

least

Returns the least value in "expression_list".

Syntax

```
least ( expression_list )
```

length

Returns the number of characters in "string_expression".

Syntax

```
length ( string_expression )
```

lengthb

Returns the number of bytes in "string_expression".

Syntax

```
lengthb ( string_expression )
```

lpad

Returns "string_expression1" left-padded to the length defined by "integer_expression" with occurrences of "string_expression2". If "string_expression1" is longer than "integer_expression", the appropriate portion of "string_expression1" is returned.

Syntax

```
lpad ( string_expression1, integer_expression [ , string_expression2 ] )
```

ltrim

Returns "string_expression1" with leading characters removed up to the first character not in "string_expression2".

Syntax

```
ltrim ( string_expression1 [ , string_expression2 ] )
```

Example

```
ltrim ( 'xyxXxyAB' , 'xy' )
```

Result: XxyAB

months_between

Returns the number of months from "date_expression1" to "date_expression2". If "date_expression1" is later than "date_expression2" then the result will be a positive number. The days and time portion of the difference are ignored, so the months are not rounded unless "date_expression1" and "date_expression2" are the last days of a month.

Syntax

```
months_between ( date_expression1 , date_expression2 )
```

new_time

Returns the datetime in "new_timezone" for "datetime_expression" in "old_timezone". "Old_timezone" and "new_timezone" can be one of 'AST', 'ADT', 'BST', 'BDT', 'CST', 'CDT', 'EST', 'EDT', 'HST', 'HDT', 'MST', 'MDT', 'NST', 'PST', 'PDT', 'YST', or 'YDT'.

Syntax

```
new_time ( datetime_expression , old_timezone , new_timezone )
```

next_day

Returns the datetime of the first weekday named by "string_expression" that is later than "datetime_expression". The return value has the same format as "datetime_expression".

Syntax

```
next_day ( datetime_expression , string_expression )
```

nls_initcap

Returns "string_expression1" with the first letter of each word in uppercase and all other letters in lowercase. A word begins after any character other than a letter. Thus, in addition to a blank space, symbols such as commas, periods, and colons can introduce a new word. "String_expression2" specifies the sorting sequence.

Syntax

```
nls_initcap ( string_expression1 [ , string_expression2 ] )
```

nls_lower

Returns "string_expression1" with all letters in lowercase. "String_expression2" specifies the sorting sequence.

Syntax

```
nls_lower ( string_expression1 [ , string_expression2 ] )
```

nls_upper

Returns "string_expression1" with all letters in uppercase. "String_expression2" specifies the sorting sequence.

Syntax

```
nls_upper ( string_expression1 [ , string_expression2 ] )
```

nvl

Returns "expression" unless it is null. If "expression" is null, returns "constant". Valid for "numeric_expression", "string_expression", "date_expression", and "time_expression".

Syntax

```
nvl ( expression , constant )
```

Example

```
nvl ( [Unit sale price] , [Unit price] )
```

Result: Returns the unit sale price, or returns the unit price if the unit sale price is NULL.

replace

Replaces all occurrences of "string_expression2" in "string_expression1" with "string_expression3". If "string_expression3" is not specified, then it removes all occurrences of "string_expression2".

Syntax

```
replace ( string_expression1 , string_expression2 [ , string_expression3 ] )
```

round

Returns "numeric_expression" rounded to the nearest value "integer_expression" places right of the decimal point. If "integer_expression" is negative, "numeric_expression" is rounded to the nearest absolute value "integer_expression" places to the left of the decimal point. Rounding takes place before data formatting is applied.

Syntax

```
round ( numeric_expression [ , integer_expression ] )
```

Example

```
round ( 125 , -1 )
```

Result: Returns 130

rpad

Returns "string_expression1" right-padded to length "integer_expression" with occurrences of "string_expression2". If "string_expression1" is longer than "integer_expression", the appropriate portion of "string_expression1" is returned. If "string_expression2" is not specified, then occurrences of "string_expression2" are replaced with spaces.

Syntax

```
rpad ( string_expression1 , integer_expression [ , string_expression2 ] )
```

rtrim

Returns "string_expression1" with the final characters removed after the last character not in "string_expression2". If "string_expression2" is not specified, the final space characters are removed.

Syntax

```
rtrim ( string_expression1 [ , string_expression2 ] )
```

Example

```
rtrim ( 'ABxXxyx' , 'xy' )
```

Result: Returns 'ABxX'

sign

Returns an indicator of the sign of "numeric_expression", +1 if positive, 0 if zero, or -1 if negative.

Syntax

```
sign ( numeric_expression )
```

soundex

Returns a character string containing the phonetic representation of "string_expression".

Syntax

```
soundex ( string_expression )
```

substr

Returns the substring of "string_expression" that starts at position "integer_expression1" for "integer_expression2" characters or to the end of "string_expression" if "integer_expression2" is omitted. The first character in "string_expression" is at position 1.

Syntax

```
substr ( string_expression , integer_expression1 [ , integer_expression2 ] )
```

Example

```
substr ( [Sales (query)].[Sales staff].[Position code], 3 , 5 )
```

Result: Returns characters 3 to 7 of the position codes.

substrb

Returns the substring of "string_expression" that starts at position "numeric_expression1" and ends after "numeric_expression2" bytes. The first byte in "string_expression" is at position 1. If you omit "numeric_expression2", returns the substring of "string_expression" that starts at position "numeric_expression1" and ends at the end of "string_expression".

Syntax

```
substrb ( string_expression , numeric_expression1 [ , numeric_expression2 ] )
```

Example

```
substrb ( [Sales (query)].[Sales staff].[Position code], 3 , 5 )
```

Result: Returns characters 3 to 7 of the position codes.

{sysdate}

Returns a datetime value representing the current date and time of the computer that the database software runs on.

Syntax

```
{ sysdate }
```

to_char

Returns the string representation of "expression" with the format of "string_expression". "Expression" can be either a date value or a numeric value.

Syntax

```
to_char ( expression [ , string_expression ] )
```

to_date

Converts "string_expression1" to a datetime value as specified by the format "string_expression2". "String_expression3" specifies the format elements, such as language.

Syntax

```
to_date ( string_expression1 [ , string_expression2 [ , string_expression3 ] ] )
```

to_number

Converts "string_expression1" to a numeric value as specified by the format "string_expression2". "String_expression3" specifies the format elements, such as currency information.

Syntax

```
to_number ( string_expression1 , string_expression2 , string_expression3 )
```

translate

Returns "string_expression1" with all occurrences of each character in "string_expression2" replaced by the corresponding character in "string_expression3".

Syntax

```
translate ( string_expression1 , string_expression2 , string_expression3 )
```

trunc

Truncates "date_expression" using the format specified by "string_expression". For example, if "string_expression" is 'year', then "date_expression" is truncated to the first day of the year.

Syntax

```
trunc ( date_expression , string_expression )
```

Example

```
trunc ( 2003-08-22 , 'year' )
```

Result: Returns 2003-01-01.

trunc

Truncates digits from "numeric_expression1" using "numeric_expression2" as the precision.

Syntax

```
trunc ( numeric_expression1 , numeric_expression2 )
```

{user}

Returns the username of the current Oracle user.

Syntax

```
{ user }
```

vsize

Returns the number of bytes in the internal representation of "expression". "Expression" must be a string expression.

Syntax

```
vsize ( expression )
```

Oracle Math**log**

Returns the logarithm of "numeric_expression2" to the base "numeric_expression1".

Syntax

```
log ( numeric_expression1 , numeric_expression2 )
```

Oracle Trigonometry**acos**

Returns the arccosine of "numeric_expression" in radians. The arccosine is the angle whose cosine is "numeric_expression".

Syntax

```
acos ( numeric_expression )
```

asin

Returns the arcsine of "numeric_expression" in radians. The arcsine is the angle whose sine is "numeric_expression".

Syntax

```
asin ( numeric_expression )
```

atan

Returns the arctangent of "numeric_expression" in radians. The arctangent is the angle whose tangent is "numeric_expression".

Syntax

```
atan ( numeric_expression )
```

atan2

Returns the arctangent of the x and y coordinates specified by "numeric_expression1" and "numeric_expression2", respectively, in radians. The arctangent is the angle whose tangent is "numeric_expression2" / "numeric_expression1".

Syntax

```
atan2 ( numeric_expression1 ,numeric_expression2 )
```

cos

Returns the cosine of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

Syntax

```
cos ( numeric_expression )
```

cosh

Returns the hyperbolic cosine of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

Syntax

```
cosh ( numeric_expression )
```

sin

Returns the sine of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

Syntax

```
sin ( numeric_expression )
```

sinh

Returns the hyperbolic sine of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

Syntax

```
sinh ( numeric_expression )
```

tan

Returns the tangent of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

Syntax

```
tan ( numeric_expression )
```

tanh

Returns the hyperbolic tangent of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

Syntax

```
tanh ( numeric_expression )
```

Red Brick**ceil**

Returns the smallest integer greater than or equal to "numeric_expression" or "string_expression". Note that "string_expression" must represent a valid numeric value.

Syntax

```
ceil ( numeric_expression|string_expression )
```

concat

Returns a string that is the result of concatenating, or joining, "string_expression1" to "string_expression2".

Syntax

```
concat ( string_expression1 , string_expression2 )
```

Example

```
concat ( [Sales (query)].[Sales staff].[First name], [Sales (query)].[Sales  
staff].[Last name] )
```

Result: Returns the first name and last name; e.g., Bob Smith.

{current_user}

Returns the database username (authorization ID) of the current user.

Syntax

```
{ current_user }
```

date

Returns a date value. "Expression" can be either characters or a timestamp.

Syntax

```
date ( expression )
```

dateadd

Adds "interval" to "datetime_expression" and returns a result that is the same datetime data type as "datetime_expression". "Datepart" refers to the year, month, day, hour, minute, second. "Interval" must be an integer and "datetime_expression" can be a date, time, or timestamp.

Syntax

```
dateadd ( { datepart } , interval, datetime_expression )
```

datediff

Determines the difference between two datetime expressions and returns an integer result in "datepart" units. "Datepart" refers to a year, month, day, hour, minute, or second. "Datetime_expression1" and "datetime_expression2" can be dates, times, or timestamps.

Syntax

```
datediff ( { datepart } , datetime_expression1, datetime_expression2 )
```

datetime

Extracts "datepart" of "datetime_expression" and returns its value as a character string. "Datepart" refers to a year, month, day, hour, minute, or second. "Datetime_expression" can be a date, a time, or a timestamp.

Syntax

```
datetime ( { datepart } , datetime_expression )
```

dec

Converts "expression" to a decimal value with the data type decimal (precision, scale). The default value of precision is 9. The default value of scale is 0.

Syntax

```
dec ( expression , [ precision , scale ] )
```

decimal

Converts "expression" to a decimal value with the data type decimal (precision, scale). The default value of precision is 9. The default value of scale is 0.

Syntax

```
decimal ( expression , [ precision , scale ] )
```

decode

Compares and converts "expression" to another value. If "expression" matches "target", it is replaced, otherwise it is replaced by "default" or null if no default is specified. The expressions can be any data type as long as they are all the same data type.

Syntax

```
decode ( expression , target , replacement [ ,default ] )
```

float

Converts "numeric_expression" into a double-precision floating-point value.

Syntax

```
float ( numeric_expression )
```

ifnull

Tests "expression" for missing values and replaces each one with "substitute". If "expression" is null, "substitute" is returned, otherwise it returns the value of "expression". The expressions can be any data type as long as they are all the same data type.

Syntax

```
ifnull ( expression, substitute )
```

int

Converts "numeric_expression" into an integer value and returns an integer value. If "numeric_expression" is null, it returns null.

Syntax

```
int ( numeric_expression )
```

integer

Converts "numeric_expression" into an integer value and returns an integer value. If "numeric_expression" is null, it returns null.

Syntax

```
integer ( numeric_expression )
```

Example

```
integer ( 84.95 )
```

Result: 85

length

Returns an integer result specifying the number of characters in "string_expression". If "string_expression" is null, it returns null.

Syntax

```
length ( string_expression )
```

lengthb

Returns an integer result specifying the number of bytes in "string_expression". If "string_expression" is null, it returns null.

Syntax

```
lengthb ( string_expression )
```

ltrim

Removes leading blanks from "string_expression". If "string_expression" is null, it returns null.

Syntax

```
ltrim ( string_expression )
```

nullif

Returns null if both "expression1" and "expression2" have the same value. If they have different values, the value of "expression1" is returned. "Expression1" and "expression2" can be any data type as long as they are the same data type.

Syntax

```
nullif ( expression1 , expression2 )
```

positionb

Returns an integer that is relative to the beginning byte position of "string_expression1" in "string_expression2". If "string_expression1" is not located, the result is 0. If "string_expression1" is of zero length, the result is 1. If "string_expression1" is null, an error message is returned. If "string_expression2" is null, the result is 0.

Syntax

```
positionb ( string_expression1, string_expression2 )
```

real

Returns a real value. If "numeric_expression" is null, it returns null.

Syntax

```
real ( numeric_expression )
```

round

Returns "numeric_expression" rounded to the nearest value "integer_expression" places to the right of the decimal point. If "integer_expression" is negative, "numeric_expression" is rounded to the nearest absolute value "integer_expression" places to the left of the decimal point. Rounding takes place before data formatting is applied.

Syntax

```
round ( numeric_expression , integer_expression )
```

Example

```
round (125, -1)
```

Result: 130

rtrim

Removes trailing blanks from "string_expression". If "string_expression" is null, it returns null.

Syntax

```
rtrim ( string_expression )
```

Example

```
rtrim ( [Sales (query)].[Sales staff].[Last name] )
```

Result: Returns last names with any spaces at the end of the name removed.

sign

Determines the sign of "numeric_expression", and returns 1 for a positive value, -1 for a negative value, and 0 for zero.

Syntax

```
sign ( numeric_expression )
```

string

Converts "expression" to a character string. "Expression" can be either numeric or datetime.

Syntax

```
string ( expression [ , length [ , scale ] ] )
```

substr

Returns a substring of "string_expression" that begins at position "start_integer" and continues for "length_integer" characters. If "length_integer" is not specified, a substring from "start_integer" to the end of "string_expression" is returned.

Syntax

```
substr ( string_expression , start_integer , length_integer )
```

Example

```
substr ( [Sales (query)].[Sales staff].[Position code], 3 , 5 )
```

Result: Returns characters 3 to 7 of the position codes.

substrb

Returns a substring of "string_expression" that begins at position "start_integer" and continues for "length_integer" bytes. If "length_integer" is not specified, a substring from "start_integer" to the end of "string_expression" is returned.

Syntax

```
substrb ( string_expression , start_integer , length_integer )
```

time

Creates a time value from "expression", which can be a character string or a time-stamp data type expression.

Syntax

```
time ( expression )
```

timestamp

Creates a time-stamp value from "timestamp_expression", which is a character string.

Syntax

```
timestamp ( timestamp_expression )
```

timestamp

Creates a time-stamp value from "time_expression" and "date_expression". If either "time_expression" or "date_expression" is null, the resulting time-stamp expression is also null.

Syntax

```
timestamp ( date_expression , time_expression )
```

to_char

Converts "source_date" to the character string specified by "format_string". "Source_date" can be a date, time, or timestamp data type.

Syntax

```
to_char ( source_date, format_string )
```

SQL Server

ascii

Returns a number representing the ascii code value of the leftmost character of "string_expression".

Syntax

```
ascii ( string_expression )
```

Example

```
ascii ( 'A' )
```

Result: 65

char

Returns the character that has the ASCII code value specified by "integer_expression". "Integer_expression" should be between 0 and 255.

Syntax

```
char ( integer_expression )
```

Example

```
char ( 65 )
```

Result: A

charindex

Searches "string_expression2" for the first occurrence of "string_expression1" and returns an integer. "Start_location" is the character position to start searching for "string_expression1" in "string_expression2". If "start_location" is not specified, is a negative number, or is zero, the search starts at the beginning of "string_expression2".

Syntax

```
charindex ( string_expression1 , string_expression2 [ , start_location ] )
```


{current_user}

Returns the name of the current user.

Syntax

```
{ current_user }
```

datalength

Returns the length in bytes of "string_expression".

Syntax

```
datalength ( string_expression )
```

dateadd

Returns the date resulting from the addition of "integer_expression" units (indicated by "datepart" (day, month, year)) to "date_expression".

Syntax

```
dateadd ( { datepart } , integer_expression , date_expression )
```

datediff

Returns the number of "datepart" (day, month, year) units between "date_expression1" and "date_expression2".

Syntax

```
datediff ( {datepart} , date_expression1 , date_expression2 )
```

Example

```
datediff ( {yy} , 1984-01-01 , 1997-01-01 )
```

Result: 13

datetimeame

Returns "datepart" from "date_expression", which can be a datetime, smalldatetime, date, or time value as an ASCII string. Note that "datepart" must be a keyword representing a datepart or its abbreviation recognized by Microsoft® SQL Server and must be enclosed in curly brackets.

Syntax

```
datetimeame ( ' { ' datepart ' } ' , date_expression )
```

Example

```
datetimeame ( {mm} , 2000-01-01 )
```

Result: January

datepart

Returns part of "date_expression" (for example, the month) as an integer. "date_expression" can be a datetime, smalldatetime, date, or time value. Note that "datepart" must be a keyword repre-

senting a datepart or its abbreviation recognized by Microsoft® SQL Server and must be enclosed in curly brackets.

Syntax

```
datepart ( ' { ' datepart ' } ' , date_expression )
```

Example

```
datepart ( {wk} , 2000-01-01 )
```

Result: 1 (first week of the year)

day

Returns the day portion of "date_expression". Same as extract (day from date_expression).

Syntax

```
day ( date_expression )
```

difference

Returns an integer value representing the difference between the values returned by the data source-specific soundex function for "string_expression1" and "string_expression2". The value returned ranges from 0 to 4, with 4 indicating the best match. Note that 4 does not mean that the strings are equal.

Syntax

```
difference ( string_expression1 , string_expression2 )
```

Example 1

```
difference ([Sales target (query)].[Sales Staff].[First name],[Sales (query)].  
[Retailers].[Contact first name])
```

Result: 0

Example 2

```
difference ([Sales target (query)].[Sales Staff].[First name],[Sales target  
(query)].[Sales Staff].[First name])
```

Result: 4

getdate

Returns a datetime value representing the current date and time of the computer that the database software runs on.

Syntax

```
getdate ( )
```

left

Returns the leftmost "integer_expression" characters of "string_expression".

Syntax

```
left ( string_expression , integer_expression )
```

Example

```
left ( [Sales (query)].[Sales staff].[Last name] , 3 )
```

Result: Returns the first three characters of each last name.

ltrim

Returns "string_expression" with leading spaces removed.

Syntax

```
ltrim ( string_expression )
```

month

Returns the month portion of "date_expression". Same as extract (month from date_expression).

Syntax

```
month ( date_expression )
```

patindex

Returns an integer that represents the starting position of the first occurrence of "string_expression1" in the "string_expression2". Returns 0 if "string_expression1" is not found. The % wildcard character must precede and follow "string_expression1", except when searching for first or last characters.

Syntax

```
patindex ( string_expression1 , string_expression2 )
```

Example

```
patindex ( '%po%', 'Report' )
```

Result: 3

replace

Replaces all occurrences of "string_expression2" in "string_expression1" with "string_expression3".

Syntax

```
replace ( string_expression1 , string_expression2 , string_expression3 )
```

replicate

Returns a string consisting of "string_expression" repeated "integer_expression" times.

Syntax

```
replicate ( string_expression , integer_expression )
```

reverse

Returns "string_expression" in reverse order.

Syntax

```
reverse ( string_expression )
```

right

Returns the rightmost "integer_expression" characters of "string_expression".

Syntax

```
right ( string_expression , integer_expression )
```

round

Returns "numeric_expression" rounded to the nearest value "integer_expression" places to the right of the decimal point. Rounding takes place before data formatting is applied.

Syntax

```
round ( numeric_expression , integer_expression )
```

Example

```
round (125, -1)
```

Result: 130

rtrim

Returns "string_expression" with trailing spaces removed.

Syntax

```
rtrim ( string_expression )
```

Example

```
rtrim ( [Sales (query)].[Sales staff].[Last name] )
```

Result: Returns last names with any spaces at the end of the name removed.

sign

Returns an indicator of the sign "numeric_expression": +1 if "numeric_expression" is positive, 0 if zero or -1 if negative.

Syntax

```
sign ( numeric_expression )
```

soundex

Returns a four character string representing the sound of the words in "string_expression".

Syntax

```
soundex ( string_expression )
```

space

Returns a string consisting of "integer_expression" spaces.

Syntax

```
space ( integer_expression )
```

str

Returns a string representation of "numeric_expression" where "integer_expression1" is the length of the string returned and "integer_expression2" is the number of decimal digits.

Syntax

```
str ( numeric_expression [ , integer_expression1 [ , integer_expression2 ] ] )
```

stuff

Returns a string where "integer_expression2" characters have been deleted from "string_expression1" beginning at "integer_expression1", and where "string_expression2" has been inserted into "string_expression1" at its start. The first character in a string is at position 1.

Syntax

```
stuff ( string_expression1 , integer_expression1 , integer_expression2 , string_expression2 )
```

year

Returns the year portion of "date_expression". Same as extract (year from date_expression).

Syntax

```
year ( date_expression )
```

SQL Server Math**log**

Returns the natural logarithm of "numeric_expression".

Syntax

```
log ( numeric_expression )
```

log10

Returns the base ten logarithm of "numeric_expression".

Syntax

```
log10 ( numeric_expression )
```

pi

Returns the constant value of pi as a floating point value.

Syntax

```
pi ( )
```

rand

Generates a random number using "integer_expression" as the seed value.

Syntax

```
rand ( integer_expression )
```

SQL Server Trigonometry

acos

Returns the arccosine of "numeric_expression" in radians. The arccosine is the angle whose cosine is "numeric_expression".

Syntax

```
acos ( numeric_expression )
```

asin

Returns the arcsine of "numeric_expression" in radians. The arcsine is the angle whose sine is "numeric_expression".

Syntax

```
asin ( numeric_expression )
```

atan

Returns the arctangent of "numeric_expression" in radians. The arctangent is the angle whose tangent is "numeric_expression".

Syntax

```
atan ( numeric_expression )
```

atn2

Returns the arctangent of the x and y coordinates specified by "numeric_expression1" and "numeric_expression2", respectively, in radians. The arctangent is the angle whose tangent is "numeric_expression1".

Syntax

```
atn2 ( numeric_expression1, numeric_expression2 )
```

cos

Returns the cosine of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

Syntax

```
cos ( numeric_expression )
```

cot

Returns the cotangent of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

Syntax

```
cot ( numeric_expression )
```

degrees

Returns "numeric_expression" radians converted to degrees.

Syntax

```
degrees ( numeric_expression )
```

radians

Returns the number of radians converted from "numeric_expression" degrees.

Syntax

```
radians ( numeric_expression )
```

sin

Returns the sine of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

Syntax

```
sin ( numeric_expression )
```

tan

Returns the tangent of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

Syntax

```
tan ( numeric_expression )
```

Teradata**account**

Returns the account string for the current user.

Syntax

```
{account}
```

add_months

Returns the date or the datetime resulting from adding "integer_expression" months to "date_expression" or "datetime_expression".

Syntax

```
add_months ( date_expression|datetime_expression , integer_expression )
```

bytes

Returns the number of bytes contained in "byte_expression". "Byte_expression" is restricted to BYTE or VARBYTE.

Syntax

```
bytes ( byte_expression )
```

case_n

Evaluates "condition_expression_list" and returns the position of the first true condition, provided that no prior condition in the list evaluates to unknown. The keywords must be enclosed in curly brackets. No case is an optional condition that evaluates to true if every expression in the list evaluates to false. No case or unknown is an optional condition that evaluates to true if every expression in the list evaluates to false, or if an expression evaluates to unknown and all prior conditions in the list evaluate to false. Unknown is an optional condition that evaluates to true if an expression evaluates to unknown and all prior conditions in the list evaluate to false.

Syntax

```
case_n ( condition_expression_list [ , NO CASE|UNKNOWN|NO CASE OR UNKNOWN  
[ , UNKNOWN ] ] )
```

char2hexint

Returns the hexadecimal representation for "string_expression".

Syntax

```
char2hexint ( string_expression )
```

characters

Returns an integer value representing the number of logical characters or bytes contained in "string_expression".

Syntax

```
characters ( string_expression )
```

database

Returns the name of the default database for the current user.

Syntax

```
{database}
```

date

Returns the current date.

Syntax

```
{date}
```


format

Returns the declared format for "expression" as a variable character string of up to 30 characters.

Syntax

```
format ( expression )
```

index

Returns the starting position of "string_expression2" in "string_expression1".

Syntax

```
index ( string_expression1 , string_expression2 )
```

log

Computes the base 10 logarithm of "numeric_expression". "Numeric_expression" must be a non-zero, positive, numeric expression.

Syntax

```
log ( numeric_expression )
```

nullif

Returns null if "scalar_expression1" and "scalar_expression2" are equal. Otherwise, it returns "scalar_expression1". "Scalar_expression1" and "scalar_expression2" can be any data type.

Syntax

```
nullif ( scalar_expression1 , scalar_expression2 )
```

nullifzero

If "numeric_expression" is zero, converts it to null to avoid division by zero.

Syntax

```
nullifzero ( numeric_expression )
```

profile

Returns the current profile for the session or null if none.

Syntax

```
{profile}
```

random

Returns a random integer number for each row of the results table. "Lower_bound" and "upper_bound" are integer constants. The limits for "lower_bound" and "upper_bound" range from -2147483648 to 2147483647 inclusive. "Upper_bound" must be greater than or equal to "lower_bound".

Syntax

```
random ( lower_bound , upper_bound )
```

role

Returns the current role for the session or null if none.

Syntax

```
{role}
```

session

Returns the number of the session for the current user.

Syntax

```
{session}
```

soundex

Returns a character string that represents the Soundex code for "string_expression".

Syntax

```
soundex ( string_expression )
```

substr

Returns the substring of "string_expression" that starts at position "integer_expression1" for "integer_expression2" characters. The first character in "string_expression" is at position 1. If "integer_expression2" is omitted, returns the substring of "string_expression" that starts at position "integer_expression1" and ends at the end of "string_expression".

Syntax

```
substr ( string_expression , integer_expression1 [ , integer_expression2 ] )
```

Example

```
substr ( [Sales (query)].[Sales staff].[Position code], 3 , 5 )
```

Result: Returns characters 3 to 7 of the position codes.

time

Returns the current time based on a 24-hour day.

Syntax

```
{time}
```

type

Returns the data type defined for "expression".

Syntax

```
type ( expression )
```

user

Returns the user name of the current user.

Syntax

```
{user}
```

vargraphic

Returns a character string that represents the vargraphic code for "string_expression".

Syntax

```
vargraphic ( string_expression )
```

zeroifnull

Converts data from null to 0 to avoid errors created by a null value. If "numeric_expression" is not null, returns the value of "numeric_expression". If "numeric_expression" is a character string, it is converted to a numeric value of float data type. If "numeric_expression" is null or zero, it returns zero.

Syntax

```
zeroifnull ( numeric_expression )
```

Teradata Trigonometry**acos**

Returns the arccosine of "numeric_expression" in radians. The arccosine is the angle whose cosine is "numeric_expression". "Numeric_expression" must be between -1 and 1, inclusive.

Syntax

```
acos ( numeric_expression )
```

acosh

Returns the inverse hyperbolic cosine of "numeric_expression" where "numeric_expression" can be any real number equal to or greater than 1.

Syntax

```
acosh ( numeric_expression )
```

asin

Returns the arcsine of "numeric_expression" in radians. The arcsine is the angle whose sine is "numeric_expression". "Numeric_expression" must be between -1 and 1, inclusive.

Syntax

```
asin ( numeric_expression )
```

asinh

Returns the inverse hyperbolic sine of "numeric_expression" where "numeric_expression" can be any real number.

Syntax

```
asinh ( numeric_expression )
```

atan

Returns the arctangent of "numeric_expression" in radians where the arctangent is the angle whose tangent is "numeric_expression".

Syntax

```
atan ( numeric_expression )
```

atan2

Returns the arctangent of the x and y coordinates specified by "numeric_expression1" and "numeric_expression2", respectively, in radians. The returned angle will be between - and π radians, excluding π .

Syntax

```
atan2 ( numeric_expression1, numeric_expression2 )
```

atanh

Returns the inverse hyperbolic tangent of "numeric_expression" where "numeric_expression" can be any real number between 1 and -1, excluding 1 and -1.

Syntax

```
atanh ( numeric_expression )
```

cos

Returns the cosine of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

Syntax

```
cos ( numeric_expression )
```

cosh

Returns the hyperbolic cosine of "numeric_expression" where "numeric_expression" can be any real number.

Syntax

```
cosh ( numeric_expression )
```

sin

Returns the sine of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

Syntax

```
sin ( numeric_expression )
```

sinh

Returns the hyperbolic sine of "numeric_expression" where "numeric_expression" can be any real number.

Syntax

```
sinh ( numeric_expression )
```

tan

Returns the tangent of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

Syntax

```
tan ( numeric_expression )
```

tanh

Returns the hyperbolic tangent of "numeric_expression" where "numeric_expression" can be any real number.

Syntax

```
tanh ( numeric_expression )
```

SAP BW**SAP BW Trigonometry****arccos**

Returns the arccosine of "numeric_expression" in radians. The arccosine is the angle whose cosine is "numeric_expression".

Syntax

```
arccos ( numeric_expression )
```

arcsin

Returns the arcsine of "numeric_expression" in radians. The arcsine is the angle whose sine is "numeric_expression".

Syntax

```
arcsin ( numeric_expression )
```

arctan

Returns the arctangent of "numeric_expression" in radians. The arctangent is the angle whose tangent is "numeric_expression".

Syntax

```
arctan ( numeric_expression )
```

cos

Returns the cosine of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

Syntax

```
cos ( numeric_expression )
```

sin

Returns the sine of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

Syntax

```
sin ( numeric_expression )
```

tan

Returns the tangent of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

Syntax

```
tan ( numeric_expression )
```

coshyp

Returns the hyperbolic cosine of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

Syntax

```
coshyp ( numeric_expression )
```

sinhyp

Returns the hyperbolic sine of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

Syntax

```
sinhyp ( numeric_expression )
```

tanhyp

Returns the hyperbolic tangent of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

Syntax

```
tanhyp ( numeric_expression )
```

SAP BW Math

log10

Returns the base ten logarithm of "numeric_expression".

Syntax

```
log10 ( numeric_expression )
```

Sybase**ascii**

Returns a number representing the ascii code value of the leftmost character of "string_expression".

Syntax

```
ascii ( string_expression )
```

Example

```
ascii ( 'A' )
```

Result: 65

char

Converts "integer_expression" to a character value. Char is usually used as the inverse of ascii where "integer_expression" must be between 0 and 255. If the resulting value is the first byte of a multibyte character, the character may be undefined.

Syntax

```
char ( integer_expression )
```

charindex

Returns an integer that represents the starting position of "string_expression1" within "string_expression2". If "string_expression1" is not found, zero is returned. If "string_expression1" contains wildcard characters, charindex treats them as literals.

Syntax

```
charindex ( string_expression1 , string_expression2 )
```

datalength

Returns the length in bytes of "string_expression".

Syntax

```
datalength ( string_expression )
```

dateadd

Returns the date resulting from adding "integer_expression" units indicated by datepart (day, month, year) to "date_expression". Note that "datepart" must be enclosed in curly brackets.

Syntax

```
dateadd ( ' { ' datepart ' } ' , integer_expression , date_expression )
```

Example

```
dateadd ( {dd} , 16 , 1997-06-16 )
```

Result: Jul 2, 1997

datediff

Returns the number of units indicated by "datepart" (day, month, year) between "date_expression1" and "date_expression2". Note that "datepart" must be enclosed in curly brackets.

Syntax

```
datediff ( ' { ' datepart ' } ' , date_expression1 , date_expression2 )
```

Example

```
datediff ( {yy} , 1984-01-01 , 1997-01-01 )
```

Result: 13

datetimeame

Returns "datepart" of "date_expression" as an ASCII string. "Date_expression" can be a datetime, smalldatetime, date, or time value. Note that "datepart" must be enclosed in curly brackets.

Syntax

```
datetimeame ( ' { ' datepart ' } ' , date_expression )
```

Example

```
datetimeame ( {mm} , 1999-05-01 )
```

Result: May

datepart

Returns "datepart" of "date_expression" as an integer. "Date_expression" can be a datetime, smalldatetime, date, or time value. Note that the datepart argument must be enclosed in curly brackets.

Syntax

```
datepart ( ' { ' datepart ' } ' , date_expression )
```

Example

```
datepart ( {mm} , 1999-05-01 )
```

Result: 5

day

Returns the day of the month (1-31) from "date_expression".

Syntax

```
day ( date_expression )
```

difference

Returns an integer value representing the difference between the values returned by the data source-specific soundex function for "string_expression1" and "string_expression2". The value that is

returned ranges from 0 to 4, with 4 indicating the best match. Note that 4 does not mean that the strings are equal.

Syntax

```
difference ( string_expression1 , string_expression2 )
```

Example 1

```
difference ([Sales target (query)].[Sales staff].[First name],[Sales (query)].[Retailers].[Contact first name])
```

Result: 0

Example 2

```
difference ([Sales target (query)].[Sales staff].[First name],[Sales target (query)].[Sales staff].[First name])
```

Result: 4

getdate

Returns current system date and time.

Syntax

```
getdate ()
```

left

Returns the leftmost "integer_expression" characters of "string_expression".

Syntax

```
left ( string_expression , integer_expression )
```

Example

```
left ( [Sales (query)].[Sales staff].[Last name] , 3 )
```

Result: Returns the first three characters of each last name.

ltrim

Returns "string_expression" with any leading spaces removed.

Syntax

```
ltrim ( string_expression )
```

month

Returns the month number (1-12) from "date_expression".

Syntax

```
month ( date_expression )
```

patindex

Returns an integer representing the starting position of the first occurrence of "string_expression1" in "string_expression2" or returns 0 if "string_expression1" is not found. By default, patindex returns the offset in characters. The offset can be returned in bytes by setting the return type to bytes. The % wildcard character must precede and follow the pattern in "string_expression1", except when searching for first or last characters.

Syntax

```
patindex ( string_expression1 , string_expression2 [ using {bytes|chars|characters} ] )
```

rand

Returns a random float value between 0 and 1, using the optional "integer_expression" as a seed value.

Syntax

```
rand ( integer_expression )
```

replicate

Returns a string with the same datatype as "string_expression", containing the same expression repeated "integer_expression" times or as many times as will fit into a 225-byte space, whichever is less.

Syntax

```
replicate ( string_expression , integer_expression )
```

reverse

Returns the reverse of "string_expression".

Syntax

```
reverse ( string_expression )
```

right

Returns the rightmost "integer_expression" characters of "string_expression".

Syntax

```
right ( string_expression , integer_expression )
```

round

Returns "numeric_expression" rounded to the nearest value "integer_expression" places to the right of the decimal point. Rounding takes place before data formatting is applied.

Syntax

```
round ( numeric_expression, integer_expression )
```

Example

```
round (125, -1)
```

Result: 130

rtrim

Returns "string_expression" with trailing spaces removed.

Syntax

```
rtrim ( string_expression )
```

Example

```
rtrim ( [Sales (query)].[Sales staff].[Last name] )
```

Result: Returns last names with any spaces at the end of the name removed.

soundex

Returns a four-character soundex code for character strings that are composed of a contiguous sequence of valid single- or double-byte Roman letter.

Syntax

```
soundex ( string_expression )
```

space

Returns a string with "integer_expression" single-byte spacing.

Syntax

```
space ( integer_expression )
```

str

Returns a string representation of "numeric_expression". "Integer_expression1" is the length of the returned string and has a default setting of 10. "Integer_expression2" is the number of decimal digits and has a default setting of 0. Both are optional values.

Syntax

```
str ( numeric_expression [ , integer_expression1 [ , integer_expression2 ] ] )
```

stuff

Deletes "integer_expression2" characters from "string_expression1" starting at "integer_expression1", and inserts "string_expression2" into "string_expression1" at that position. To delete characters without inserting other characters, "string_expression2" should be null and not " ", which indicates a single space.

Syntax

```
stuff ( string_expression1 , integer_expression1 , integer_expression2 , string_expression2 )
```

substring

Returns the substring of "string_expression" that starts at position "integer_expression1". "Integer_expression2" specifies the number of characters in the substring.

Syntax

```
substring ( string_expression , integer_expression1 , integer_expression2 )
```

Example

```
substring ( [Sales (query)].[Sales staff].[Position code], 3 , 5 )
```

Result: Returns characters 3 to 7 of the position codes.

to_unichar

Returns a unichar expression with the value "integer_expression". If "integer_expression" is in the range 0xD800..0xDFFF, the operation is aborted. If the "integer_expression" is in the range 0..0xFFFF, a single Unicode value is returned. If "integer_expression" is in the range 0x10000..0x10FFFF, a surrogate pair is returned.

Syntax

```
to_unichar ( integer_expression )
```

uhighsurr

Returns 1 if the Unicode value at "integer_expression" is the high half of a surrogate pair (which should appear first in the pair). Otherwise, it returns 0. This function allows you to write explicit code for surrogate handling. Particularly, if a substring starts on a Unicode character where uhighsurr () is true, extract a substring of at least 2 Unicode values, as substr() does not extract just 1. Substr () does not extract half of a surrogate pair.

Syntax

```
uhighsurr ( string_expression , integer_expression )
```

ulowsurr

Returns 1 if the Unicode value at "integer_expression" is the low half of a surrogate pair (which should appear second in the pair). Otherwise, it returns 0. This function allows you to explicitly code around the adjustments performed by substr (), stuff (), and right (). Particularly, if a substring ends on a Unicode value where ulowsurr () is true, extract a substring of 1 less characters (or 1 more), since substr () does not extract a string that contains an unmatched surrogate pair.

Syntax

```
ulowsurr ( string_expression , integer_expression )
```

uscalar

Returns the Unicode scalar value for the first Unicode character in "string_expression". If the first character is not the high-order half of a surrogate pair, then the value is in the range 0..0xFFFF. If the first character is the high-order half of a surrogate pair, a second value must be a low-order half, and the return value is in the range 0x10000..0x10FFFF. If this function is called on a Unicode character expression containing an unmatched surrogate half, the operation is aborted.

Syntax

```
uscalar ( string_expression )
```

year

Returns the year from date_expression.

Syntax

```
year ( date_expression )
```

Sybase Math**log**

Returns the natural logarithm of "numeric_expression".

Syntax

```
log ( numeric_expression )
```

log10

Returns the base ten logarithm of "numeric_expression".

Syntax

```
log10 ( numeric_expression )
```

pi

Returns the constant value of pi as a floating point value.

Syntax

```
pi ( )
```

sign

Returns an indicator denoting the sign of "numeric_expression": +1 if "numeric_expression" is positive, 0 if "numeric_expression" is zero, or -1 if "numeric_expression" is negative.

Syntax

```
sign ( numeric_expression )
```

Sybase Trigonometry**acos**

Returns the arccosine of "numeric_expression" in radians. The arccosine is the angle whose cosine is "numeric_expression".

Syntax

```
acos ( numeric_expression )
```

asin

Returns the arcsine of "numeric_expression" in radians. The arcsine is the angle whose sine is "numeric_expression".

Syntax

```
asin ( numeric_expression )
```

atan

Returns the arctangent of "numeric_expression" in radians. The arctangent is the angle whose tangent is "numeric_expression".

Syntax

```
atan ( numeric_expression )
```

tan

Returns the tangent of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

Syntax

```
tan ( numeric_expression )
```

atn2

Returns the angle, in radians, whose tangent is "numeric_expression1" / "numeric_expression2".

Syntax

```
atn2 ( numeric_expression1, numeric_expression2 )
```

cos

Returns the cosine of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

Syntax

```
cos ( numeric_expression )
```

cot

Returns the cotangent of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

Syntax

```
cot ( numeric_expression )
```

degrees

Returns "numeric_expression" radians converted to degrees.

Syntax

```
degrees ( numeric_expression )
```

radians

Returns the degree equivalent of "numeric_expression". Results are of the same type as "numeric_expression". For numeric or decimal expressions, the results have an internal precision of 77 and

a scale equal to that of "numeric_expression". When the money datatype is used, an internal conversion to float may cause some loss of precision.

Syntax

```
radians ( numeric_expression )
```

sin

Returns the sine of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

Syntax

```
sin ( numeric_expression )
```

Postgres

Postgres String

overlay

Returns the "string_expression1" replacing "string_expression2" from character position numeric_expression.

Syntax

```
overlay ( string_expression1 , string_expression2 , numeric_expression1 [ ,  
numeric_expression2 ] )
```

btrim

Returns string_expression1 after removing the longest string of characters in "string_expression2".

Syntax

```
btrim ( string_expression1 [ , string_expression2 ] )
```

initcap

Returns "string_expression", with the first letter of each word in uppercase and all other letters in lowercase. Words are delimited by white space or characters that are not alphanumeric.

Syntax

```
initcap ( string_expression )
```

lpad

Returns "string_expression1" padded to length "integer_expression" with occurrences of "string_expression2". If "string_expression1" is longer than "integer_expression", the appropriate portion of "string_expression1" is returned.

Syntax

```
lpad ( string_expression1 , integer_expression [ , string_expression2 ] )
```

ltrim

Returns "string_expression1", with leading characters removed up to the first character not in "string_expression2"; for example, `ltrim ('xyxXxyAB' , 'xy')` returns XxyAB.

Syntax

```
ltrim ( string_expression1 [ , string_expression2 ] )
```

md5

Returns the MD5 hash of "string_expression1".

Syntax

```
md5 ( string_expression1 )
```

to_hex

Returns the hexadecimal string representation of "numeric_expression1".

Syntax

```
to_hex ( numeric_expression1 )
```

repeat

Returns the "string_expression" repeated "numeric_expression1" times.

Syntax

```
repeat ( string_expression , numeric_expression1 )
```

replace

Returns "string_expression" with "string_expression2" replaced with "string_expression3".

Syntax

```
replace ( string_expression , string_expression2 , string_expression3)
```

rpad

Returns "string_expression1" right-padded to length "integer_expression" with occurrences of "string_expression2". If "string_expression1" is longer than "integer_expression", the appropriate portion of "string_expression1" is returned. If "string_expression2" is not specified, then spaces are used.

Syntax

```
rpad ( string_expression1 , integer_expression [ , string_expression2 ] )
```

rtrim

Returns "string_expression1", with final characters removed after the last character not in "string_expression2"; for example, `rtrim ('ABxXxyx' , 'xy')` returns ABxX. If "string_expression2" is not specified, the final space characters are removed.

Syntax

```
rtrim ( string_expression1 [ , string_expression2 ] )
```

split_part

Returns "numeric_expression" field having split "string_expression1" on "string_expression2".

Syntax

```
split_part ( string_expression1 , string_expression2 , numeric_expression )
```

ascii

Returns a number representing the ASCII code value of the leftmost character of "string_expression"; for example, ascii ('A') is 65.

Syntax

```
ascii ( string_expression )
```

chr

Returns the character that has the ASCII code value specified by "integer_expression". "Integer_expression" should be between 0 and 255.

Syntax

```
chr ( integer_expression )
```

{current_catalog}**Syntax**

```
{current_catalog}
```

{current_db}**Syntax**

```
{current_db}
```

{current_schema}**Syntax**

```
{current_schema}
```

{current_user}**Syntax**

```
{current_user}
```

{session_user}**Syntax**

```
{session_user}
```

Postgres Data type formatting

to_char

Returns the string representation of "expression" with the format of "string_expression".
"Expression" can be either a date value or a numeric value.

Syntax

```
to_char ( expression , string_expression )
```

to_date

Converts "string_expression1" to a date value as specified by the format "string_expression2".

Syntax

```
to_date ( string_expression1 , string_expression2 )
```

to_number

Converts "string_expression1" to a numeric value as specified by the format "string_expression2".

Syntax

```
to_number ( string_expression1 , string_expression2 )
```

to_timestamp

Converts "string_expression1" to a timestamp value as specified by the format "string_expression2".

Syntax

```
to_timestamp ( string_expression1 , string_expression2 )
```

translate

Returns "string_expression1", with each occurrence of each character in "string_expression2" replaced by its corresponding character in "string_expression3".

Syntax

```
translate ( string_expression1 , string_expression2 , string_expression3 )
```

date_trunc

Returns the timestamp to the specified precision.

Syntax

```
date_trunc ( string_expression , timestamp_expression )
```

version

Returns the string value of the database version.

Syntax

```
version ( )
```

Postgres Math

log

Returns the base 10 logarithm of "numeric_expression1" or logarithm to the base "numeric_expression2".

Syntax

```
log ( numeric_expression1 [ , numeric_expression2 ] )
```

ln

Returns the natural logarithm of "numeric_expression1".

Syntax

```
ln ( numeric_expression )
```

cbrt

Returns the cube root of "numeric_expression1".

Syntax

```
cbrt ( numeric_expression )
```

div

Returns the integer quotient of "numeric_expression1" divided by "numeric_expression2".

Syntax

```
div ( numeric_expression1 , numeric_expression2 )
```

pi

Returns the constant of pi.

Syntax

```
pi ( )
```

Postgres Trigonometry

acos

Returns the arccosine of "numeric_expression" in radians. The arccosine is the angle whose cosine is "numeric_expression".

Syntax

```
acos ( numeric_expression )
```

asin

Returns the arcsine of "numeric_expression" in radians. The arcsine is the angle whose sine is "numeric_expression".

Syntax

```
asin ( numeric_expression )
```

atan

Returns the arctangent of "numeric_expression" in radians. The arctangent is the angle whose tangent is "numeric_expression".

Syntax

```
atan ( numeric_expression )
```

atan2

Returns the arctangent of the x and y coordinates specified by "numeric_expression1" and "numeric_expression2", respectively, in radians. The arctangent is the angle whose tangent is "numeric_expression2" / "numeric_expression1".

Syntax

```
atan2 ( numeric_expression1 , numeric_expression2 )
```

cos

Returns the cosine of "numeric_expression", where "numeric_expression" is an angle expressed in radians.

Syntax

```
cos ( numeric_expression )
```

cot

Returns the cotangent of "numeric_expression", where "numeric_expression" is an angle expressed in radians.

Syntax

```
cot ( numeric_expression )
```

degrees

Returns the degrees where "numeric_expression" is an angle expressed in radians.

Syntax

```
degrees ( numeric_expression )
```

radians

Returns the radians where "numeric_expression" is an angle expressed in degrees.

Syntax

```
radians ( numeric_expression )
```

sin

Returns the sine of "numeric_expression", where "numeric_expression" is an angle expressed in radians.

Syntax

```
sin ( numeric_expression )
```

tan

Returns the tangent of "numeric_expression", where "numeric_expression" is an angle expressed in radians.

Syntax

```
tan ( numeric_expression )
```

Vertica

Vertica String

overlay

Returns the "string_expression1", replacing "string_expression2" from character position numeric_expression.

Syntax

```
overlay ( string_expression1 , string_expression2 , numeric_expression1 [ ,  
numeric_expression2 ] )
```

btrim

Returns string_expression1 after removing the longest string of characters in string_expression2.

Syntax

```
btrim ( string_expression1 [ , string_expression2 ] )
```

initcap

Returns "string_expression", with the first letter of each word in uppercase and all other letters in lowercase. Words are delimited by white space or characters that are not alphanumeric.

Syntax

```
initcap ( string_expression )
```

lpad

Returns "string_expression1" padded to length "integer_expression" with occurrences of "string_expression2". If "string_expression1" is longer than "integer_expression", the appropriate portion of "string_expression1" is returned.

Syntax

```
lpad ( string_expression1 , integer_expression [ , string_expression2 ] )
```

ltrim

Returns "string_expression1", with leading characters removed up to the first character not in "string_expression2"; for example, ltrim ('xyxXxyAB' , 'xy') returns XxyAB.

Syntax

```
ltrim ( string_expression1 [ , string_expression2 ] )
```

to_hex

Returns the hexadecimal string representation of "numeric_exp1".

Syntax

```
to_hex ( numeric_expression1 )
```

repeat

Returns the "string_expression" repeated "numeric_expression1" times.

Syntax

```
repeat ( string_expression , numeric_expression1 )
```

replace

Returns "string_expression" having replaced "string_expression2" with "string_expression3".

Syntax

```
replace ( string_expression , string_expression2 , string_expression3 )
```

rpad

Returns "string_expression1" right-padded to length "integer_expression" with occurrences of "string_expression2". If "string_expression1" is longer than "integer_expression", the appropriate portion of "string_expression1" is returned. If "string_expression2" is not specified, then spaces are used.

Syntax

```
rpad ( string_expression1 , integer_expression [ , string_expression2 ] )
```

rtrim

Returns "string_expression1", with final characters removed after the last character not in "string_expression2"; for example, rtrim ('ABxXxyx' , 'xy') returns ABxX. If "string_expression2" is not specified, the final space characters are removed.

Syntax

```
rtrim ( string_expression1 [ , string_expression2 ] )
```

ascii

Returns a number representing the ASCII code value of the leftmost character of "string_expression"; for example, ascii ('A') is 65.

Syntax

```
ascii ( string_expression )
```

chr

Returns the character that has the ASCII code value specified by "integer_expression". "Integer_expression" should be between 0 and 255.

Syntax

```
chr ( integer_expression )
```

current_database

Returns the name of the current database.

Syntax

```
current_database ()
```

current_schema

Returns the name of the current schema

Syntax

```
current_schema ()
```

{current_user}**Syntax**

```
{current_user}
```

{session_user}**Syntax**

```
{session_user}
```

Vertica Data type formatting**to_char**

Returns the string representation of "expression" with the format of "string_expression". "Expression" can be either a date value or a numeric value.

Syntax

```
to_char ( expression , string_expression )
```

to_date

Converts "string_expression1" to a date value as specified by the format "string_expression2".

Syntax

```
to_date ( string_expression1 , string_expression2 )
```

to_number

Converts "string_expression1" to a numeric value as specified by the format "string_expression2".

Syntax

```
to_number ( string_expression1, string_expression2 )
```

to_timestamp

Converts "string_expression1" to a timestamp value as specified by the format "string_expression2".

Syntax

```
to_timestamp ( string_expression1, string_expression2 )
```

translate

Returns "string_expression1", with each occurrence of each character in "string_expression2" replaced by its corresponding character in "string_expression3".

Syntax

```
translate ( string_expression1 , string_expression2 , string_expression3 )
```

date_trunc

Returns the timestamp to the specified precision.

Syntax

```
date_trunc ( string_expression , timestamp_expression)
```

version

Returns the string value of the database version.

Syntax

```
version ()
```

Vertica Math

log

Returns the base 10 logarithm of "numeric_expression1" or logarithm to the base "numeric_expression2".

Syntax

```
log ( numeric_expression1 [ , numeric_expression2 ] )
```

ln

Returns the natural logarithm of "numeric_expression1".

Syntax

```
ln ( numeric_expression )
```


cbrt

Returns the cube root of "numeric_expression1".

Syntax

```
cbrt ( numeric_expression )
```

pi

Returns the constant of pi.

Syntax

```
pi ( )
```

Vertica Trigonometry**acos**

Returns the arccosine of "numeric_expression" in radians. The arccosine is the angle whose cosine is "numeric_expression".

Syntax

```
acos ( numeric_expression )
```

asin

Returns the arcsine of "numeric_expression" in radians. The arcsine is the angle whose sine is "numeric_expression".

Syntax

```
asin ( numeric_expression )
```

atan

Returns the arctangent of "numeric_expression" in radians. The arctangent is the angle whose tangent is "numeric_expression".

Syntax

```
atan ( numeric_expression )
```

atan2

Returns the arctangent of the x and y coordinates specified by "numeric_expression1" and "numeric_expression2", respectively, in radians. The arctangent is the angle whose tangent is "numeric_expression2" / "numeric_expression1".

Syntax

```
atan2 ( numeric_expression1 , numeric_expression2 )
```

cos

Returns the cosine of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

Syntax

```
cos ( numeric_expression )
```

cot

Returns the cotangent of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

Syntax

```
cot ( numeric_expression )
```

degrees

Returns the degrees where "numeric_expression" is an angle expressed in radians.

Syntax

```
degrees ( numeric_expression )
```

radians

Returns the radians where "numeric_expression" is an angle expressed in degrees.

Syntax

```
radians ( numeric_expression )
```

sin

Returns the sine of "numeric_exp" where "numeric_expression" is an angle expressed in radians.

Syntax

```
sin ( numeric_expression )
```

tan

Returns the tangent of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

Syntax

```
tan ( numeric_expression )
```

Paracel

Paracel String

overlay

Returns the "string_expression1", replacing "string_expression2" from character position numeric_expression.

Syntax

```
overlay ( string_expression1 , string_expression2 , numeric_expression1 [ ,  
numeric_expression2 ] )
```

ltrim

Returns "string_expression1", with leading characters removed up to the first character not in "string_expression2"; for example, ltrim ('xyxXxyAB' , 'xy') returns XxyAB.

Syntax

```
ltrim ( string_expression1 [ , string_expression2 ] )
```

replace

Returns "string_expression", having replaced "string_expression2" with "string_expression3".

Syntax

```
replace ( string_expression , string_expression2 , string_expression3 )
```

rtrim

Returns "string_expression1", with final characters removed after the last character not in "string_expression2"; for example, rtrim ('ABxXxyx' , 'xy') returns ABxX. If "string_expression2" is not specified, the final space characters are removed.

Syntax

```
rtrim ( string_expression1 [ , string_expression2 ] )
```

current_database

Returns the name of the current database.

Syntax

```
current_database ()
```

current_schema

Returns the name of the current schema

Syntax

```
current_schema ()
```

{current_user}**Syntax**

```
{current_user}
```

{session_user}**Syntax**

```
{session_user}
```

Paracel Data type formatting

to_char

Returns the string representation of "expression" with the format of "string_expression".
"Expression" can be either a date value or a numeric value.

Syntax

```
to_char ( expression , string_expression )
```

to_date

Converts "string_expression1" to a date value as specified by the format "string_expression2".

Syntax

```
to_date ( string_expression1 , string_expression2 )
```

to_number

Converts "string_expression1" to a numeric value as specified by the format "string_expression2".

Syntax

```
to_number ( string_expression1 , string_expression2 )
```

translate

Returns "string_expression1", with each occurrence of each character in "string_expression2" replaced by its corresponding character in "string_expression3".

Syntax

```
translate ( string_expression1 , string_expression2 , string_expression3 )
```

version

Returns the string value of the database version.

Syntax

```
version ()
```

Paracel Math

cbirt

Returns the cube root of "numeric_expression1".

Syntax

```
cbirt ( numeric_expression )
```

pi

Returns the constant of pi.

Syntax

```
pi ()
```

MySQL

MySQL String

lpad

Returns "string_expression1" padded to length "integer_expression" with occurrences of "string_expression2". If "string_expression1" is longer than "integer_expression", the appropriate portion of "string_expression1" is returned.

Syntax

```
lpad ( string_expression1 , integer_expression [ , string_expression2 ] )
```

ltrim

Returns "string_expression1", with leading characters removed up to the first character not in "string_expression2"; for example, ltrim ('xyxXxyAB' , 'xy') returns XxyAB.

Syntax

```
ltrim ( string_expression1 [ , string_expression2 ] )
```

hex

Returns the hexadecimal string representation of "numeric_expression1".

Syntax

```
hex ( numeric_expression1 )
```

repeat

Returns the "string_expression" repeated "numeric_expression1" times.

Syntax

```
repeat ( string_expression , numeric_expression1 )
```

replace

Returns "string_expression" having replaced "string_expression2" with "string_expression3".

Syntax

```
replace ( string_expression , string_expression2 , string_expression3 )
```

reverse

Returns "string_expression" reversed.

Syntax

```
reverse ( string_expression )
```

right

Returns the rightmost "numeric_expression" characters from "string_expression1".

Syntax

```
right ( string_expression1 , numeric_expression )
```

rpad

Returns "string_expression1" right-padded to length "integer_expression" with occurrences of "string_expression2". If "string_expression1" is longer than "integer_expression", the appropriate portion of "string_expression1" is returned. If "string_expression2" is not specified, then spaces are used.

Syntax

```
rpad ( string_expression1 , integer_expression [ , string_expression2 ] )
```

rtrim

Returns "string_expression1", with final characters removed after the last character not in "string_expression2"; for example, rtrim ('ABxXxyx' , 'xy') returns ABxX. If "string_expression2" is not specified, the final space characters are removed.

Syntax

```
rtrim ( string_expression1 [ , string_expression2 ] )
```

soundex

Returns a soundex string of "string_expression1".

Syntax

```
soundex ( string_expression1 )
```

ascii

Returns a number representing the ASCII code value of the leftmost character of "string_expression"; for example, ascii ('A') is 65.

Syntax

```
ascii ( string_expression )
```

database

Returns the current database name

Syntax

```
database ( )
```

schema

Returns the current schema name

Syntax

```
schema ( )
```

session_user

Return the user name returned by the client

Syntax

```
session_user ()
```

system_user

Return the user name returned by the client

Syntax

```
system_user ()
```

version

Returns the string value of the database version.

Syntax

```
version ()
```

MySQL Math**log**

Returns the base 10 logarithm of "numeric_expression1" or logarithm to the base "numeric_expression2".

Syntax

```
log ( numeric_expression )
```

ln

Returns the natural logarithm of "numeric_expression1".

Syntax

```
ln ( numeric_expression )
```

pi

Returns the constant of pi.

Syntax

```
pi ()
```

MySQL Trigonometry**acos**

Returns the arccosine of "numeric_expression" in radians. The arccosine is the angle whose cosine is "numeric_expression".

Syntax

```
acos ( numeric_expression )
```

asin

Returns the arcsine of "numeric_expression" in radians. The arcsine is the angle whose sine is "numeric_expression".

Syntax

```
asin ( numeric_expression )
```

atan

Returns the arctangent of "numeric_expression" in radians. The arctangent is the angle whose tangent is "numeric_expression".

Syntax

```
atan ( numeric_expression )
```

atan2

Returns the arctangent of the x and y coordinates specified by "numeric_expression1" and "numeric_expression2", respectively, in radians. The arctangent is the angle whose tangent is "numeric_expression2" / "numeric_expression1".

Syntax

```
atan2 ( numeric_expression1 ,numeric_expression2 )
```

cos

Returns the cosine of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

Syntax

```
cos ( numeric_expression )
```

cot

Returns the cotangent of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

Syntax

```
cot ( numeric_expression )
```

degrees

Returns the degrees where "numeric_expression" is an angle expressed in radians.

Syntax

```
degrees ( numeric_expression )
```

radians

Returns the radians where "numeric_expression" is an angle expressed in degrees.

Syntax

```
radians ( numeric_expression )
```

sin

Returns the sine of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

Syntax

```
sin ( numeric_expression )
```

tan

Returns the tangent of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

Syntax

```
tan ( numeric_expression )
```

Greenplum

Greenplum String

overlay

Returns the "string_expression1" replacing "string_expression2" from character position "numeric_expression".

Syntax

```
overlay ( string_expression1 , string_expression2 , numeric_expression1 [ ,  
numeric_expression2 ] )
```

btrim

Returns "string_expression1" after removing the longest string of characters in "string_expression2".

Syntax

```
btrim ( string_expression1 [ , string_expression2 ] )
```

initcap

Returns "string_expression" with the first letter of each word in uppercase and all other letters in lowercase. Words are delimited by white space or characters that are not alphanumeric.

Syntax

```
initcap ( string_expression )
```

lpad

Returns "string_expression1" padded to length "integer_expression" with occurrences of "string_expression2". If "string_expression1" is longer than "integer_expression", the appropriate portion of "string_expression1" is returned.

Syntax

```
lpad ( string_expression1 , integer_expression [ , string_expression2 ] )
```

ltrim

Returns "string_expression1", with leading characters removed up to the first character not in "string_expression2"; for example, ltrim ('xyxXxyAB' , 'xy') returns XxyAB.

Syntax

```
ltrim ( string_expression1 [ , string_expression2 ] )
```

md5

Returns the MD5 hash of "string_expression1".

Syntax

```
md5 ( string_expression1 )
```

to_hex

Returns the hexadecimal string representation of "numeric_expression1".

Syntax

```
to_hex ( numeric_expression1 )
```

repeat

Returns the "string_expression" repeated "numeric_expression1" times.

Syntax

```
repeat ( string_expression , numeric_expression1 )
```

replace

Returns "string_expression" having replaced "string_expression2" with "string_expression3".

Syntax

```
replace ( string_expression , string_expression2 , string_expression3)
```

rpadd

Returns "string_expression1" right-padded to length "integer_expression" with occurrences of "string_expression2". If "string_expression1" is longer than "integer_expression", the appropriate portion of "string_expression1" is returned. If "string_expression2" is not specified, then spaces are used.

Syntax

```
rpadd ( string_expression1 , integer_expression [ , string_expression2 ] )
```

rtrim

Returns "string_expression1", with final characters removed after the last character not in "string_expression2"; for example, `rtrim ('ABxXxyx' , 'xy')` returns ABxX. If "string_expression2" is not specified, the final space characters are removed.

Syntax

```
rtrim ( string_expression1 [ , string_expression2 ] )
```

split_part

Returns "numeric_expression" field having split "string_expression1" on "string_expression2".

Syntax

```
split_part ( string_expression1 , string_expression2 , numeric_expression )
```

ascii

Returns a number representing the ascii code value of the leftmost character of "string_expression"; for example, `ascii ('A')` is 65.

Syntax

```
ascii ( string_expression )
```

chr

Returns the character that has the ASCII code value specified by "integer_expression". "Integer_expression" should be between 0 and 255.

Syntax

```
chr ( integer_expression )
```

current_database

Returns the name of the current database.

Syntax

```
current_database ()
```

current_schema

Returns the name of the current schema.

Syntax

```
current_schema ()
```

{current_user}**Syntax**

```
{current_user}
```

{session_user}

Syntax

```
{session_user}
```

Greenplum Data type formatting

to_char

Returns the string representation of "expression" with the format of "string_expression".
"Expression" can either be a date value or a numeric value.

Syntax

```
to_char ( expression , string_expression )
```

to_date

Converts "string_expression1" to a date value as specified by the format "string_expression2".

Syntax

```
to_date ( string_expression1 , string_expression2 )
```

to_number

Converts "string_expression1" to a numeric value as specified by the format "string_expression2".

Syntax

```
to_number ( string_expression1 , string_expression2 )
```

to_timestamp

Converts "string_expression1" to a timestamp value as specified by the format "string_expression2".

Syntax

```
to_timestamp ( string_expression1 , string_expression2 )
```

translate

Returns "string_expression1" with each occurrence of each character in "string_expression2" replaced by its corresponding character in "string_expression3".

Syntax

```
translate ( string_expression1 , string_expression2 , string_expression3 )
```

date_trunc

Returns the timestamp to the specified precision.

Syntax

```
date_trunc ( string_expression , timestamp_expression)
```

version

Returns the string value of the database version.

Syntax

```
version ()
```

Greenplum Math**log**

Returns the base 10 logarithm of "numeric_expression1" or logarithm to the base "numeric_expression2".

Syntax

```
log ( numeric_expression1 [ , numeric_expression2 ] )
```

ln

Returns the natural logarithm of "numeric_expression1".

Syntax

```
ln ( numeric_expression )
```

cbrt

Returns the cube root of "numeric_expression1".

Syntax

```
cbrt ( numeric_expression )
```

pi

Returns the constant of pi.

Syntax

```
pi ()
```

Greenplum Trigonometry**acos**

Returns the arccosine of "numeric_expression" in radians. The arccosine is the angle whose cosine is "numeric_expression".

Syntax

```
acos ( numeric_expression )
```

asin

Returns the arcsine of "numeric_expression" in radians. The arcsine is the angle whose sine is "numeric_expression".

Syntax

```
asin ( numeric_expression )
```

atan

Returns the arctangent of "numeric_expression" in radians. The arctangent is the angle whose tangent is "numeric_expression".

Syntax

```
atan ( numeric_expression )
```

atan2

Returns the arctangent of the x and y coordinates specified by "numeric_expression1" and "numeric_expression2", respectively, in radians. The arctangent is the angle whose tangent is "numeric_expression2" / "numeric_expression1".

Syntax

```
atan2 ( numeric_expression1 ,numeric_expression2 )
```

cos

Returns the cosine of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

Syntax

```
cos ( numeric_expression )
```

cot

Returns the cotangent of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

Syntax

```
cot ( numeric_expression )
```

degrees

Returns the degrees where "numeric_expression" is an angle expressed in radians.

Syntax

```
degrees ( numeric_expression )
```

radians

Returns the radians where "numeric_expression" is an angle expressed in degrees.

Syntax

```
radians ( numeric_expression )
```

sin

Returns the sine of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

Syntax

```
sin ( numeric_expression )
```

tan

Returns the tangent of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

Syntax

```
tan ( numeric_expression )
```

Report Functions

_add_days

Returns the datetime resulting from adding "integer_expression" days to "timestamp_expression".

Syntax

```
_add_days ( timestamp_expression , integer_expression )
```

Example

```
_add_days ( 2007-01-14 00:00:00.000 , 3 )
```

Result: 2007-01-17 00:00:00.000

_add_months

Returns the datetime resulting from adding "integer_expression" months to "timestamp_expression".

Syntax

```
_add_months ( timestamp_expression , integer_expression )
```

_add_years

Returns the datetime resulting from adding "integer_expression" years to "timestamp_expression".

Syntax

```
_add_years ( timestamp_expression , integer_expression )
```

_age

Returns a number by subtracting "timestamp_expression" from today's date.

Syntax

```
_age ( timestamp_expression )
```

Example

```
_age ([Query1].[Date]), where [Query1].[Date] is March 2, 2004, and today is  
July 8, 2009
```

Result: 50,406, where 5 is the number of years, 04 is the number of months, and 06 is the number of days.

_day_of_week

Returns the day of the week (between 1 and 7) for "timestamp_expression" where "integer_expression" indicates which day of that week is day 1. To determine "integer_expression", choose the day of the week and count from Monday; for example, if you choose Wednesday, "integer_expression" would be 3 because Wednesday is the third day from Monday.

Syntax

```
_day_of_week ( timestamp_expression , integer_expression )
```

Example

```
_day_of_week ( 2009-01-01 , 7 ), where 7 means that Sunday is the first day of the week.
```

Result: 5

_day_of_year

Returns the ordinal for the day of the year in "timestamp_expression" (1 to 366). Also known as Julian day.

Syntax

```
_day_of_year ( timestamp_expression )
```

_days_between

Returns a positive or negative number representing the number of days between "timestamp_expression1" and "timestamp_expression2". If "timestamp_expression1" < "timestamp_expression2", the result will be a negative number.

Syntax

```
_days_between ( timestamp_expression1 , timestamp_expression2 )
```

_days_to_end_of_month

Returns a number representing the number of days remaining in the month represented by "timestamp_expression".

Syntax

```
_days_to_end_of_month ( timestamp_expression )
```

_first_of_month

Returns a datetime that is the first day of the month represented by "timestamp_expression".

Syntax

```
_first_of_month ( timestamp_expression )
```

Example 1

```
_first_of_month ( 2009-05-04 00:00:00.000 )
```

Result: Returns 2009-05-01 00:00:00.000

Example 2

```
_first_of_month (current_date)
```

Result: Returns Jul 1, 2009 if the current date is July 30, 2009.

`_last_of_month`

Returns a datetime that is the last day of the month represented by "timestamp_expression".

Syntax

```
_last_of_month ( timestamp_expression )
```

`_make_timestamp`

Returns a timestamp constructed from "integer_expression1" (the year), "integer_expression2" (the month), and "integer_expression3" (the day). The time portion defaults to 00:00:00.000 .

Syntax

```
_make_timestamp ( integer_expression1 , integer_expression2 , integer_expression3 )
```

`_months_between`

Returns a positive or negative number representing the number of months between "timestamp_expression1" and "timestamp_expression2". If "timestamp_expression1" < "timestamp_expression2", the result will be a negative number.

Syntax

```
_months_between ( timestamp_expression1 , timestamp_expression2 )
```

`_week_of_year`

Returns the week number (1-53) of the year for "timestamp_expression". According to the ISO 8601, week 1 of the year is the first week to contain a Thursday, which is equivalent to the first week containing January 4th. A week starts on a Monday (day 1) and ends on a Sunday (day 7).

Syntax

```
_week_of_year ( timestamp_expression )
```

`_years_between`

Returns a positive or negative integer representing the number of years between "timestamp_expression1" and "timestamp_expression2". If "timestamp_expression1" < "timestamp_expression2", a negative value is returned.

Syntax

```
_years_between ( timestamp_expression1 , timestamp_expression2 )
```

`_ymdint_between`

Returns a number representing the difference between "timestamp_expression1" and "timestamp_expression2". This value has the form YYMMDD, where YY represents the number of years, MM represents the number of months, and DD represents the number of days.

Syntax

```
_ymdint_between ( timestamp_expression1 , timestamp_expression2 )
```

Example

```
_ymdint_between ( [Query1].[Date (close date)] , [Query1].[Date (ship date)] ),  
where [Query1].[Date (close date)] is February 20, 2004, and [Query1].[Date  
(ship date)] is January 19, 2004.
```

Result: 101, where 1 is the number of months and 01 is the number of days.

abs

Returns the absolute value of "numeric_expression". If "numeric_expression" is negative, a positive value is returned.

Syntax

```
abs ( numeric_expression )
```

AsOfDate

Returns the date value of the AsOfDate expression, if it is defined. Otherwise, AsOfDate returns the report execution date.

Syntax

```
AsOfDate ( )
```

AsOfTime

Returns the time value of the AsOfTime expression, if it is defined. Otherwise, AsOfTime returns the report execution time.

Syntax

```
AsOfTime ( )
```

BurstKey

Returns the burst key.

Syntax

```
BurstKey ( )
```

BurstRecipients

Returns the distribution list of burst recipients.

Syntax

```
BurstRecipients ( )
```

ceiling

Returns the smallest integer that is greater than or equal to "numeric_expression".

Syntax

```
ceiling ( numeric_expression )
```

CellValue

Returns the value of the current crosstab cell.

Syntax

```
CellValue ()
```

character_length

Returns the number of characters in "string_expression".

Syntax

```
character_length ( string_expression )
```

ColumnNumber

Returns the current column number.

Syntax

```
ColumnNumber ()
```

CubeCreatedOn

Returns the date and time when the cube was created. "Dimension" specifies from which cube to retrieve the metadata. If the dimension source is an IBM® Cognos® PowerCube (.mdc), the function returns a blank string because the initial creation date of a PowerCube is not maintained.

Syntax

```
CubeCreatedOn ( dimension )
```

CubeCurrentPeriod

Returns the current period for the cube. "Dimension" specifies from which cube to retrieve the metadata.

Syntax

```
CubeCurrentPeriod ( dimension )
```

CubeDataUpdatedOn

Returns the date time that data in the cube was last updated. "Dimension" specifies from which cube to retrieve the metadata.

Syntax

```
CubeDataUpdatedOn ( dimension )
```

CubeDefaultMeasure

Returns the name of the default measure for the cube. "Dimension" specifies from which cube to retrieve the metadata.

Syntax

```
CubeDefaultMeasure ( dimension )
```

CubeDescription

Returns the description of the cube. "Dimension" specifies from which cube to retrieve the metadata.

Syntax

```
CubeDescription ( dimension )
```

CubeIsOptimized

Returns "true" if the cube is optimized. "Dimension" specifies from which cube to retrieve the metadata.

Syntax

```
CubeIsOptimized ( dimension )
```

CubeName

Returns the name of the cube. "Dimension" specifies from which cube to retrieve the metadata.

Syntax

```
CubeName ( dimension )
```

CubeSchemaUpdatedOn

Returns the date time that the cube schema was last updated. "Dimension" specifies from which cube to retrieve the metadata.

Syntax

```
CubeSchemaUpdatedOn ( dimension )
```

exp

Returns the constant 'e' raised to the power of "numeric_expression". The constant 'e' is the base of the natural logarithm.

Syntax

```
exp ( numeric_expression )
```

Example

```
exp ( 2 )
```

Result: 7.389056

extract

Returns an integer representing the value of "date_part_expression" in "datetime_expression". "Date_part_expression" could be the year, month, day, hour, minute, or second.

Syntax

```
extract ( date_part_expression , datetime_expression )
```

Example 1

```
extract ( year , 2003-03-03 16:40:15.535 )
```

Result: 2003

Example 2

```
extract ( hour , 2003-03-03 16:40:15.535 )
```

Result: 16

floor

Returns the largest integer that is less than or equal to "numeric_expression".

Syntax

```
floor ( numeric_expression )
```

GetLocale

Returns the run locale (deprecated).

Syntax

```
GetLocale ()
```

HorizontalPageCount

Returns the current horizontal page count.

Syntax

```
HorizontalPageCount ()
```

HorizontalPageNumber

Returns the current horizontal page number.

Syntax

```
HorizontalPageNumber ()
```

InScope

Returns Boolean 1 (true) when the cell is in the scope of the data items and MUNs; otherwise, returns Boolean 0 (false).

Syntax

```
InScope ( dataItem , MUN, ... )
```

IsAccessible

Returns Boolean 1 (true) if the report is run with the accessibility features enabled. Use this function as a variable expression with a conditional block to make your reports accessible. For example, you can add a list or crosstab equivalent to a chart in reports that are run with accessibility features enabled.

Syntax

```
IsAccessible()
```

IsBursting

Returns Boolean 1 (true) when the report will be distributed to the recipient; otherwise, returns Boolean 0 (false).

Syntax

```
IsBursting ('recipientName')
```

IsCrosstabColumnNodeMember

Returns Boolean 1 (true) if the current node is a crosstab column node member.

Syntax

```
IsCrosstabColumnNodeMember ()
```

IsCrosstabRowNodeMember

Returns Boolean 1 (true) if the current node is a crosstab row node member.

Syntax

```
IsCrosstabRowNodeMember ()
```

IsFirstColumn

Returns Boolean 1 (true) if the current column is the first column.

Syntax

```
IsFirstColumn ()
```

IsInnerMostCrosstabColumnNodeMember

Returns Boolean 1 (true) if the current node is an innermost crosstab column node member.

Syntax

```
IsInnerMostCrosstabColumnNodeMember ()
```

IsInnerMostCrosstabRowNodeMember

Returns Boolean 1 (true) if the current node is an innermost crosstab row node member.

Syntax

```
IsInnerMostCrosstabRowNodeMember ()
```

IsLastColumn

Returns Boolean 1 (true) if the current column is the last column.

Syntax

```
IsLastColumn ()
```

IsLastInnerMostCrosstabColumnNodeMember

Returns Boolean 1 (true) if the current node is the last innermost crosstab column node member.

Syntax

```
IsLastInnerMostCrosstabColumnNodeMember ()
```

IsLastInnerMostCrosstabRowNodeMember

Returns Boolean 1 (true) if the current node is the last innermost crosstab row node member.

Syntax

```
IsLastInnerMostCrosstabRowNodeMember ()
```

IsOuterMostCrosstabColumnNodeMember

Returns Boolean 1 (true) if the current node is an outermost crosstab column node member.

Syntax

```
IsOuterMostCrosstabColumnNodeMember ()
```

IsOuterMostCrosstabRowNodeMember

Returns Boolean 1 (true) if the current node is an outermost crosstab row node member.

Syntax

```
IsOuterMostCrosstabRowNodeMember ()
```

IsPageCountAvailable

Returns Boolean 1 (true) if the page count is available for the current execution of the report; otherwise, returns Boolean 0 (false).

Syntax

```
IsPageCountAvailable ()
```

ln

Returns the natural logarithm of "numeric_expression".

Syntax

```
ln ( numeric_expression )
```

Locale

Returns the run locale.

Syntax

```
Locale ()
```

lower

Returns "string_expression" with all uppercase characters converted to lowercase. This function appears in the Bursted Sales Performance Report sample report in the GO Data Warehouse (query) package.

Syntax

```
lower ( string_expression )
```

mapNumberToLetter

Adds "integer_expression" to "string_expression".

Syntax

```
mapNumberToLetter ( string_expression , integer_expression )
```

Example

```
mapNumberToLetter ( 'a' , 1 )
```

Result: b

mod

Returns an integer value representing the remainder (modulo) of "integer_expression1" / "integer_expression2".

Syntax

```
mod ( integer_expression1 , integer_expression2 )
```

ModelPath

Returns the model path.

Syntax

```
ModelPath ()
```

Now

Returns the current system time.

Syntax

```
Now ()
```

nullif

Returns null if "string_expression1" equals "string_expression2" (case-insensitive), otherwise returns "string_expression1".

Syntax

```
nullif ( string_expression1 , string_expression2 )
```

octet_length

Returns the number of bytes in "string_expression".

Syntax

```
octet_length ( string_expression )
```

PageCount

Returns the current page count. This function works only when the report output is Adobe® PDF or Microsoft® Excel. If you save the report output, this function works for all formats.

Syntax

```
PageCount ()
```

PageName

Returns the current page name.

Syntax

```
PageName ()
```

PageNumber

Returns the current page number.

Syntax

```
PageNumber ()
```

ParamCount

Returns the parameter count of "parameterName".

Syntax

```
ParamCount ('parameterName')
```

ParamDisplayValue

Returns a string that is the parameter display value of "parameterName". This function appears in the Recruitment Report sample report in the GO Data Warehouse (analysis) package.

Syntax

```
ParamDisplayValue ('parameterName')
```

ParamName

Returns the parameter name of "parameterName".

Syntax

```
ParamName ('parameterName')
```

ParamNames

Returns all parameter names.

Syntax

```
ParamNames ()
```

ParamValue

Returns the parameter value of "parameterName".

Syntax

```
ParamValue ('parameterName')
```

position

Returns the integer value representing the starting position of "string_expression1" in "string_expression2". Returns 0 if "string_expression1" is not found.

Syntax

```
position ( string_expression1 , string_expression2 )
```

power

Returns "numeric_expression1" raised to the power of "numeric_expression2".

Syntax

```
power ( numeric_expression1 , numeric_expression2 )
```

Example

```
power ( 3 , 2 )
```

Result: 9

ReportAuthorLocale

Returns the author locale.

Syntax

```
ReportAuthorLocale ()
```

ReportCreateDate

Returns the date when the report was created.

Syntax

```
ReportCreateDate ()
```

ReportDate

Returns the report execution date and time.

Syntax

```
ReportDate ()
```

ReportDescription

Returns the report description. This function works only when the report is run from IBM® Cognos® Connection.

Syntax

```
ReportDescription ()
```

ReportID

Returns the report ID.

Syntax

```
ReportID ()
```

ReportLocale

Returns the run locale.

Syntax

```
ReportLocale ()
```

ReportName

Returns the report name. This function works only when the report is run from IBM® Cognos® Connection.

Syntax

```
ReportName ()
```

ReportOption

Returns the value of the run option variable identified by "optionName", such as attachmentEncoding, burst, cssURL, email, emailAsAttachment, emailAsURL, emailBody, emailSubject, emailTo, emailToAddress, history, metadataModel, outputEncapsulation, outputFormat, outputLocale, outputPageDefinition, outputPageOrientation, primaryWaitThreshold, print, printer, printerAddress, prompt, promptFormat, saveAs, saveOutput, secondaryWaitThreshold, verticalElements, or xslURL.

Syntax

```
ReportOption ('optionName')
```

ReportOutput

Returns the name of the output format, such as CSV, HTML, layoutDataXML, MHT, PDF, rawXML, singleXLS, spreadsheetML, XLS, XML, or XLWA.

Syntax

```
ReportOutput ()
```

ReportPath

Returns the report path. This function works only when the report is run from IBM® Cognos® Connection.

Syntax

```
ReportPath ()
```

ReportProductLocale

Returns the product locale.

Syntax

```
ReportProductLocale ()
```

ReportSaveDate

Returns the date when the report was last saved.

Syntax

```
ReportSaveDate ()
```

round

Returns "numeric_expression" rounded to the nearest value with "integer_expression" significant digits to the right of the decimal point. If "integer_expression" is negative, "numeric_expression" is rounded to the nearest absolute value with "integer_expression" significant digits to the left of the decimal point. Rounding takes place before data formatting is applied.

Syntax

```
round ( numeric_expression , integer_expression )
```

Example

```
round (125, -1)
```

Result: 130

RowNumber

Returns the current row.

Syntax

```
RowNumber ()
```

ServerLocale

Returns the locale of the server that runs the report.

Syntax

```
ServerLocale ()
```

ServerName

Returns the name of the web server where the run request originated from. The value may be empty if the request is executed from the scheduler.

Syntax

```
ServerName ()
```

sqrt

Returns the square root of "numeric_expression". "Numeric_expression" must not be a negative value.

Syntax

```
sqrt ( numeric_expression )
```

substring

Returns the substring of "string_expression" that starts at position "integer_expression1" for "integer_expression2" characters or to the end of "string_expression" if "integer_expression2" is -1. The first character in "string_expression" is at position 1.

Syntax

```
substring ( string_expression , integer_expression1 , integer_expression2 )
```

Example

```
substring ( [Sales (analysis)].[Sales staff].[Sales staff].[Sales  
staff].[Position code], 3 , 5 )
```

Result: Returns characters 3 to 7 of the position codes.

TOCHHeadingCount

Returns the table of contents heading count for a specified heading level.

Syntax

```
TOCHHeadingCount ( headingLevel )
```

Today

Returns the current system date.

Syntax

```
Today ()
```

trim

Returns "string_expression" trimmed of any leading and trailing blanks or trimmed of the character specified by "match_character_expression". "Trim_what_expression" may be "leading", "trailing", or "both" (default). "Match_character_expression" can be an empty string to trim blanks or can specify a character to be trimmed.

Syntax

```
trim ( trim_what_expression , match_character_expression , string_expression )
```

upper

Returns "string_expression" with all lowercase characters converted to uppercase.

Syntax

```
upper ( string_expression )
```

URLEncode

Returns the URL encoded value of the input text.

Syntax

```
URLEncode ('text')
```

Data Type Casting Functions

date2string

Returns a date as a string in YYYY-MM-DD format.

Syntax

```
date2string ( date_expression )
```

date2timestamp

Converts "date_expression" to a timestamp. The time part of the timestamp will equal zero.

Syntax

```
date2timestamp ( date_expression )
```

date2timestampTZ

Converts "date_expression" to a timestamp with a time zone. The time and time zone parts of the timestamp will equal zero.

Syntax

```
date2timestampTZ ( date_expression )
```

DTinterval2string

Returns a date time interval as a string in DDDD HH:MM:SS.FFFFFFFF or -DDDD HH:MM:SS.FFF format.

Syntax

```
DTinterval2string ( date_time_interval_expression )
```

DTinterval2stringAsTime

Returns a date time interval as a string in HHHH:MM:SS.FFFFFFFF or HH:MM:SS.FFF format. Days are converted to hours.

Syntax

```
DTinterval2stringAsTime ( date_time_interval_expression )
```

int2DTinterval

Converts an integer to a date time interval. "String_expression" specifies what "integer_expression" represents: "ns" = nanoseconds, "s" = seconds (default), "m" = minutes, "h" = hours, "d" = days.

Syntax

```
int2DTinterval ( integer_expression , string_expression )
```

Example 1

```
int2DTinterval (1020,"h")
```

Result: 42 days 12 hours

Example 2

```
int2DTinterval (1020,"s")
```

Result: 17 minutes

int2YMinterval

Converts "integer_expression" to a year month interval. "String_expression" specifies what "integer_expression" represents: "y" = years, "m" = months (default).

Syntax

```
int2YMinterval ( integer_expression , string_expression )
```

number2string

Converts "numeric_expression" to a string, using the %g format specifier (C/C++ syntax).

Syntax

```
number2string ( numeric_expression )
```

string2date

Returns "string_expression" as a date in YYYY-MM-DD format.

Syntax

```
string2date ( string_expression )
```

string2double

Returns a floating point number. "String_expression" has the following form: "[whitespace] [sign] [digits] [digits] [{d|D|e|E } [sign] digits]"

Syntax

```
string2double ( string_expression )
```

string2DTinterval

Returns "string_expression" as a date time interval in [-]DD HH:MM[:SS[.FFF]] format.

Syntax

```
string2DTinterval ( string_expression )
```

string2int32

Returns an integer. "String_expression" has the following form: "[whitespace] [{+|-}] [digits]"

Syntax

```
string2int32 ( string_expression )
```

string2int64

Returns a long integer. "String_expression" has the following form: "[whitespace] [{+|-}] [digits]"

Syntax

```
string2int64 ( string_expression )
```

string2time

Returns "string_expression" as a time in HH:MM:SS.FFFFFFFF format.

Syntax

```
string2time ( string_expression )
```

string2timestamp

Returns "string_expression" as a timestamp in YYYY-MM-DD [T|t][white space|+] HH:MM:SS.FFFFFFFF format.

Syntax

```
string2timestamp ( string_expression )
```

string2timestampTZ

Returns "string_expression" in YYYY-MM-DD HH:MM:SS.FFFFFFFF +HHMM or YYYY-MM-DD [T|t] HH:MM:SS.FFF -HHMM format.

Syntax

```
string2timestampTZ ( string_expression )
```

string2YMininterval

Returns "string_expression" as a Year Month Interval in [-]YY MM format.

Syntax

```
string2YMininterval ( string_expression )
```

time2string

Returns a time as a string in HH:MM:SS.FFF format.

Syntax

```
time2string ( time_expression )
```

timestamp2date

Converts "timestamp_expression" to a date. The time part of the timestamp will be ignored.

Syntax

```
timestamp2date ( timestamp_expression )
```

timestamp2string

Returns a timestamp as a string in YYYY-MM-DD HH:MM:SS.FFFFFFFF format.

Syntax

```
timestamp2string ( timestamp_expression )
```


timestamp2timestampTZ

Converts "timestamp_expression" to a timestamp with a time zone. The displacement part of the timestamp with the time zone will be zero.

Syntax

```
timestamp2timestampTZ ( timestamp_expression )
```

timestampTZ2date

Converts "timestamp_time_zone_expression" to a date. The time and time zone parts of the timestamp will be ignored.

Syntax

```
timestampTZ2date ( timestamp_time_zone_expression )
```

timestampTZ2string

Returns a timestamp with the time zone as a string in YYYY-MM-DD HH:MM:SS.FFFFFFFF +HHMM or YYYY-MM-DD HH:MM:SS.FFF -HHMM format.

Syntax

```
timestampTZ2string ( timestamp_time_zone_expression )
```

timestampTZ2timestamp

Converts "timestamp_time_zone_expression" to a timestamp. The displacement part of the timestamp with the time zone will be ignored.

Syntax

```
timestampTZ2timestamp ( timestamp_time_zone_expression )
```

timeTZ2string

Returns a time with the time zone as a string in HH:MM:SS.FFF +HHMM or HH:MM:SS.FFFFFFFF -HHMM format. For example, -05:30 means a timezone of GMT minus 5 hours and 30 minutes

Syntax

```
timeTZ2string ( timeTZ_expression )
```

YMininterval2string

Returns "year_month_interval_expression" as a string in (YY MM) or -(YY MM) format.

Syntax

```
YMininterval2string ( year_month_interval_expression )
```

Appendix C: Accessibility features

Accessibility features help users who have a physical disability, such as restricted mobility or limited vision, to use information technology products.

Accessibility features in Framework Manager

The major accessibility features are accelerators and command keys that you can use to navigate through Framework Manager.

- An underlined letter on the screen designates an accelerator; for example, F is the accelerator for the File menu. In Microsoft® Windows®, press the Alt key, then the accelerator to trigger an action; for example, Alt+F shows the File menu. If they are enabled, you can also use extended accelerators.
- Command keys directly trigger an action and usually make use of the Ctrl keys. For example, to print, press Ctrl+P.

Keyboard Shortcuts for Framework Manager

You can use keyboard shortcuts to navigate through and perform some tasks in Framework Manager. This product uses standard Microsoft® Windows® operating system navigation keys in addition to application-specific keys.

The following keyboard shortcuts are based in US standard keyboards.

Description	Shortcut
Context-sensitive help	F1
New file	Ctrl + N
Open file	Ctrl + O
Save file	Ctrl + S
Undo	Ctrl + Z
Redo	Ctrl + Y
Cut	Ctrl+X
Copy	Ctrl+C

Description	Shortcut
Paste	Ctrl+V
Delete	Del

Keyboard shortcuts for Model Design Accelerator

You can use keyboard shortcuts to navigate through and perform some tasks in Model Design Accelerator.

Description	Diagram shortcut
Autolayout as a tree	Ctrl + A
Autolayout as a star	Ctrl + S
Zoom in	Ctrl + +
Zoom out	Ctrl + -
Fit diagram to window	Ctrl + 0
Zoom to 100%	Ctrl + 1
Undo	Ctrl + Z
Redo	Ctrl + Y
Add directly related tables	Ctrl + D
Add recursively all tables linked by Many to One relationships	Ctrl + T
Add tables with One to Many	Ctrl + O
Add tables with Many to One	Ctrl + M
Remove selected tables	Del
Remove all tables except those selected	Shift + Del

Description	Model Accelerator shortcut
Rename	F2
Delete	Del
Zoom In	Ctrl + +
Zoom out	Ctrl + -
Fit diagram to window	Ctrl + 0
Zoom to 100%	Ctrl + 1
Undo	Ctrl + Z
Redo	Ctrl + Y

IBM and accessibility

See the IBM® Accessibility Center (<http://ibm.com/able>) for more information about the commitment that IBM has to accessibility.

Appendix D: Data Formatting Reference

This chapter contains definitions of data formatting properties found in IBM® Cognos® Framework Manager. The definition for each formatting property is also shown when you select a property in the **Data Format** dialog box in Framework Manager.

For SAP BW metadata:

- any unit of measure information that exists in SAP BW is automatically appended to the data value
- you cannot define a format for each currency in a multi-currency query subject.

Data Formatting Properties

The following is a list of properties available in the data formatting dialog.

"Not Applicable" Characters

Specifies the characters to be displayed when the value to be formatted was not applicable. The default value is two dashes (--). Note that the format will be applied only if the data source supports this error condition.

Any Error Characters

Specifies the characters to be displayed when the value to be formatted was not available because of an error. This property is overridden by the more specific formatting error conditions, such as Security Error Characters. The default value is two dashes (--). Note that the format will be applied only if the data source supports this error condition.

Calendar Type

Specifies the type of calendar to be displayed. The date values will be mapped to the selected calendar before being formatted. The default value is inherited from the user's content language. Note that the Japanese Imperial setting is only applicable for Japanese languages.

Clock

Specifies whether to display the time in 12-hour or 24-hour format. The default value is inherited from the user's content language.

Currency

Specifies the currency to be used. The default currency symbol will be displayed unless the values of the Currency Display and Currency Symbol properties are changed. The default value is inherited from the model.

Currency Display

Specifies whether to display the international or local currency symbol. By default, the local currency symbol is displayed.

Currency Symbol

Specifies a character or characters to use as the symbol to identify the local currency. This symbol will precede the number and any sign, even if it is a leading sign. A space between the symbol and the numeric value can be specified by entering it in this property, after the symbol. The default value is inherited from the user's content language.

Currency Symbol Position

Specifies where the currency symbol will appear. If End is selected, any spaces that follow the character or characters in the Currency Symbol or International Currency Symbol properties will be rendered between the number and the symbol. The default value is inherited from the user's content language.

Date Ordering

Specifies the order in which to display the day, month, and year. The default value is inherited from the user's content language.

Date Separator

Specifies the character to be displayed between the year, month, and day. The default value is inherited from the user's content language.

Date Style

Specifies the date style. The results rendered are determined by the language. Generally, Short uses only numbers, Medium uses some abbreviated words, Long uses complete words, and Full includes all available details.

Decimal Separator

Specifies the character that will separate non-decimal numbers from decimals. This property is ignored if no decimals are displayed. The default value is inherited from the user's content language.

Display AM / PM Symbols

Specifies whether to display the AM or PM symbols. The default value is inherited from the user's content language.

Display As Exponent

Specifies whether to render values in scientific notation, using exponents. If this property is set to No, scientific notation will not be used. If this property is not specified, scientific notation will be

used only when values exceed the maximum number of digits. The default value is inherited from the user's content language.

Display Days

Specifies whether to display the day. The format of the day can be controlled by selecting one of the specific formats. Selecting Julian means that the 3-digit day of the year will be displayed. The default value is inherited from the user's content language.

Display Eras

Specifies whether to display the era. The default value is inherited from the user's content language.

Display Hours

Specifies whether to display the hours. The default value is inherited from the user's content language.

Display Milliseconds

Specifies whether to display the milliseconds. The format of the milliseconds can be controlled by selecting one of the specific formats. This property is ignored if seconds are not displayed. The default value is inherited from the user's content language.

Display Minutes

Specifies whether to display the minutes. The format of the minutes can be controlled by selecting one of the specific formats. The default value is inherited from the user's content language.

Display Months

Specifies whether to display the month. The format of the month can be controlled by selecting one of the specific formats. The default value is inherited from the user's content language.

Display Months

Specifies whether to display the month.

Display Seconds

Specifies whether to display the seconds. The format of the seconds can be controlled by selecting one of the specific formats. The default value is inherited from the user's content language.

Display Time Zone

Specifies whether to display the time zone. The default value is inherited from the user's content language.

Display Weekdays

Specifies whether to display the weekday. The format of the weekday can be controlled by selecting one of the specific formats. The default value is inherited from the user's content language.

Display Years

Specifies whether to display the year. The first two digits of the year, which indicate the century, can be controlled by selecting one of the associated property values. The default value is inherited from the user's content language.

Display Years

Specifies whether to display the year.

Divide By Zero Characters

Specifies the characters to be displayed when a numeric value is the result of a division by zero. The default value is /0. Note that the format will be applied only if the data source supports this error condition.

Exponent Symbol

Specifies the character to be displayed to identify exponents if the scientific notation is used. The symbol will be rendered after the number, separated by a space. The default value is inherited from the user's content language.

Group Size (digits)

Specifies the primary grouping size. If a value is specified it represents the number of digits to the left of the decimal point to be grouped together and separated by the thousands separator. The default value is inherited from the user's content language.

International Currency Symbol

Specifies a character or characters to use as a symbol to identify the international currency. This symbol will replace the currency symbol. A space between the symbol and the numeric value can be specified by entering it in this property, after the symbol. The default value is inherited from the user's content language.

Mantissa (digits)

Specifies the number of digits to be displayed following the exponent symbol if the scientific notation is used.

Maximum No. of Digits

Specifies the maximum number of digits that can be displayed. If the maximum number of digits is not sufficient to display the value, a scientific notation will be used. The default value is inherited from the user's content language.

Minimum No. of Digits

Specifies the minimum number of digits that can be displayed. If the minimum number of digits is too high to display a value, the padding character will be used. The default value is inherited from the user's content language.

Missing Value Characters

Specifies the character or characters to be displayed when the value is missing. If no value is entered for this property, an empty string will be displayed.

Negative Pattern

Specifies a presentation format, based on patterns, for negative numbers. Some restrictions exist. The numerical part of the negative pattern is ignored. Only the suffix and the prefix are used. For example, in the pattern ABC###0.#EFG, ABC is the prefix, EFG is the suffix and ###0.# is the numerical part of the pattern.

Negative Sign Position

Specifies where the negative sign will appear. The default value is inherited from the user's content language.

Negative Sign Symbol

Specifies how to display negative numbers. The default value is inherited from the user's content language.

No. of Decimal Places

Specifies the number of digits to be displayed to the right of the decimal point. If this property is not set, the number of decimal places will vary depending on the number rendered.

Numeric Overflow Characters

Specifies the characters to be displayed when a numeric value is the result of a numeric overflow. The default value is two dashes (--). Note that the format will be applied only if the data source supports this error condition.

Padding Character

Specifies the character that will be used to pad values that have fewer digits than the minimum number of digits. The default value is inherited from the user's content language.

Pattern

Specifies a presentation format that is based on patterns. The pattern format overrides formats specified in other properties. For example, to format the date as 2009/12/31 23:59:59 PM, use the pattern yyyy/mm/dd hh:mm:ss aa. For example, to format thousands using the letter K, set the Format Type to Number, set the Scale to -3 (to remove 000), and then use the pattern to ####K.

Percentage Symbol

Specifies whether to display the values per hundred (percent) or per thousand. The symbol will be appended to the number and any trailing sign. A space between the numeric value and the symbol can be specified by entering it in this property, after the symbol. The default value is inherited from the user's content language.

Percent Scale (integer)

Scale to be applied to value after formatting. If omitted, no percent scale will be applied and the value will be formatted according to the normal decimal positioning associated with the percent (or per mille) symbol.

Scale

Specifies how many digits to move the decimal delimiter for formatting purposes. For example, move the decimal three spaces to present values in thousands. The default value is inherited from the database field.

Secondary Group Size (digits)

Specifies the secondary grouping size. If a value is specified it represents the number of digits to the left of the primary group that will be grouped together and separated by the thousands separator. If this property is left blank, the secondary grouping of digits is the same number as the primary group size, as specified by the Group Size (digits) property. The default value is inherited from the user's content language.

Security Error Characters

Specifies the characters to be displayed when the value to be formatted was not available for security reasons. The default value is #!Security. Note that the format will be applied only if the data source supports this error condition.

Thousands Separator

Specifies how to delimit digit groups, such as thousands. This property is only used if the Use Thousands Separator property is set to Yes. The default value is inherited from the user's content language.

Time Separator

Specifies the character to be displayed between the hour, minute, and second. The default value is inherited from the user's content language.

Time Style

Specifies the time style to be displayed. The exact results that will be rendered are determined by the language. Generally, Short means that the minimum details will be displayed, Long adds seconds, and Full means that all details are displayed, including the time zone. The default value is inherited from the user's content language.

Time Unit

Specifies the unit of measure of the value. This property will be ignored if any day or time components are shown. The default value is inherited from the user's content language.

Use Thousands Separator

Specifies whether the grouping delimiter will be applied as defined by the Group Size property. The default value is inherited from the user's content language.

Zero Value Characters

Specifies the character or characters to be displayed when the value is zero (0). If no value is entered for this property, the Maximum No. of Digits property determines how many zero digits are displayed.

Appendix E: Using Patterns to Format Data

You can format data so that it matches any pattern of text and numbers when default formats are not appropriate. For example, you can format dates to use full text including the era, or you can format them to only use numbers and show the last two digits of years to save space.

Using symbols and patterns can provide similar results as basic data formatting tasks. For example, you can set how many digits appear after the decimal point. You can achieve these types of results with a pattern, or you can set the **No. of Decimal Places** property. Patterns allow flexibility for more complex requirements.

Each supported content language code requires a specific set of symbols to be used in patterns. For each language code, there are two tables you will need; one for date and time symbols, and one for decimal symbols. The decimal symbols are the same for all locales, however, date and time symbols are grouped into six locale groups. Check the Date and Time Symbol section to see which locale group is used for your locale.

To define patterns, open the **Data Format** dialog box, and edit the **Pattern** property for each format type. Use the symbols that are defined in the language code tables, and follow these guidelines.

Pattern Guidelines

When you define a pattern, the number of symbols you use affects how the data will be shown. There are different rules for text, numbers, and values that can take the form of text or numbers.

Text

You can specify whether text is produced in full or abbreviated form.

Number of symbols	Meaning	Example
4 or more	Full text form	EEEE produces Monday
Less than 4	Abbreviated form	EEE produces Mon

Numbers

The number of symbols you use in a pattern sets the minimum number of digits that are produced in a report. Numbers that have fewer digits than specified are zero-padded. For example, if you specify mm for minutes, and the database value is 6, the report will show 06.

Note: The year value is handled differently. If you specify two symbols for year, the last two digits of the year value is produced. For example, yyyy produces 1997, and yy produces 97.

Text and Numbers

For values that can produce text or numbers, such as months, you can specify whether text or numbers are produced, and whether words are abbreviated.

Number of symbols	Meaning	Example
3 or more	Text	MMMM produces January MMM produces Jan
Less than 3	Numbers	MM produces 01 M produces 1

Date and Time Symbols

Date and time symbols are divided into locales, each of which is detailed below.

Locale Group A

Locales: af-za, en, en-au, en-be, en-bw, en-ca, en-gb, en-hk, en-ie, en-in, en-mt, en-nz, en-ph, en-sg, en-us, en-vi, en-za, fo-fo, gl-es, id, id-id, is, is-is, it, it-ch, it-it, kk-kz, ms, ms-bn, ms-my, nb-no, nl, nl-be, nl-nl, no, no-no, om-et, om-so, pl, pl-pl, pt, pt-br, pt-pt, so-dj, so-et, so-ke, so-so, sv, sv-fi, sv-se, sw-ke, sw-tz

Meaning	Symbol	Presentation	Example
Era	G	Text	AD
Year	y	Number	1996
Year (of 'Week of Year')	Y	Number	1996
Month in year	M	Text and number	July and 07
Week in year	w	Number	27
Week in month	W	Number	2
Day in year	D	Number	189
Day in month	d	Number	10
Day of week in month	F	Number	2 (2nd Wed in July)
Day of Week (1=first day)	e	Number	2
Day in week	E	Text	Tuesday

Meaning	Symbol	Presentation	Example
a.m. or p.m. marker	a	Text	pm
Hour in day (1 to 24)	k	Number	24
Hour in a.m. or p.m. (0 to 11)	K	Number	0
Hour in a.m. or p.m. (1 to 12)	h	Number	12
Hour in day (0 to 23)	H	Number	0
Minute in hour	m	Number	30
Second in minute	s	Number	55
Millisecond	S	Number	978
Time zone	z	Text	Pacific Standard Time
Escape used in text	'	n/a	n/a
Single quote	"	n/a	'

Locale Group B

Locales: be-by, bg-bg, el, el-gr, fi, fi-fi, hr, hr-hr, hu, hu-hu, ja, ja-jp, ko, ko-kr, ro, ro-ro, ru, ru-ua, ru-ru, sh-yu, sk, sk-sk, sl-si, sq-al, sr-sp, th, tr, tr-tr, uk-ua, zh, zh-cn, zh-hk, zh-mo, zh-sg, zh-tw

Meaning	Symbol	Presentation	Example
Era	G	Text	AD
Year	a	Number	1996
Year (of 'Week of Year')	A	Number	1996
Month in year	n	Text and number	July and 07
Week in year	w	Number	27
Week in month	W	Number	2

Meaning	Symbol	Presentation	Example
Day in year	D	Number	189
Day in month	j	Number	10
Day of week in month	F	Number	2 (2nd Wed in July)
Day of Week (1=first day)	e	Number	2
Day in week	E	Text	Tuesday
a.m. or p.m. marker	x	Text	pm
Hour in day (1 to 24)	h	Number	24
Hour in a.m. or p.m. (0 to 11)	K	Number	0
Hour in a.m. or p.m. (1 to 12)	k	Number	12
Hour in day (0 to 23)	H	Number	0
Minute in hour	m	Number	30
Second in minute	s	Number	55
Millisecond	S	Number	978
Time zone	z	Text	Pacific Standard Time
Escape used in text	'	n/a	n/a
Single quote	"	n/a	'

Locale Group C

Locales: ca-es, cs, cs-cz, da, da-dk, es, es-ar, es-bo, es-cl, es-co, es-cr, es-do, es-ec, es-es, es-gt, es-hn, es-mx, es-ni, es-pa, es-pe, es-pr, es-py, es-sv, es-us, es-uy, es-ve, eu-es, mk-mk

Meaning	Symbol	Presentation	Example
Era	G	Text	AD

Meaning	Symbol	Presentation	Example
Year	u	Number	1996
Year (of 'Week of Year')	U	Number	1996
Month in year	M	Text and number	July and 07
Week in year	w	Number	27
Week in month	W	Number	2
Day in year	D	Number	189
Day in month	t	Number	10
Day of week in month	F	Number	2 (2nd Wed in July)
Day of Week (1=first day)	e	Number	2
Day in week	E	Text	Tuesday
a.m. or p.m. marker	a	Text	pm
Hour in day (1 to 24)	h	Number	24
Hour in a.m. or p.m. (0 to 11)	K	Number	0
Hour in a.m. or p.m. (1 to 12)	k	Number	12
Hour in day (0 to 23)	H	Number	0
Minute in hour	m	Number	30
Second in minute	s	Number	55
Millisecond	S	Number	978
Time zone	z	Text	Pacific Standard Time
Escape used in text	'	n/a	n/a

Meaning	Symbol	Presentation	Example
Single quote	"	n/a	'

Locale Group D

Locales: de, de-at, de-be, de-ch, de-de, de-lu

Meaning	Symbol	Presentation	Example
Era	G	Text	AD
Year	j	Number	1996
Year (of 'Week of Year')	J	Number	1996
Month in year	M	Text and number	July and 07
Week in year	w	Number	27
Week in month	W	Number	2
Day in year	D	Number	189
Day in month	t	Number	10
Day of week in month	F	Number	2 (2nd Wed in July)
Day of Week (1=first day)	e	Number	2
Day in week	E	Text	Tuesday
a.m. or p.m. marker	a	Text	pm
Hour in day (1 to 24)	h	Number	24
Hour in a.m. or p.m. (0 to 11)	K	Number	0
Hour in a.m. or p.m. (1 to 12)	k	Number	12
Hour in day (0 to 23)	H	Number	0

Meaning	Symbol	Presentation	Example
Minute in hour	m	Number	30
Second in minute	s	Number	55
Millisecond	S	Number	978
Time zone	z	Text	Pacific Standard Time
Escape used in text	'	n/a	n/a
Single quote	"	n/a	'

Locale Group E

Locales: fr, fr-be, fr-ca, fr-ch, fr-fr, fr-lu

Meaning	Symbol	Presentation	Example
Era	G	Text	AD
Year	a	Number	1996
Year (of 'Week of Year')	A	Number	1996
Month in year	M	Text and number	July and 07
Week in year	w	Number	27
Week in month	W	Number	2
Day in year	D	Number	189
Day in month	j	Number	10
Day of week in month	F	Number	2 (2nd Wed in July)
Day of Week (1=first day)	e	Number	2
Day in week	E	Text	Tuesday
a.m. or p.m. marker	x	Text	pm

Meaning	Symbol	Presentation	Example
Hour in day (1 to 24)	h	Number	24
Hour in a.m. or p.m. (0 to 11)	K	Number	0
Hour in a.m. or p.m. (1 to 12)	k	Number	12
Hour in day (0 to 23)	H	Number	0
Minute in hour	m	Number	30
Second in minute	s	Number	55
Millisecond	S	Number	978
Time zone	z	Text	Pacific Standard Time
Escape used in text	'	n/a	n/a
Single quote	"	n/a	'

Locale Group F

Locales: ga-ie

Meaning	Symbol	Presentation	Example
Era	R	Text	AD
Year	b	Number	1996
Year (of 'Week of Year')	B	Number	1996
Month in year	M	Text and number	July and 07
Week in year	t	Number	27
Week in month	T	Number	2
Day in year	l	Number	189
Day in month	L	Number	10

Meaning	Symbol	Presentation	Example
Day of week in month	F	Number	2 (2nd Wed in July)
Day of Week (1=first day)	e	Number	2
Day in week	E	Text	Tuesday
a.m. or p.m. marker	a	Text	pm
Hour in day (1 to 24)	u	Number	24
Hour in a.m. or p.m. (0 to 11)	K	Number	0
Hour in a.m. or p.m. (1 to 12)	k	Number	12
Hour in day (0 to 23)	U	Number	0
Minute in hour	n	Number	30
Second in minute	s	Number	55
Millisecond	S	Number	978
Time zone	c	Text	Pacific Standard Time
Escape used in text	'	n/a	n/a
Single quote	"	n/a	'

Decimal Format Symbols

All locales

Symbol	Meaning
0	A digit that is shown even if the value is zero.
#	A digit that is suppressed if the value is zero.
.	A placeholder for decimal separator.

Symbol	Meaning
,	A placeholder for thousands grouping separator.
E	Separates mantissa and exponent for exponential formats.
;	Separates formats for positive numbers and formats for negative numbers.
-	The default negative prefix.
%	Multiplied by 100, as percentage.
‰	Multiplied by 1000, as per mille.
¤	The currency symbol. If this symbol is present in a pattern, the monetary decimal separator is used instead of the decimal separator.
¤¤	The international currency sign. It will be replaced by an international currency symbol. If it is present in a pattern, the monetary decimal separator is used instead of the decimal separator.
X	Other characters that can be used in the prefix or suffix.
'	Used to quote special characters in a prefix or suffix.
/u221E	Infinity symbol.
/uFFFD	Not a Number symbol.

Appendix F: Guidelines for Working with SAP BW Data for Use in Transformer

Starting with Transformer version 8.4, you can use Framework Manager packages published to Content Manager to leverage your SAP BW data. The SAP-based packages can be used as data sources to create Transformer models. As result, Transformer PowerCubes can be used as high speed data access cache methods for distributing smaller or focused areas of your business information. This is the recommended method for leveraging your SAP BW data.

There are special considerations when using SAP-based packages created in Framework Manager. For detailed information about creating your SAP queries, creating the SAP-based packages in Framework Manager and using them in Transformer, see ["Working with SAP BW Data Using a Package in Framework Manager"](#) (p. 593). For general information about creating packages in Framework Manager, see "Create or Modify a Package" in the Framework Manager *User Guide*.

Transformer versions 7.x, 8.1, 8.2 and 8.3

In Transformer versions 7.x, 8.1, and 8.2, you can leverage your SAP BW data using a Framework Manager package in which the query subjects and dimensions are externalized using CSV files. Transformer can use the CSV files as a data source to create a model and generate PowerCubes. This method should only be used in a IBM® Cognos® environment when you want to leverage data in IBM Cognos BI to build PowerCubes for IBM Cognos BI.

In Transformer version 8.3, use published packages for dimensions and use CSV files for facts.

There are special considerations when using externalized CSV files with SAP data in Framework Manager. For more information, see ["Working with SAP BW Data Using Externalized CSV Files in Framework Manager"](#) (p. 602). For general information about externalizing query subjects and dimensions using the CSV method, see "Externalizing Query Subjects and Dimensions" in the Framework Manager *User Guide*.

Working with SAP BW Data Using a Package in Framework Manager

You can leverage SAP BW data in Transformer by using an SAP-based package created in Framework Manager and published to Content Manager. This is the recommended method to leverage your SAP BW data. There are special considerations when using SAP-based packages created in Framework Manager.

You can use Transformer to import both dimensional and fact data from an SAP BW query source. The following instructions describe how to rebuild an SAP BW cube as an IBM® Cognos® Transformer cube. To do so, the SAP BW query package must be in a specific format.

There are three stages to importing a SAP BW query to access both dimensions and facts using IBM Cognos BI:

- [Creating a Query in SAP BW Business Explorer Query Designer](#)
- [Creating a Package in Framework Manager](#)
- [Creating a Model in Transformer](#)

Limitations

- This extract process is limited to SAP BW data sources only.
- The data source must be a specifically constructed query defined in the SAP BW data source.


For general information about creating packages, see "Create or Modify a Package" in the Framework Manager *User Guide*.

Creating a BW Query in SAP Business Explorer Query Designer

You must create a query that includes the cube that you wish to import. We recommend that you base the query on a single InfoCube in the database. A query based on multiple sources may result in SAP BW errors during data retrieval.


After creating the query, you can create a variable ([p. 595](#)).

Steps

1. In **Query Designer**, click **New Query**.
2. In the **New Query** dialog box, select the information provider that contains the cube that you want to import.
3. Click the **Tools** icon  to view the technical name of the **InfoObject**.
4. Drag a characteristic that you wish to import from the **InfoObject** catalog on the left column to one of the fields on the right-hand side of the page. For example, **Columns** or **Rows**.

The characteristics you select will define the metadata in the Transformer cube. The characteristics must adhere to the following restrictions:

- You must have at least a single optional variable to segment the data.
- Select a characteristic that is representative of the data source. The characteristics can be either key figures, which will become measures in Transformer, or dimensions, which will become the Transformer dimensions.
- Do not assign any of the characteristics a display hierarchy, either explicitly or by a variable.
- All key figures in the SAP BW query must be numeric.
- Do not select the **Currency/Unit** characteristic.
- Ensure that all selected key figures use the same currency.
- Only include characteristics in the SAP BW query that you wish to extract using Framework Manager. Including unnecessary characteristics increases data volume, thereby adversely affecting performance.

- Characteristics must be copied to the **Columns** or **Rows** fields of the query definition. If copied to the **Free Characteristics** or the **Filter** fields, the characteristics show as dimensions when importing from the package but the stream extract processing is not able to fetch the values.
 - If you have filters defined, they must reference only dimensions that have been included elsewhere in the query definition.
 - If you include a free characteristic, no values will appear for that characteristic in the key figures extract. A filter on a free characteristic acts as a filter on the returned SAP BW data. You can use this as a filter to define a subset of an InfoCube.
 - Use a picklist prompt, rather than a type-in prompt for the query. A picklist prompt provides values for segmenting the data.
5. To define the metadata that will populate the Transformer cube, you must change the properties of each characteristic that you have selected for inclusion. Right-click a characteristic, and select **Properties**.
 6. In the **Properties of Characteristic** dialog box, change the **Display As** value to **Key**, and the **Suppress Results Rows** value to **Always**. Note that any restriction or filter applied here will be carried forward in Transformer.
 7. Repeat steps 5 and 6 for each characteristic that you selected in step 4.
- Note:** You should only select the characteristics that you require. To avoid excessive memory consumption, and decreased system performance or failure, carefully consider what characteristics you want to include in the query. We recommend that you consult an SAP BW administrator to ensure that the data volumes are not exceeded.
8. Click the **Queries Properties** icon , and in the **Extended** tab select the **Allow External Access to this Query** check box. This exposes the query to Framework Manager.
 9. Click **Save**, and provide the new query with a **Description** and a **Technical Name**. We recommend that you use the SAP BW naming convention in the **Technical Name** field. That is, begin the entry with the letter 'Z' followed by an intuitive name or your standard naming convention. It is important to write down this technical name, as you will need it to find the query in Framework Manager.

You are now ready to create a variable ([p. 595](#)). For more information on using the **SAP Query Designer**, see your SAP BW documentation.

Create a Variable

You must now create an optional prompt parameter for the query so Transformer can issue smaller queries to SAP, and thereby retrieve the entire data set.

Guidelines for Using SAP BW Fact Data in Transformer

There are no set rules for variable usage when extracting SAP BW data for use in Transformer. However, you must be careful not to request too much data that could potentially perform poorly or error out with out-of-memory messages within your SAP environment.

A basic guideline to follow is that when a variable is utilized for the extraction, Transformer will first fetch all members that exist for the dimension against which the variable is defined. After this, Transformer will perform individual data fetches to extract the fact data for each of the individual members within the dimension in order to satisfy the variable.

This allows Transformer to break down your data extraction into manageable chunks that the SAP BW server can handle. There are no set standards as to which dimension to apply it to. To achieve optimal performance, you must understand your SAP BW data and determine which dimension evenly breaks up the factual data.

You must choose carefully which dimension to define the variable on. It may require some experimentation to achieve optimal performance. For example, you may have a [COUNTRY] dimension that contains three countries as members, United States (US), Canada (CA), and Mexico (MX). If most of the business is performed in the US (90%) and the remaining business (10%) is recorded against Canada and Mexico evenly, this dimension would not evenly split up the data. The resulting queries would have one very large request (US) and two small ones (CA and MX). Therefore, this dimension would not be a good candidate.

You do not want to apply a variable on a dimension that would cause too many very small requests. For example, [0MATERIAL], a dimension often utilized in SAP BW environments would probably not be a good candidate because it would cause too many small requests to be performed.

You may have a dimension defined for [COSTCENTER] that evenly divides up the data for 10 distinct cost centers that may serve to segment the data evenly. Another good alternative may be calendar year or calendar month because it may divide your data into sections that perform adequately.

It is not necessary to apply any variables to queries for data extraction. Some extraction will perform perfectly well when no variables are applied. For example, a good approach may be to apply a variable on a dimension which splits the data into 20 individual fetches and test the extraction. If this performs well, you may choose to apply a variable on a different dimension which may contain 5 distinct members and see how it compares.

No formula can be applied as no two environments are alike. However, a cautious approach is recommended to avoid disrupting your SAP BW environment.


Steps

1. In **Query Designer**, right-click a characteristic that you have selected in the previous procedure and select **Restrict**.

To ensure that data is distributed evenly, select a characteristic that is representative of the cube and will not result in a large number of values. You want a resulting variable where the number of rows for each value of the variable is similar; you do not want a resulting variable that is too fine-grained (for example, not many rows per value resulting in an excessive number of queries), nor do you want a variable that is too coarse-grained (for example, more than one million rows per value).

2. In the **Selection for ...** dialog box, click the **Variables** tab, right-click anywhere inside the **Description** window and select **New Variable**.

Note: If one of the characteristics that you have chosen already has a variable, you can avoid creating a new variable and skip to step 7 of this procedure.

3. In the **New Variable Wizard General Information** page, type a **Variable Name** and **Description**, and select a dimension as the characteristic. Click **Next**.
4. In the **Details** page, select **Single Value**, **Multiple Single Values**, or **Interval** in the **Variable Represents** field, **Optional** in the **Variable entry is** field, and select the **Ready for Input** check box. Click **Next**.
5. In the **Default Values** page, ensure that the **Default Value** field is empty.
6. Click **Next** until you are returned to **Selection for ...** dialog box. The new variable appears in the **Description** window.
7. Select the variable and click the right arrow  to move the selected variable over to the **Selection** window, and save the query. You are now ready to import the query in Framework Manager.

Creating a Package in Framework Manager

To create a package in Framework Manager you must

- Import the SAP BW metadata using the MetaData Wizard ([p. 598](#))

Framework Manager imports the SAP BW query into a model, and defines a package that it exports to Content Manager.

When importing, note the following:

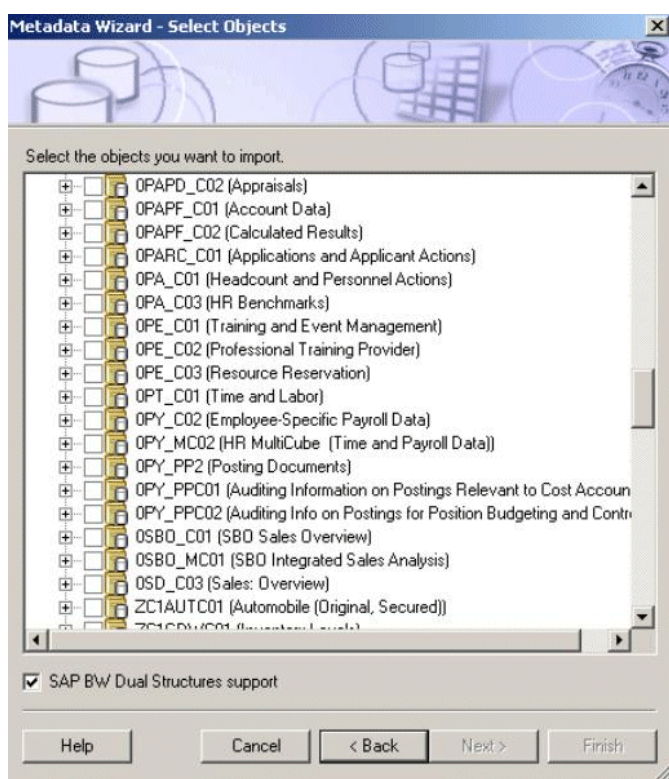
- The dimensions selected in the SAP BW query are available in the **Dimension Folders** in the **Import** dialog box.
- Each dimension will contain at least one hierarchy.
- Always select the primary hierarchy whose name matches the hierarchy.
- If other hierarchies are available, select one that gives the desired set of levels within the hierarchy.
- Framework Manager imports time dimensions into the model from the SAP BW data source only if a configuration parameter is turned on. Setting the configuration as a time dimension is a global entry; every imported dimension will then be treated as time strings.
- Create a package ([p. 599](#))

When creating the package for publishing to Content Manager, hide the primary hierarchy in those dimensions where you imported two hierarchies. The primary hierarchy is necessary, and

must be in the package for querying to work correctly. You can hide the hierarchy if you don't want it visible.

Steps for Importing Using the Metadata Wizard

1. In Framework Manager, click **Create a new project**.
2. Complete the fields in the **New Project** dialog box. Click **OK**.
3. Complete the steps in the **Metadata Wizard**. When prompted to select a data source, if you need to create a new data source, click **New...**
4. In the **Select Objects** page, locate the query that you defined in SAP BW query Designer in the previous stage (p. 594). Scroll the list for the technical name that you provided when you created the variable. The folder structure is as follows: Hierarchies > Level definitions > Query Item definitions.



5. Select the main query items that directly relate to the level. That is, those labeled **(Key)**, **(Name)**, and so on.

Tip: Secondary or additional attributes are removed on import to Transformer. Only items that are needed are imported. However, to improve performance, we recommend that you do not select secondary or additional attributes. If you select all the attributes here, you can exclude unwanted query items when publishing the package.

6. Complete the remaining screens in the **Metadata Wizard**, accepting the default values, and click **Next**. This will generate dimensions and import the metadata.

- At the final wizard screen, verify the results, and click **Finish**.

Steps for Creating a Package

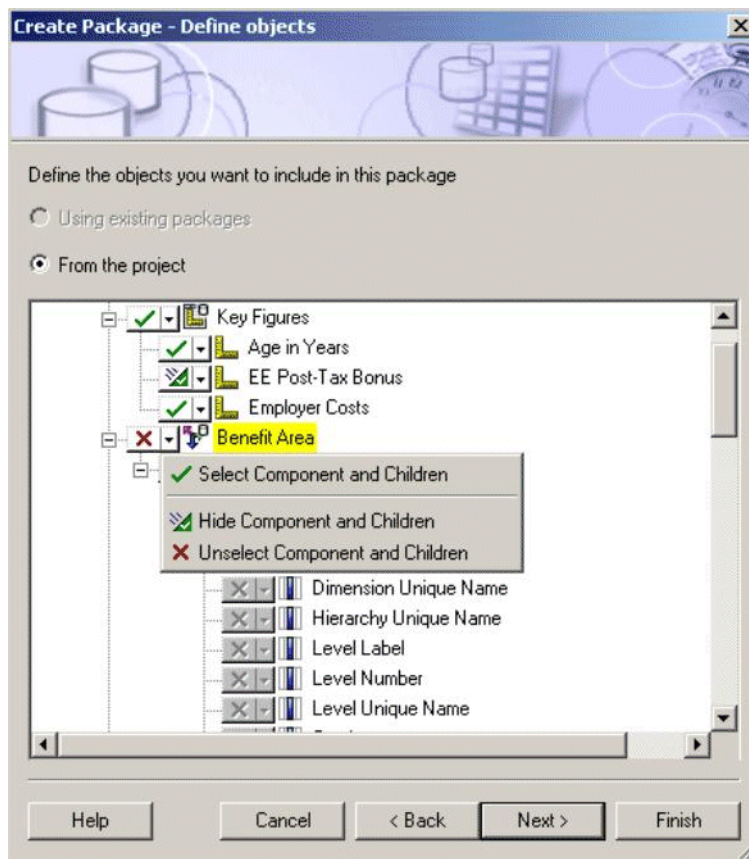
- Click the **Packages** folder, and from the **Actions** menu, click **Create, Package**.
- In the **Provide Name** page, type the name for the package and, if you want, a description and screen tip. Click **Next**.
- Select the query that you imported in the previous section.

For more information, see ["Creating a BW Query in SAP Business Explorer Query Designer" \(p. 594\)](#).

- In the **Define objects** page, when hiding or excluding child objects from the package, you must select each of them individually. Excluding parent objects also exclude all of its children. Note that excluding (or unselecting) many objects from larger cubes will require a significant amount of time.

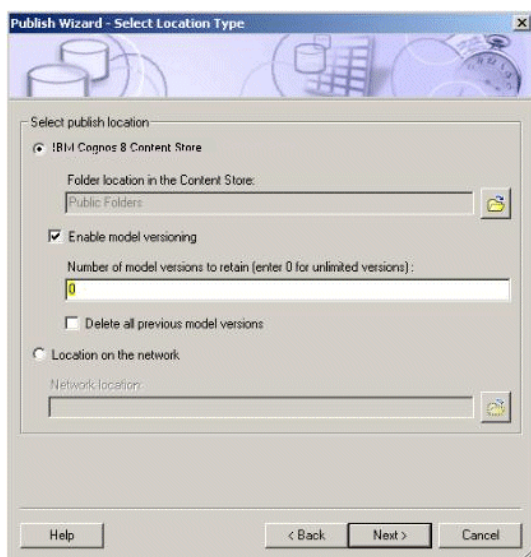
Note: Framework Manager supports ctrl+shift and alt+shift functionality. Use these keystrokes to select multiple objects that you wish to include or hide in the cube. For example, if you wish to only include two items in a large branch, select the entire branch, then use ctrl+shift to de-select the items you wish to include, and hide the remaining selected items.

For more information about including, excluding and hiding objects, see "Create or Modify a Package" in the Framework Manager *User Guide*.



- Choose whether to use the default access permissions for the package:

- To accept the default access permissions, click **Finish**.
 - To set the access permissions, click **Next**.
6. When you are prompted to open the **Publish Package Wizard**, click **Yes**.



7. Select the default values, and click **Publish**. This will publish the package to the content store, and will allow you to access the package in Transformer.
8. At the final screen verify the results, and click **Finish**.

You are now ready to create a model in Transformer. For more information on creating a package, see "Create or Modify a Package" in the *Framework Manager User Guide*.

Creating a Model in Transformer

Use Transformer to access a published SAP-based package and use it as a data source to create a model. After the model is created, you can create PowerCubes for use with the desired IBM® Cognos® component, accessing the dimensional and fact data from the original SAP BW source. In addition, you can combine the SAP metadata in a Transformer model with other corporate metadata or personal sources provided you have the necessary matching key information to join the data during cube building.

When you create the Transformer model, you must use the **Insert Dimension from Package** wizard rather than the **New Model Wizard**. You use the **Insert Dimension from Package** wizard because it

- creates a single query for each dimension and for the facts.
- imports facts and dimensions in the same manner as dimensionally-modeled relational models. That is, facts and dimensions are imported at the same time.
- ensures that the scope is set properly between the dimensions and facts.
- populates the dimension with the appropriate business key and caption information.

- only imports the necessary items from the BW package required for cube building, when the metadata is imported. This reduces the number of attributes and keeps the data volumes to only the necessary items for cube building.

If you want to define business rules, do so in the Transformer model rather than in Framework Manager. Calculations that you define in Framework Manager are not imported into Transformer.

Steps

1. In Transformer, click **Create a new model**.
2. In the **New Model Wizard**, click **Cancel**.
3. With the **Dimension Map** pane selected, from the **Edit** menu, click **Insert Dimension from Package**.
4. Click **Browse** to open the metadata browser.
5. In the **Browse Metadata** dialog box, select the package that contains your SAP BW query and click **OK**.
6. In the **Insert Dimension from Package** dialog box, click **Finish**.
7. In the **Select Dimension and Measures from Package** dialog box, click the dimensions and measures to include in the data source.

Select a query item that will provide the dates for the PowerCube. Note that the dates for the PowerCube can be derived entirely from the transaction data.

8. If there are errors or warnings, you are notified. In the **Data Sources** pane, expand the package to view the data source queries and query items. Key figures or measures appear in the **Measures** pane.

Ensure that the aggregation rule for each measure is correctly defined within Transformer to align as closely as possible with the aggregation rule defined in SAP BW.

It is recommended that the storage type for all measures be set to 64-bit floating point.

For the root level of each characteristic (dimension), ensure it is marked as unique.

SAP BW presentation hierarchies may contain ragged paths, typically in association with the "not assigned" and "#" nodes in the hierarchy. The gaps in these hierarchies produce blanks at the associated level in the Transformer hierarchy. In Transformer, it is possible to define the text that should be used for blanks (the default text is "<blank>"). A best practice is to define a more appropriate text for blank entries for all such levels.

9. If you want to add another query, repeat steps 3 to 7.

Tip: If you add a new dimension to a model after adding the measures, Transformer does not automatically add the key for the new dimension to the Key Figures. As a result, the scope is not defined in the Dimension Map. In such situations, you must manually add the key for the new dimension from the lowest level of the dimension to the Key Figures.

10. Save the model.

You can now use this model to create PowerCubes for use with the desired IBM Cognos component, accessing the dimensional and fact data from the original SAP BW data source. For more information, see the section "Create a Model" in the Transformer *User Guide*.

Working with SAP BW Data Using Externalized CSV Files in Framework Manager

When you externalize query subjects and dimensions into formats that you can use in other applications, there are special considerations. When extracting data from SAP BW using Framework Manager, you must understand the distinction that Framework Manager makes between different types of dimensions. Each type of dimension exhibits a different behavior when it is externalized, and can be modified before externalizing.

In Transformer versions 7.x, 8.1, and 8.2, you can leverage your SAP BW data using a Framework Manager package in which the query subjects and dimensions are externalized using CSV files. Transformer can use the CSV files as a data source to create a model and generate PowerCubes. CSV files are also supported in Transformer version 8.3 but it is recommended that you use package support for dimensional data and CSV files for fact data.

In this version of Transformer, using Framework Manager packages published to Content Manager is the preferred method to leverage SAP BW data. For general information about creating packages, see "Create or Modify a Package" in the Framework Manager *User Guide*. For SAP-specific information about creating packages, see ["Working with SAP BW Data Using a Package in Framework Manager"](#) (p. 593).

Extract Size

The **Extract Size** data source property within Framework Manager controls the amount of data retrieved from SAP BW at any one time.

If this setting is negative, zero, or empty, a single query is issued to SAP BW to extract the characteristic data.

If this setting is a positive value, Framework Manager issues multiple queries to SAP BW, each of which returns approximately the number of megabytes specified by the **Extract Size** property.

This feature can reduce the overall size of the query result on the SAP BW server. Overall query execution may take longer, but for large characteristics, not using this feature may result in consumption of a user's allotted memory space on the SAP BW server.

The entire data for a characteristic dimension will be in memory within Framework Manager prior to the production of an extract file. It is important that only the required query items be extracted from SAP BW to ensure that an extract does not fail due to memory allocation errors within Framework Manager.

Model query subjects are extracted using the same mechanism by which queries are executed within IBM® Cognos®. Therefore, the **Extract Size** property has no effect on the query execution.

Measure Dimensions

When extracting a measure dimension, you should create a model query subject containing the measures that you want. You should include the business key query item from each of the levels of each dimension, depending on the level of granularity that you are trying to achieve.

For information about externalizing model query subjects, see ["Framework Manager Considerations"](#) (p. 605).

Characteristic Dimensions

Characteristic dimensions are externalized independent of the type of SAP BW data source, such as InfoCube or SAP BW query.

Framework Manager uses a single approach to externalize all dimensions that do not contain fact query items. In these cases, the extract size configuration setting is used to control the manner in which data is extracted from SAP BW.

Note: Model query subjects are externalized in a different manner, regardless of whether they contain fact query items or not. For information about externalizing model query subjects, see ["Framework Manager Considerations"](#) (p. 605).

Key Figures Dimensions from an SAP BW InfoCube

When externalizing the key figures dimension from a model based on an InfoCube, Framework Manager uses exactly the same approach as used for externalizing model query subjects.

For an InfoCube containing more than a few thousand transactions, externalizing an InfoCube directly from Framework Manager can easily exceed both time and memory limits on either the client or server. In such cases, it is highly recommended that an SAP BW query be used as the basis for externalizing the SAP BW metadata.

Key Figures Dimensions from an SAP BW Query

Using a BEx query as the basis for externalizing key figures from an SAP BW data source is, in most cases, the best approach. By using a BEx variable to break the data of the key figure dimension into manageable sections, arbitrarily large volumes of transaction data can be extracted from SAP BW.

Note, however, that this approach incurs some restrictions as to what can be extracted from SAP BW, and how it can be extracted. The remainder of this section describes how an SAP BW query is used to extract data from SAP BW, including all known restrictions and limitations.

SAP BW Query Requirements

For the remainder of this section, we assume that an SAP BW query is being used as the basis for externalizing the data, not as the basis for reporting, and not with the intent of exceeding the memory and time limitations associated with extracting data directly from an InfoCube.

It is not possible to externalize an arbitrary SAP BW query. An SAP BW query must adhere to the following restrictions if you want to externalize it:

- Set the characteristic display to **Key**. Setting the display to anything else may result in incorrect data.

To change what appears for a characteristic, right-click the characteristic and click **Properties**. In the **Properties of Characteristic** dialog box, change the **Display As** value to **Key**.

We strongly recommend that you use **Key**.

- To reduce data volumes, as well as the amount of aggregation performed by the SAP BW server, we strongly recommend that summarization for all characteristics in the query be disabled in its property sheet.

To disable summarization for a characteristic, right-click the characteristic along the edge of the SAP BW query and click **Properties**. In the **Properties** dialog box, set the **Suppress Results Rows** value to **Always**.

- If at least one characteristic in an SAP BW query is displayed as something other than **Key**, then summarization for all characteristics must be suppressed.
- The query must not contain the **Currency/Unit** characteristic.
- None of the characteristics may be assigned a display hierarchy, either explicitly or by a variable.
- If a characteristic is included in an SAP BW query as a free characteristic, no values will appear for that characteristic in the key figures extract.

A filter on a free characteristic acts as a filter on the data returned by SAP BW. It is an efficient mechanism for defining a subset of an InfoCube.

Such a filter may also be applied to a characteristic along an axis of an SAP BW query, in which case the filtered values appear in the key figures extract.

- All key figures in the SAP BW query must be numeric.
- The values of each key figure should be in a single currency. A variable should not be used to drive the assignment of a target currency.
- Include in the SAP BW query only those characteristics which are to be extracted using Framework Manager. Including unnecessary characteristics increases the volume of data transferred from SAP BW, thus affecting performance.

Guidelines and Constraints When Working with SAP BW Cubes

You must use CSV files when importing metadata from SAP BW cubes. For performance reasons, we recommend that you filter on geography, time periods, or some other dimension that limits the amount of data retrieved. Remember to apply your dimension filter to the related dimensions and their fact tables (measures). For more information, see "Create a Filter" in the *Framework Manager User Guide*.

Because SAP BW cubes are multidimensional, rollups are applied at the source. If you change the rollup type after importing the data into Transformer, your results will not be valid.

Missing data or metadata that is out-of-scope for a particular measure may yield different results, depending on the context. You may see:

- NULL values
- # symbols

- REST_H
- Not assigned

Because such duplicate tokens can cause problems in Transformer, in unique levels for example, we recommend that you assign filters to the dimension so that they do not appear in the imported data.

Finally, remember to select only those query items needed to generate your filtered data.

Framework Manager Considerations

When extracting the measure dimension from an SAP BW query, the **Extract Size** property of the data source controls the amount of data retrieved from the SAP BW server at one time. Model query subjects are externalized in a different manner, regardless of whether they contain fact query items or not. In this scenario, the setting has no affect on the SAP BW server, but it does limit the amount of memory Framework Manager allocates at any one time to retrieve the data.

Note that filters defined on the key figures dimension are not enforced when extracting data from an SAP BW query. To obtain performance benefits of extracting data from an SAP BW query, filters must be defined in an SAP BW query.

In addition, any calculations defined within the key figures dimension are ignored. These may be defined either within the SAP BW query in BEx, or in a model query subject in Framework Manager.

Each characteristic extracted must contain at least one query item from the lowest level of its hierarchy (if there is one) to provide linkage with the key figures extract. You should include the business key query item from each of the levels of each dimension, depending on the level of granularity that you are trying to achieve.

Use of Variables to Externalize Key Figures from an SAP BW Query

The volume of transactions within an SAP BW query is such that, in most cases, the use of a single query to extract the data from SAP BW will exceed the memory allocated to a user on an SAP BW server. In Framework Manager, you can use a single optional variable to extract fact data from an SAP BW query in reasonably sized sections.

To use this feature, one characteristic included in the SAP BW query (but not included as a free characteristic) is assigned a variable that conforms to the following restrictions:

- It must be a single value.
- It must be optional.
- It must not have a default value.
- It can be defined on the characteristic or a presentation hierarchy.

If an SAP BW query contains such a variable and the key figures dimension is externalized, Framework Manager runs a query for each possible value associated with a variable. Thus, by choosing an appropriate characteristic, the key figures dimension can be extracted without exceeding the memory restrictions of either the client or server. Memory caches on the client and server are flushed after each query.

If a presentation hierarchy is used to drive the creation of extract sections, it is important that the values for a variable be obtained from a single level in the hierarchy, otherwise the extract will contain data summarized at different levels. To restrict the values for a variable to a single level of a hierarchy, edit the Level Restriction of the variable in Framework Manager. For example, using a value such as "2:2" indicates that only values from the second level of the hierarchy are to be used (level 0 is the root of a hierarchy).

In the presence of an SAP BW query with one such variable, the value of the variable is reset after each query.

If an SAP BW query contains anything more than a single variable, or one that is defined differently than described above, Framework Manager does not attempt to use a variable to break the extraction of the key figures dimension into smaller sections.

Workaround for Problems Encountered While Externalizing

When externalizing a data source from Framework Manager, you may encounter an authentication error if

- the model is published to Content Manager
- externalizing the data takes longer to perform than the timeout period assigned to passports within IBM® Cognos® Configuration

Users are not prompted to re-enter their authentication credentials.

If an error occurs, the externalized data is still complete and valid. However, if the modeler chooses to actually publish the model, the modeler must re-authenticate and re-publish the model, but without externalizing the data.

Another solution is to publish the model to the network, in which case the authentication error does not occur.

Building PowerCubes from SAP BW Data

You can build IBM® Cognos® PowerCubes from SAP BW data. There are guidelines to consider for both Framework Manager ([p. 606](#)) and Transformer ([p. 607](#)).

Framework Manager Guidelines

When externalizing data for the purpose of creating one or more PowerCubes, keep these considerations in mind.

- The extract of each characteristic must have a common key query item that is equivalent to a surrogate key query item in the key figures extract.
- For an extract based on an SAP BW query, it is strongly recommended that all characteristics be displayed as Key in the SAP BW query.
- If a characteristic does not have a presentation hierarchy, or a new one is desired, extract one or more query items that can form the basis for levels in a hierarchy.

- During the import of SAP BW metadata into a model that will extract data, limit the model to only those query items that are absolutely required to build a PowerCube. This will improve data extract performance.
- Null values are included in CSV files when externalizing SAP BW-based query subjects and dimensions.
- A practical limit for PowerCubes is 2,000,000 categories (values) for a dimension (characteristic).

Transformer Guidelines

When using the SAP BW data that you extracted from Framework Manager, keep these considerations in mind.

- In Transformer version 8.3, you can insert regular dimensions from SAP data sources directly from a IBM® Cognos® data source, using the **Insert dimension from package** option.
- Using the model wizard in Transformer, insert a data source of type Delimited-Field Text With Column Titles and start by selecting the CSV file. Do not run auto-design.
- Drag all the key figure columns from the Data Sources pane into the Measures pane. Ensure that the aggregation rule for each measure is correctly defined within Transformer to align as closely as possible with the aggregation rule defined in SAP BW.
- It is recommended that the storage type for all measures be set to 64-bit floating point.
- Using the date wizard, select a query item that will provide the dates for the PowerCube. Note that the dates for the PowerCube can be derived entirely from the transaction data.
- Insert the various CSV files corresponding to the characteristics that were externalized using Framework Manager.

Each CSV file contains a column that corresponds to a column in the key figures CSV file. By right-clicking the various columns and editing the column properties, ensure the columns that provide the linkage between a characteristic and the key figures have the same name. For example, if a key figure column is named Customer and the corresponding column in the customer CSV file is named Customer - Key, then the name of the column in the key figures CSV file can be changed to Customer - Key.

- For each characteristic, create a new dimension, using the key columns, or other attributes of a characteristic, to drive the levels of the dimension. For each level, ensure that the properties for the label, short name, and description are assigned source columns, if applicable.
- For the root level of each characteristic (dimension), ensure it is marked as unique.
- SAP BW presentation hierarchies may contain ragged paths, typically in association with the "not assigned" and "#" nodes in the hierarchy. The gaps in these hierarchies produce blanks at the associated level in the Transformer hierarchy.

In Transformer, it is possible to define the text that should be used for blanks (the default text is "<blank>"). A best practice is to define a more appropriate text for blank entries for all such levels.

Appendix G: Reserved Words

You must ensure that the names of data sources, tables, and columns do not use names reserved by IBM® Cognos® software.

If you must use a reserved word, enclose the word in quotes in the SQL specification. For example, `select Orderdate, "Timezone".`

The following words are reserved:

Range	Reserved Words
A to C	abs, all, and, any, as, asc, at, avg, between, bigint, bit_length, boolean, both, by, call, case, cast, ceil, ceiling, char, char_length, character, character_length, coalesce, count, create, cross, cube, cume_dist, current, current_date, current_time, current_timestamp, cursor
D to M	date, day, dbkey, dec, decimal, declare, delete, dense_rank, desc, distinct, double, else, end, escape, except, exists, exp, extract, false, filename, filter, first, first_value, float, floor, following, for, from, full, group, grouping, hash, having, hour, in, inner, insert, int, integer, intersect, interval, into, is, join, last, last_value, leading, left, like, ln, local, localtime, localtimestamp, loop, lower, max, merge, min, minute, mod, month
N to Q	national, natural, nchar, no, not, ntile, null, nullif, nulls, numeric, nvarchar, octet_length, of, on, or, order, others, out, outer, over, partition, perc, percent_rank, percentile, percentile_cont, percentile_disc, position, power, preceding, precision, qualify
R to T	range, rank, ratio_to_report, ravg, rcount, rdiff, real, recursive, returning, right, rmax, rmin, rollup, row, row_number, rows, rsum, scroll, second, select, set, sets, smallint, snapshot, some, sqrt, stddev, stddev_pop, stddev_samp, substring, sum, table, tertile, then, time, timestamp, timezone_hour, timezone_minute, to, top, trailing, trim, true,

Range	Reserved Words
U to Z	unbounded, union, unknown, updatable, update, upper, user, using, values, var_pop, var_samp, varchar, variance, varying, when, where, window, with, within, without, xavg, xcount, xfirst, xlast, xmax, xmin, xmovingavg, xmovingsum, xntile, xperc, xrank, xratio, xstddev, xstddev_pop, xsum, xtertile, xvariance, xvariance_pop, year, zone

The following are also reserved words: _cursor, _local and _rowset.

Appendix H: XML Data Types

You can import XML as a tabular data source in IBM® Cognos® Framework Manager.

The following data types are supported when importing from XML into Framework Manager.

- boolean
- byte
- date
- dateTime
- decimal
- double
- ENTITIES
- ENTITY
- float
- ID
- IDREF
- int
- integer
- language
- long
- Name
- NCName
- negativeInteger
- NMTOKEN
- NMTOKENS
- nonNegativeInteger
- NonPositiveInteger
- NOTATION
- positiveInteger
- QName

- short
- string
- time
- token
- unsignedLong
- unsignedInt
- unsignedShort
- unsignedByte

The following data types are not supported.

- base64Binary
- duration
- gYearMonth
- gYear
- gMonthDay
- gMonth
- hexBinary

Glossary

access permission

A privilege that permits the access or use of an object.

alias

An alternative name used instead of a primary name.

attribute

In dimensional models, a property that provides qualitative information about members of a level in a dimension. For example, the Store level within the Retailer dimension might have properties such as address or retail space. In general, dimensional attributes do not have measure values or rollups associated with them, but are used to locate or filter members.

In relational models, a query item that is not a measure or identifier. When a query item is an attribute, it is not intended to be aggregated, or used for grouping or generating prompt pick lists.

In BI modeling, a characteristic of an entity which is descriptive rather than a unique identifier or an aggregative measure.

In TM1, a property that provides qualitative information about dimensions.

calculated member

A member of a dimension whose measure values are not stored but are calculated at run time using an expression.

cardinality

For OLAP data sources, the number of members in a hierarchy. The cardinality property for a hierarchy is used to assign solve orders to expressions.

For relational data sources, a numerical indication of the relationship between two query subjects, query items, or other model objects.

conformed dimension

A dimension, with a single definition, that is reused or shared across multiple data subject areas. The common definitions for common dimensions allow data from different subject areas to be meaningfully compared and used in calculations and drill throughs from one area to another.

In Data Manager,

a dimension with a single definition that can be reused or shared across multiple coordinated data marts.

cube

A multidimensional representation of data needed for online analytical processing, multidimensional reporting, or multidimensional planning applications.

data source

The source of data itself, such as a database or XML file, and the connection information necessary for accessing the data.

In TM1®, the file or data used as the source for the TurboIntegrator import process.

dimension

In Cognos Planning, a list of related items such as Profit and Loss items, months, products, customers, and cost centers, including calculations. The rows, columns, and pages of a cube are created from dimensions.

In Cognos BI, TM1, and Express, a broad grouping of descriptive data about a major aspect of a business, such as products, dates, or locations. Each dimension includes different levels of members in one or more hierarchies and an optional set of calculated members or special categories.

governor

A set of rules to limit user activities, such as the execution of reports, that either take too long, or consume too many resources.

hierarchy

The organization of a set of entities into a tree structure, with each entity (except the root) having one or more parent entities and an arbitrary number of child entities.

In Data Manager, a particular view of a business dimension. A hierarchy contains the definition of related reference data that is organized into a tree structure of members related as parents and children.

level

A set of entities or members that form one section of a hierarchy in a dimension and represent the same type of object. For example, a geographical dimension might contain levels for country, region, and city.

locale

A setting that identifies language or geography and determines formatting conventions such as collation, case conversion, character classification, the language of messages, date and time representation, and numeric representation.

measure

A performance indicator that is quantifiable and used to determine how well a business is operating. For example, measures can be Revenue, Revenue/Employee, and Profit Margin percent.

member

In Data Manager, a node in a reference structure.

model

In Data Manager, a system, consisting of fact data and metadata, that represents the aspects of a business.

model segment

A part of a Framework Manager project, such as a parameter map, a data source, a namespace, or a folder, that is a shortcut to a second project. Segments are used to simplify model maintenance or to facilitate multi-user modeling.

namespace

For authentication and access control, a configured instance of an authentication provider that allows access to user and group information. In Framework Manager, namespaces uniquely identify query items, query subjects, and so on. You import different databases into separate namespaces to avoid duplicate names.

In XML and XQuery, a uniform resource identifier (URI) that provides a unique name to associate with the element, attribute, and type definitions in an XML schema or with the names of elements, attributes, types, functions, and errors in XQuery expressions.

normalization

The process of restructuring a data model by reducing its relations to their simplest forms. It is a key step in the task of building a logical relational database design. Normalization helps avoid redundancies and inconsistencies in data. An entity is normalized if it meets a set of constraints for a particular normal form (first normal form, second normal form, and so on).

package

A subset of a model, which can be the whole model, to be made available to the Cognos server.

project

In Framework Manager, a set of models, packages, and related information for administration, and for sharing model information.

In Metric Studio, a task or set of tasks undertaken by a team and monitored on a scorecard. A project tracks the dates, resources, and status of the project.

In Metric Designer, a group of extracts. Each extract contains the metadata that is used to populate the Metric Studio data store or to create applications.

publish

In Cognos BI, to expose all or part of a Framework Manager model or Transformer PowerCube, through a package, to the Cognos server, so that the data can be used to create reports and other content.

In Cognos Planning, to copy the data from Contributor or Analyst to a data store, typically so that the data can be used for reporting purposes.

query item

A representation of a column of data in a data source. Query items may appear in a model or in a report and contain a reference to a database column, a reference to another query item, or a calculation.

query subject

A named collection of query items that are closely functionally related. Query subjects are defined using Framework Manager to represent relational data and form the set of available data for authoring reports in Query Studio and Report Studio. A query subject is similar to a relational view in that it can be treated as a table but does not necessarily reflect the data storage.

Index

A

- accessibility features, [571](#)
 - keyboard shortcuts, [571](#)
- accessing
 - data source connections, [56](#)
 - secured InfoCube, [197](#)
- access permissions
 - definition, [613](#)
- action logs, [295](#)
 - running in batch mode, [297](#)
- adding
 - business rules for relational metadata, [155](#)
 - business rules for SAP BW metadata, [237](#)
 - data source functions, [314](#)
 - function sets, [314](#)
 - groups, [256](#)
 - languages for relational metadata, [134](#)
 - languages to packages, [262](#)
 - metadata security, [261](#)
 - object security, [259](#)
 - roles, [256](#)
 - security, [261](#)
 - users, [256](#)
- additive, [145](#), [223](#)
- administrative access, [261](#)
- advisor, model, [191](#)
- aggregate rollups, [311](#)
- aggregation, [322](#)
 - rules for relational metadata, [141](#), [143](#), [144](#), [145](#)
 - rules for SAP BW metadata, [222](#), [223](#), [224](#)
 - types, [223](#)
 - types for relational metadata, [145](#)
- aggregation for calculations, [334](#)
- aggregation rules, [146](#)
- Aggregation Rules property
 - relational metadata, [141](#)
- aggregation type
 - calculated, [191](#)
- aliases
 - definition, [613](#)
 - using with parameters, [163](#), [243](#)
- ambiguous objects, [360](#)
- ambiguous paths, [181](#)
- ambiguous relationships, [339](#)
- Analysis Studio and sparse data, [81](#)
- analyzing
 - models, [191](#)
 - problems, [21](#)
 - publishing impact on packages, [287](#)
- applying
 - filters for relational metadata, [160](#)
 - filters for SAP BW metadata, [242](#)
- Architect XML files
 - importing, [61](#)
- As View SQL, [112](#)
 - model query subjects, [113](#)
- attributes, [114](#), [122](#), [123](#), [205](#), [210](#), [212](#), [368](#)
 - definition, [613](#)
- auditing, [272](#), [287](#)
- automatic aggregation types
 - relational metadata, [144](#)
 - SAP BW metadata, [222](#), [224](#)
- auto save
 - options, [30](#)

B

- balanced hierarchies, [118](#), [206](#)
- binary round-off errors, [379](#)
- BLOB, [305](#)
- BmtScriptPlayer
 - syntax, [298](#)
- braces in expressions, [377](#)
- branches
 - creating, [276](#)
- branching
 - methodologies, [274](#)
 - projects, [273](#)
- broken reports, [269](#)
- business layer, [77](#)
- business rules for relational metadata, [155](#)
 - calculations, [155](#)

- filters, [158](#)
 - macros, [168](#)
 - parameter maps, [163](#)
 - parameters, [167](#)
 - prompts, [149](#)
 - session parameters, [165](#)
 - business rules for SAP BW metadata, [237](#)
 - calculations, [237](#)
 - filters, [239](#)
 - business view, [77](#)
 - business view for relational metadata, [179](#)
 - folders, [187](#)
 - namespaces, [187](#)
 - shortcuts, [184, 185](#)
 - shortcuts and relationships, [185](#)
 - business view for SAP BW metadata, [246](#)
 - folders, [248](#)
 - namespaces, [248](#)
 - shortcuts, [247](#)
 - shortcuts and dimensions, [248](#)
- C**
- cached data, [304](#)
 - reusing, [313](#)
 - cached metadata, [191](#)
 - calculated aggregations
 - relational metadata, [143](#)
 - calculated aggregation type, [191, 334](#)
 - calculated key figures, [197](#)
 - calculated members
 - definition, [613](#)
 - calculations
 - creating for relational metadata, [155](#)
 - creating for SAP BW metadata, [237](#)
 - order of operations, [334](#)
 - security, [260](#)
 - unexplained number discrepancies, [379](#)
 - cardinality
 - 1-1, [355](#)
 - 1-n, [355](#)
 - checking, [339](#)
 - data source, [79](#)
 - definition, [613](#)
 - dimensions and facts, [339](#)
 - facts, [191](#)
 - mixed, [191](#)
 - notation, [79](#)
 - queries, [80, 320](#)
 - redefining, [82](#)
 - rules, [80, 320](#)
 - types, [80, 320](#)
 - catalog, [317](#)
 - categories
 - for verifying, [251](#)
 - changed features, [13, 18](#)
 - changes in model not in report, [270](#)
 - changing
 - metadata, [284](#)
 - type of SQL, [107](#)
 - changing packages
 - analyzing the effects, [287](#)
 - characteristic
 - mapping to Framework Manager, [203](#)
 - characteristic dimensions
 - externalizing, [603](#)
 - characteristic structures, [201](#)
 - checking
 - projects, [251](#)
 - relationships, [78](#)
 - choosing
 - query processing, [312](#)
 - clearing
 - object security, [259](#)
 - CLOB
 - Oracle, [375](#)
 - Cognos PowerCubes, *See* [IBM Cognos PowerCubes](#)
 - Cognos SQL, [109](#)
 - comments
 - adding to SQL, [106](#)
 - Common Warehouse Metamodel, [291](#)
 - exporting, [378](#)
 - comparing tables, [97](#)
 - complex expressions
 - relationships, [82](#)
 - Composite stored procedures, [89](#)
 - concepts, [319](#)
 - conditional query subjects, [163, 243](#)
 - conformed dimension
 - definition, [613](#)
 - conformed dimensions, [348](#)
 - creating, [97](#)
 - multiple facts, [352, 357](#)

- SAP BW, [15, 17](#)
 - conformed star schema groups, [348](#)
 - connecting to multiple PowerCubes, [53](#)
 - connections between
 - dimensions, [127, 213](#)
 - query subjects, [97](#)
 - content manager data source, [317](#)
 - Context Explorer, [97, 127, 213](#)
 - controlling
 - access, [256](#)
 - SQL generation, [304](#)
 - converting
 - measures into query items, [124, 125](#)
 - model query subjects into data source query subjects, [106](#)
 - query items into measures, [154](#)
 - query subjects into dimensions, [105](#)
 - regular dimensions, [130](#)
 - copying
 - projects, [285](#)
 - creating, [291](#)
 - branches, [276](#)
 - calculations for relational metadata, [155](#)
 - calculations for SAP BW metadata, [237](#)
 - data source connections, [56](#)
 - dimensions for relational metadata, [114, 122, 123](#)
 - dimensions for SAP BW metadata, [205](#)
 - filters for relational metadata, [158](#)
 - filters for SAP BW metadata, [239](#)
 - folders for relational metadata, [187](#)
 - folders for SAP BW metadata, [248](#)
 - links, [283](#)
 - measure dimensions, [347](#)
 - measure dimensions for relational metadata, [124](#)
 - measure folders for relational metadata, [188](#)
 - namespaces, [76](#)
 - namespaces for relational metadata, [187](#)
 - namespaces for SAP BW metadata, [248](#)
 - packages, [254](#)
 - parameter maps for relational metadata, [163](#)
 - parameter maps for SAP BW metadata, [243](#)
 - projects, [26, 47](#)
 - prompts with query macros, [171](#)
 - query item folders for relational metadata, [188](#)
 - query sets for relational metadata, [97](#)
 - query subjects for relational metadata, [86, 87, 90](#)
 - query subjects for SAP BW metadata, [217](#)
 - regular dimensions, [345](#)
 - regular dimensions for relational metadata, [115, 127](#)
 - relationships for relational metadata, [83](#)
 - relationship shortcuts for relational metadata, [83](#)
 - segments, [282](#)
 - star schema groups, [348](#)
 - cross-fact queries, [339](#)
 - cross-product joins, [304](#)
 - crowsfeet notation, [33, 47](#)
 - csv files, [263, 266](#)
 - CSVIdentityName function, [258](#)
 - CSVIdentityNameList function, [259](#)
 - cubes, [317](#)
 - definition, [613](#)
 - curly brackets, [168](#)
 - currency
 - format type, [148, 225](#)
 - custom properties
 - SAP BW, [230](#)
 - custom relationship expressions, [82](#)
 - CWM, *See* [Common Warehouse Metamodel](#)
- ## D
- data
 - formatting for relational metadata, [148](#)
 - formatting for SAP BW metadata, [225](#)
 - security, [256, 257](#)
 - using multilingual, [131](#)
 - database connections, *See* [data source connections](#)
 - database functions
 - vendor-specific, [314](#)
 - database layer, [77](#)
 - database only, [312](#)
 - databases
 - importing, [59](#)
 - data extraction
 - SAP BW guidelines for variable usage, [595](#)
 - data formats
 - date and time symbols, [584](#)
 - decimal format symbols, [591](#)
 - using patterns, [583](#)
 - data source
 - adding functions, [314](#)
 - updated by stored procedure, [90](#)

- data source connections, [56](#)
 - accessing, [56](#)
 - creating, [56](#)
 - isolation levels, [54](#)
 - Microsoft SQL, [56](#)
 - OLAP cubes, [52](#)
 - testing, [58](#)
- data source functions
 - adding, [314](#)
- data source query subjects
 - creating, [86](#)
 - definition, [25](#)
- data source query subjects for relational metadata, [85](#)
 - determinants, [92, 94](#)
 - using parameters, [167](#)
- data sources
 - Architect XML files, [61](#)
 - connection levels, [54](#)
 - DecisionStream, [62](#)
 - definition, [613](#)
 - ERwin, [69](#)
 - IBM Cognos models, [61](#)
 - IBM DB2 Cube Views, [69](#)
 - Impromptu XML files, [61](#)
 - Microsoft Analysis Server, [60](#)
 - Microsoft SQL Server, [60](#)
 - modifying properties, [317](#)
 - multilingual, [133](#)
 - Oracle Designer, [74](#)
 - other metadata sources, [69](#)
 - paths to file-based data source connections, [57](#)
 - properties, [209, 312](#)
 - relational databases, [59](#)
 - SAP BW, [15, 17, 197](#)
 - XML files, [75](#)
- datatypes
 - graphic, [376](#)
- data types, [366](#)
- data warehouse
 - relationships, [78](#)
- date
 - format type, [148, 225](#)
 - prompts, [151, 227](#)
- date prompts, [149, 226](#)
- DB2, [377](#)
- DB2 Cube Views
 - imported expressions, [74](#)
- DecisionStream
 - importing, [62](#)
- default data set
 - setting, [293](#)
- defining
 - function sets, [314](#)
 - languages for relational metadata, [134](#)
 - prompt controls, [149](#)
 - prompt controls for SAP BW metadata, [226](#)
- deleting
 - projects, [287](#)
- dense data
 - semi-additive measures, [146](#)
- dependencies, object, [289](#)
- dependencies, report, [288](#)
- deprecated features, [13, 18](#)
- detailed fact query subject, [593](#)
- detailed key figures, [593](#)
- detecting
 - relationships, [84](#)
- determinants, [92](#)
 - cardinality, [79](#)
 - converting from dimension information, [368](#)
 - defining, [322](#)
 - query subjects, [332](#)
 - relationships, [191](#)
 - specifying, [92, 94](#)
 - SQL generation, [96](#)
 - uniquely identified, [92, 94](#)
- diagrams, [28](#)
 - settings, [33, 47](#)
 - viewing, [32, 44, 45](#)
- Diagram tab, [32](#)
- dimensional data, [343](#)
- dimensionally modeled relational metadata
 - semi-additive measures, [146](#)
- dimensionally modeling relational metadata, [345](#)
- dimensional models
 - star schemas, [180](#)
- dimensional queries, [351](#)
 - multiple facts and grains, [352, 357](#)
 - single fact, [351](#)
- dimension information, [368](#)
- dimension map, [114, 122, 123, 205](#)

- Dimension Map tab, [34](#)
- dimensions
 - ambiguous, [360](#)
 - converting from query subjects, [368](#)
 - creating, [34](#)
 - definition, [25](#), [614](#)
 - exploring, [32](#), [44](#)
 - hierarchies, [346](#)
 - identifying, [339](#)
 - measure, [336](#), [347](#)
 - mixed cardinality, [191](#)
 - model, [326](#)
 - modifying, [34](#)
 - properties, [34](#)
 - query subjects, [326](#)
 - regular, [326](#), [332](#), [336](#), [345](#)
 - role-playing, [339](#)
 - SAP BW, [593](#)
 - searching, [36](#)
 - semi-additive measures, [146](#)
 - shared, [336](#)
 - star schema groups, [348](#)
- dimensions for relational metadata, [114](#), [180](#)
 - balanced hierarchies, [118](#)
 - converting, [130](#)
 - converting from query subjects, [105](#)
 - dimension map, [114](#), [122](#), [123](#)
 - exploring, [127](#)
 - hierarchies, [117](#)
 - keys, [120](#)
 - levels, [117](#), [119](#)
 - measure, [124](#)
 - merging, [127](#)
 - modifying, [34](#)
 - network hierarchies, [119](#)
 - ragged hierarchies, [119](#)
 - regular, [115](#)
 - roles, [122](#), [123](#)
 - scope relationships, [126](#)
 - shortcuts, [185](#)
 - snowflaked, [183](#)
 - testing, [128](#)
 - unbalanced hierarchies, [119](#)
- dimensions for SAP BW metadata, [204](#)
 - balanced hierarchies, [206](#)
 - dimension map, [205](#)
 - hierarchies, [205](#)
 - key figures, [212](#)
 - levels, [205](#), [209](#)
 - network hierarchies, [209](#)
 - ragged hierarchies, [208](#)
 - regular, [205](#)
 - roles, [210](#), [212](#)
 - shortcuts, [248](#)
 - testing, [213](#)
 - unbalanced hierarchies, [207](#)
- DMR (dimensionally modeled relational) metadata, [345](#)
- documenting the model, [272](#)
- double-counting, [92](#), [339](#), [355](#), [360](#)
- double quotation marks, [168](#)
- drop-down list prompts, [149](#), [151](#), [226](#), [227](#)
- duplicate object names
 - importing, [59](#)
- durable models, [13](#), [189](#)
 - renaming query items, [287](#)
- dynamic query mode, [13](#), [266](#), [268](#), [305](#), [306](#), [308](#)
- E**
- editing
 - SQL, [106](#)
- embedded files, [263](#), [266](#)
- embedded strings, [168](#)
 - in expressions, [168](#)
 - in macros, [168](#)
- environment
 - Framework Manager, [23](#)
- error messages
 - QE-DEF-0177, [376](#)
 - QE-DEF-0259, [377](#)
 - UDA-SQL-0107, [376](#)
 - UDA-SQL-0114, [376](#)
- errors
 - repairing, [252](#)
- ERWin metadata
 - out of memory error, [375](#)
- escape symbol, [168](#)
- evaluating
 - relational query subjects, [104](#)
 - SAP BW query subjects, [218](#)
- Event Studio
 - stored procedures, [90](#)

- examples for relational metadata
 - filter, [162](#)
 - in_range function, [158](#)
 - multilingual modeling, [136](#)
 - parameter map, [165](#)
 - prompts, [174](#)
- examples for SAP BW metadata
 - in_range function, [239](#)
- except all query set, [97](#)
- except query set, [97](#)
- Explorer Diagram, [44](#)
- Explorer tab, [32](#)
- Explorer Tree, [43](#)
- exploring
 - dimensions, [114](#), [127](#), [205](#), [213](#)
 - object-based security, [271](#)
 - objects, [32](#), [44](#)
 - packages, [271](#)
 - query subjects, [97](#)
 - roles, [271](#)
- exporting
 - CWM file, [378](#)
 - metadata as CWM files, [291](#)
 - translation tables, [135](#)
- expression editor
 - function sets, [314](#)
 - searching for values, [382](#)
- expressions
 - braces, [377](#)
 - custom, [82](#)
 - imported, [74](#)
 - relationships, [82](#)
 - using prompts, [149](#)
- externalized dimensions
 - prompts, [378](#)
- externalize methods
 - IQD, [19](#)
- externalizing
 - characteristic dimensions, [603](#)
 - Framework Manager requirements, [605](#)
 - InfoCube key figures, [603](#)
 - measure dimensions, [603](#)
 - model query subjects, [602](#)
 - query subjects, [263](#), [266](#)
 - SAP BW Dimensions, [593](#)
 - SAP BW Query key figures, [603](#), [605](#)

- SAP BW Query requirements, [603](#)

F

- fact data, [344](#)
- fact-less query, [348](#)
- facts, [347](#)
 - aggregation rules, [146](#)
 - ambiguous, [360](#)
 - cardinality, [191](#)
 - identifying, [339](#)
 - mixed cardinality, [191](#)
- fact tables, [124](#), [180](#)
- features
 - changed, [13](#), [18](#)
 - deprecated, [13](#), [18](#)
 - new, [13](#), [17](#)
 - removed, [13](#), [19](#)
- file-based data sources
 - paths, [57](#)
- filters
 - mandatory, [160](#)
 - optional, [160](#)
 - security, [257](#), [260](#)
- filters for relational metadata
 - applying to query subjects, [160](#)
 - creating, [158](#)
 - determining usage, [160](#)
 - mandatory, [158](#)
 - modifying, [158](#)
 - multiple dimensions, [34](#)
 - multiple query items, [34](#)
 - multiple query subjects, [34](#)
 - optional, [158](#)
- filters for SAP BW metadata
 - applying, [242](#)
 - creating, [239](#)
 - determining usage, [242](#)
 - modifying, [239](#)
- finding
 - objects, [36](#)
- fixing
 - failed transactions, [279](#)
- folders
 - creating, [187](#), [188](#)
 - measure, [188](#)
 - query item, [188](#)

- using shortcuts, [184](#)
- folders for SAP BW metadata
 - creating, [248](#)
 - using shortcuts, [247](#)
- font
 - settings, [33, 47](#)
- foreign keys, [84](#)
- formatting
 - data for relational metadata, [148](#)
 - data for SAP BW metadata, [225](#)
- format types
 - currency, [148, 225](#)
 - date, [148, 225](#)
 - text, [148, 225](#)
- Framework Manager
 - project, [23](#)
 - security filters change for SAP BW, [19](#)
- full outer joins, [79, 376](#)
 - Oracle, [379](#)
- functions
 - project, [314](#)
- function set ID, [317](#)
- function sets
 - adding, [254, 314](#)
 - defining, [314](#)

G

- gateways
 - cannot access URI, [375](#)
- generating
 - models, using Model Design Accelerator, [49](#)
 - relationships, [84](#)
- governor
 - Allow Dynamic Generation of Dimensional Information, [310](#)
 - Allow Enhanced Model Portability at RunTime, [309](#)
 - Allow Usage of Local Cache, [310](#)
 - Cross-Product Joins, [306](#)
 - Grouping of Measure Attributes, [307](#)
 - Maximum External Data File Size, [311](#)
 - Maximum External Data Row Count, [311](#)
 - Maximum External Data Sources, [311](#)
 - Maximum Number of Report Table, [305](#)
 - Maximum Number of Retrieved Rows, [305](#)
 - Outer Joins, [305](#)
 - Publish Entire Model When Processing, [310](#)

- Query Execution Time Limit, [305](#)
- Shortcut Processing, [306](#)
- SQL Generation for Determinant Attributes, [308](#)
- SQL Generation for Level Attributes, [308](#)
- SQL Join Syntax, [307](#)
- SQL Parameter Syntax, [309](#)
- Suppress Null Values for SAP BW Data Sources, [310](#)
- Use With Clause When Generating SQL, [310](#)

- governors
 - allow usage of local cache, [313](#)
 - definition, [614](#)
 - IBM Cognos ReportNet upgrade, [366](#)
 - setting, [304](#)
- granularity, [342](#)
- graphic datatype, [376](#)
- Grouping of Measure Attributes
 - governor, [307](#)
- groups
 - adding, [256](#)
 - star schema, [180](#)

H

- hierarchies, [322, 326, 368](#)
 - balanced, [118, 206](#)
 - definition, [614](#)
 - multiple, [346](#)
 - network, [119, 209](#)
 - ragged, [119, 208](#)
 - regular dimension, [117, 205](#)
 - SAP BW, [199](#)
 - specifying roles, [122, 123](#)
 - time-dependent, [14, 200](#)
 - unbalanced, [119, 207](#)
 - versioned, [200](#)

I

- IBM accessibility page, [573](#)
- IBM Cognos 8 Planning and SAP BW data, [593](#)
- IBM Cognos models
 - importing, [61](#)
- IBM Cognos PowerCubes
 - publishing, [268](#)
- identifiers
 - unique, [92](#)
- imported metadata
 - checking, [339](#)

importing

- Architect XML files, [61](#)
- databases, [59](#)
- data sources, [69](#)
- DecisionStream, [62](#)
- duplicate object names, [59](#)
- IBM Cognos models, [61](#)
- Impromptu XML files, [61](#)
- metadata from XML files, [75](#)
- metadata into Framework Manager, [58](#)
- relational metadata, [84](#)
- relationships, [78](#)
- SAP BW hierarchies, [199](#)
- SAP BW metadata, [197](#), [202](#)
- translation tables for relational metadata, [136](#)

import view, [77](#)

Impromptu Query Definition files, [263](#), [266](#)

Impromptu XML files

- importing, [61](#)

improving performance

- reusing cached data, [313](#)

in_range function, [158](#), [239](#)

InfoCube

- mapping to Framework Manager, [203](#)
- permissions for accessing metadata, [198](#)

InfoCube key figures

- externalizing, [603](#)

Informix

- functions, [89](#)
- stored procedures, [89](#)

inner joins, [79](#), [84](#), [378](#)

intersect all query set, [97](#)

intersect query set, [97](#)

IQD

- files, [263](#), [266](#)

IQD externalize method, [19](#)

isolation levels, [54](#)

J

Japanese characters

- export paths, [378](#)

joins, [78](#)

- creating, [45](#)
- deleting, [45](#)
- full outer, [79](#), [376](#)
- inner, [79](#), [84](#), [378](#)

loop, [341](#)

modifying, [45](#), [82](#)

Oracle, [379](#)

outer, [79](#), [81](#)

overriding, [45](#)

K

keyboard shortcuts, [571](#)

key figures, [197](#)

mapping to Framework Manager, [203](#)

modifying, [212](#)

prompts, [378](#)

structures, [201](#)

keys, [114](#), [205](#), [368](#)

cardinality, [79](#)

foreign, [84](#)

for levels, [120](#)

for roles, [210](#), [212](#)

primary, [84](#)

specify roles, [122](#), [123](#)

L

languages

adding for relational metadata, [134](#)

defining for relational metadata, [134](#)

incorrect in SAP BW query, [376](#)

setting up multilingual environment, [131](#)

specifying for packages, [262](#)

Large Text Item Limit, [305](#)

layers, [77](#)

level of details

- settings, [33](#), [47](#)

levels, [114](#), [117](#), [205](#), [210](#), [212](#), [368](#)

definition, [614](#)

member unique names, [120](#), [210](#)

regular dimension, [119](#), [209](#)

security, [378](#)

sorting, [121](#)

specify roles, [122](#), [123](#)

unique, [115](#)

limitations

linking, [281](#)

macros, [171](#)

segmenting, [281](#)

limited local, [312](#)

- linked projects, [280, 372](#)
 - synchronizing, [303](#)
- linking
 - limitations, [281](#)
- links, [280](#)
 - creating, [283](#)
- literal strings, [168](#)
 - in expressions, [168](#)
 - in macros, [168](#)
- locales
 - definition, [614](#)
- locales, *See Also* [languages](#)
- local query processing
 - rollup processing, [311](#)
- log file
 - archive entries, [301](#)
- log files, [295](#)
 - play back, [296](#)
- logging on
 - to Microsoft SQL, [56](#)
- loop joins, [339, 341](#)
- M**
- macros
 - creating, [171](#)
 - limitations, [171](#)
 - prompts, [168](#)
 - using, [168](#)
- main projects, [282](#)
- managing
 - packages, [259, 261, 271, 272](#)
 - project, [271](#)
 - projects, [285, 295](#)
- mandatory cardinality, [80](#)
- mandatory prompts, [171](#)
- many-to-many relationships, [79](#)
- many-to-one relationships, [78](#)
- mapping
 - parameter values for relational metadata, [163](#)
 - parameter values for SAP BW metadata, [243](#)
 - SAP BW objects to Framework Manager, [203](#)
- master-detail tables, [344, 347](#)
- maximum cardinality, [80, 320](#)
- measure dimensions, [336](#)
 - creating, [347](#)
 - externalizing from SAP BW, [603](#)
 - role-playing, [339](#)
- measure dimensions for relational metadata
 - creating, [124](#)
 - exploring, [127](#)
 - scope relationships, [126](#)
- measure dimensions for SAP BW metadata
 - exploring, [213](#)
- measure folders, [188](#)
- measures
 - calculated aggregation type, [191](#)
 - converting, [125, 154](#)
 - converting to query items, [124](#)
 - definition, [614](#)
 - for IBM Cognos 8 Planning, [593](#)
 - IBM Cognos BI upgrade, [367](#)
 - semi-additive, [146](#)
- measure scope issues, [126](#)
- members
 - definition, [614](#)
 - sorting, [121](#)
- member unique names
 - relational metadata, [120](#)
 - SAP BW metadata, [210](#)
- merging
 - projects, [273](#)
 - query subjects, relational, [96](#)
 - query subjects, SAP BW, [217](#)
 - regular dimensions, relational, [127](#)
- merise notation, [33, 47](#)
- metadata
 - cached, [191](#)
 - changing, [284](#)
 - exporting, [291](#)
 - importing, [58](#)
 - merging, [96, 217](#)
 - moving, [284](#)
 - publishing, [251](#)
 - relational, [77](#)
 - SAP BW, [197](#)
 - security, [256, 261](#)
 - synchronizing, [301](#)
 - understanding, [271](#)
- Metadata Wizard, [47](#)
- methodologies
 - branching, [274](#)

Microsoft

- SQL server and logon, [56](#)
- SQL Server data sources, [60](#)

Microsoft Analysis Server

- data sources, [60](#)

migrating

- models, [294](#)

Minimized SQL, [112](#), [191](#)

- model query subjects, [113](#)

minimum cardinality, [80](#), [320](#)

model

- durability, [189](#)
- versioning, [270](#)

Model Accelerator, [45](#)model advisor, [191](#)model business views, [179](#), [246](#)Model Design Accelerator, [43](#)

- building query subjects, [49](#)
- creating projects, [47](#)
- creating star schemas, [48](#)
- diagram colors, [44](#)
- diagram settings, [47](#)
- Explorer Diagram, [44](#)
- Explorer Tree, [43](#)
- generating models, [49](#)
- joins, [45](#)
- managing star schemas, [50](#)
- Metadata Wizard, [47](#)
- Model Accelerator, [45](#)
- Model Warning View, [46](#)
- Query Subject Diagram, [45](#)
- Relationship Editing Mode, [45](#)

model documentation, [272](#)

modeling

- data sources, [133](#)
- distributed models, [273](#)
- main projects, [273](#)
- multilingual, [133](#)
- multiuser, [273](#)

modeling problems, [191](#)model objects, [28](#)

- shortcuts, [333](#)
- using parameters, [163](#), [243](#)
- viewing, [28](#), [43](#)

model portability, [294](#), [304](#)

model query subjects

- definition, [25](#)
- externalizing, [602](#)
- security, [260](#)

model query subjects for relational metadata, [87](#)

- creating, [87](#)
- creating from existing objects, [96](#)
- determinants, [92](#), [94](#)
- SQL type, [113](#)

model query subjects for SAP BW metadata, [216](#)

- creating, [217](#)
- creating from existing objects, [217](#)

models, [23](#)

- analyzing, [191](#)
- definition, [24](#), [614](#)
- generating, using Model Design Accelerator, [49](#)
- migrating from one relational database to another, [294](#)
- preparing relational, [77](#)
- preparing SAP BW metadata, [197](#)
- publishing, [251](#)
- sample, [39](#)
- upgrading, [27](#), [365](#)
- verifying, [251](#)
- versioning, [266](#)

model segments

- definition, [614](#)

model session parameters

- relational metadata, [165](#)
- SAP BW metadata, [245](#)

model versioning, [270](#)

model versions

- updating, [266](#)

Model Warning View, [46](#)

modifying

- filters for relational metadata, [158](#)
- filters for SAP BW metadata, [239](#)
- package access, [261](#)
- package administrative access, [261](#)
- packages, [254](#)
- properties, [34](#)
- query item properties, [34](#)
- query items for relational metadata, [138](#)
- query items for SAP BW metadata, [219](#)
- relational dimensions, [34](#)
- relational query subjects, [34](#)

- relationships, 82
- session parameters, 165, 245
- SQL at runtime, 171
- stored procedure query subjects, 90
- moving
 - metadata, 284
 - projects, 286
- MultiCube
 - mapping to Framework Manager, 203
- multidimensional
 - query subjects, 216
- multi-edge suppression, 263
- multilingual
 - data, using, 131
 - data sources, 133
- multilingual modeling
 - example for relational metadata, 136
 - packages, 262
 - parameters, 134
 - relational metadata, 131
- multilingual reporting environment
 - setting up, 131
- multiple cubes in packages, 268
- multiple data sets, 292
 - setting default data set, 293
- multiple data source connections
 - using, 53, 56
- multiple-fact queries, 326, 352, 357
- multiple-grain queries, 326, 352, 357
- multiple hierarchies, 346
- multiple relationships, 191
- multiple users
 - modeling, 273
- multiple valid relationships, 339, 341
- multiuser modeling, 273

N

- namespaces, 23
 - creating, 76
 - creating for relational metadata, 187
 - creating for SAP BW metadata, 248
 - definition, 25, 615
- naming conventions
 - objects, 38
- native metadata, 52
- native SQL, 110

- nested parameters, 163, 243
- network hierarchies, 119, 209
- new features, 13, 17
- non-additive, 145, 223
- normalization
 - definition, 615
- normalized data sources, 343
- notation, 33, 47, 78, 79
- null suppression, 263
- null values
 - SAP BW, 304
- number sign
 - reserved for macros, 168

O

- objects
 - exploring, 32, 44
 - naming conventions, 38
 - properties, 34
 - remapping, 290
 - reorder, 31
 - searching for, 36
 - unique identifiers, 38
- object security, 256
 - adding, 259
 - calculations, 260
 - exploring, 271
 - filters, 260
 - model query subjects, 260
 - removing, 259
 - shortcuts, 260
- OLAP cubes
 - as datasources, 52
- OLAP data sources
 - publishing, 268
- one-to-many relationships, 78, 355
- one-to-one relationships, 78, 355
- opening
 - projects, 27
- operations for calculations, 334
- optional cardinality, 80, 320
- optional prompts, 171
- optional relationships, 79
- Oracle
 - Designer, 74
 - full outer joins, 379

- Oracle full outer joins, 376
- order of operations for calculations, 334
- organizing models, 179, 246
- outer joins, 79, 84, 304
 - Analysis Studio, 81
 - full, 79, 376
 - Oracle, 379
- out of memory error, 375
- P**
- package administrative access
 - modifying, 261
- packages, 23
 - access, 261
 - adding security, 261
 - administrative access, 261
 - analyzing changes, 287
 - containing multiple cubes, 268
 - creating, 254
 - creating in Cognos Connection, 197, 266
 - definition, 25, 615
 - exploring, 271
 - managing, 259, 261, 271, 272
 - modifying, 254
 - multilingual support, 262
 - publishing, 251, 266
 - SAP BW data sources, 14
 - security, 256
 - verifying, 251
 - viewing inclusion, 272
- parameter maps for relational metadata
 - creating, 163
 - using as lookup tables, 131
- parameter maps for SAP BW metadata, 243
 - creating, 243
- parameters
 - in data source query subjects, 167
- parameters for relational metadata
 - aliases, 163
 - creating, 163
 - data source query subjects, 167
 - model objects, 163
 - multilingual modeling, 134
 - nested, 163
 - session, 165
 - using, 168
- parameters for SAP BW metadata
 - creating, 243
 - maps, 243
 - session, 245
- pass-through SQL, 111
- performance, optimizing for SAP BW, 233, 234
- performance tuning
 - query processing, 312
- physical layer, 77
- play back
 - action logs, 297
- plus operator, 168
- portability, 294
- PowerCubes
 - building from SAP BW data, 606
 - connecting to multiple data sources, 53
 - Framework Manager guidelines, 606
 - paths to cube files, 57
 - Transformer guidelines, 607
- presentation hierarchy level
 - mapping to Framework Manager, 203
- presentation layer, 77
- primary keys, 84
- project
 - managing, 271
- project objects
 - naming conventions, 38
- project page, 28
- projects, 23
 - branching, 273
 - copying, 285
 - creating, 26, 47
 - definition, 23, 615
 - deleting, 287
 - files, 23
 - fixing synchronization errors, 300
 - Framework Manager, 23
 - functions, 314
 - linking, 283
 - managing, 285, 295
 - merging, 273
 - moving, 286
 - opening, 27
 - options, 30
 - play back transactions, 296
 - renaming, 286

- segmenting, [282](#)
- shortcuts, [283](#)
- synchronizing, [301](#)
- view transaction history, [296](#)
- Project Viewer, [28](#)
- Prompt Info property, [149, 226](#)
- prompt information properties, [149](#)
- prompts
 - cascade on reference, [152, 228](#)
 - creating, [171](#)
 - creating with macros, [171](#)
 - date, [149, 151, 226, 227](#)
 - defining, [149, 226](#)
 - different values to use and display, [152, 228](#)
 - display item reference, [152, 228](#)
 - drop-down list, [149, 151, 226, 227](#)
 - examples, [174](#)
 - filter on reference, [153, 229](#)
 - in expressions, [149](#)
 - key figures, [378](#)
 - mandatory, [171](#)
 - optional, [171](#)
 - properties, [149, 151, 226, 227](#)
 - SAP BW variables, [230](#)
 - search, [149, 151, 226, 227](#)
 - stored procedure example, [92](#)
 - testing, [154, 230](#)
 - text box, [151, 226, 227](#)
 - time, [149, 151, 226, 227](#)
 - type-in, [149, 151, 226, 227](#)
 - use item reference, [152, 228](#)
- properties
 - aggregation rules, [141](#)
 - aggregation rules for SAP BW metadata, [222](#)
 - cascade on reference, [152, 228](#)
 - catalog, [317](#)
 - content manager data source, [317](#)
 - cubes, [317](#)
 - data sources, [312](#)
 - display item reference, [152, 228](#)
 - filter on reference, [153, 229](#)
 - function set ID, [317](#)
 - modifying, [34](#)
 - parameter map, [317](#)
 - prompt info, [149, 226](#)
 - prompt type, [151, 227](#)

- query interface, [317](#)
- query processing, [317](#)
- query type, [317](#)
- regular aggregate, [141, 222](#)
- rollup processing, [317](#)
- SAP BW data source, [209](#)
- SAP BW variables, [209](#)
- schema, [317](#)
- semi-aggregate, [222](#)
- usage, [141, 222](#)
- use item reference, [152, 228](#)
- viewing, [34](#)
- publishing
 - definition, [615](#)
 - metadata, [251](#)
 - packages, [251, 266, 269](#)
 - packages based on OLAP data sources, [268](#)

Q

QE-DEF-0177, [376](#)

QE-DEF-0259, [377](#)

queries

- fact-less, [348](#)
- multiple-fact, [326, 352, 357](#)
- multiple-grain, [326](#)
- single fact, [351](#)
- split, [361](#)
- stitched, [339](#)

Query, SAP BW

- mapping to Framework Manager, [203](#)

query items

- converting, [125, 154](#)
- converting from measures, [124](#)
- definition, [26, 615](#)
- properties, [34](#)

query items for relational metadata

- aggregation rules, [145](#)
- durable model capability, [287](#)
- folders, [188](#)
- modifying, [34, 138](#)
- renaming, [287](#)
- roles, [122, 123](#)
- usage, [144](#)

query items for SAP BW metadata

- aggregation, [223](#)
- aggregation rules, [223](#)

- formatting, [225](#)
 - modifying, [219](#)
 - roles, [210](#), [212](#)
 - usage, [223](#)
 - query macros
 - using to create prompts, [171](#)
 - query mode
 - dynamic, [13](#)
 - query processing, [312](#), [317](#)
 - choosing, [312](#)
 - query reuse, [313](#)
 - query sets for relational metadata
 - creating, [97](#)
 - testing, [101](#)
 - Query Subject Diagram, [45](#)
 - query subjects
 - building using Model Design Accelerator, [49](#)
 - converting to dimensions, [368](#)
 - csv files, [263](#), [266](#)
 - DB2, [377](#)
 - definition, [25](#), [615](#)
 - determinants, [191](#), [332](#), [368](#)
 - dimensions, [191](#), [326](#)
 - embedded files, [263](#), [266](#)
 - exploring, [32](#), [44](#)
 - externalized, [263](#), [266](#)
 - facts, [191](#)
 - Impromptu Query Definition files, [263](#), [266](#)
 - minimized SQL, [191](#)
 - mixed cardinality, [191](#)
 - multiple relationships, [191](#)
 - properties, [34](#)
 - SAP BW, [367](#)
 - searching, [36](#)
 - security, [257](#)
 - star schema groups, [348](#)
 - tab files, [263](#), [266](#)
 - testing, [74](#)
 - Transformer files, [263](#), [266](#)
 - using in Transformer, [263](#)
 - query subjects for relational metadata
 - changing type of SQL, [107](#)
 - converting, [105](#), [106](#)
 - converting from dimensions, [130](#)
 - creating, [86](#), [87](#), [96](#)
 - data source, [85](#)
 - determinants, [92](#), [94](#)
 - editing SQL, [106](#)
 - evaluating, [104](#)
 - exploring, [97](#)
 - generating SQL, [112](#)
 - model, [87](#)
 - modifying, [34](#), [90](#)
 - parameter maps, [163](#)
 - query sets, [97](#)
 - relationships, [78](#)
 - shortcuts, [185](#)
 - SQL, [85](#)
 - stored procedure, [88](#)
 - testing, [101](#)
 - types, [85](#)
 - updating, [104](#)
 - query subjects for SAP BW metadata
 - creating, [217](#)
 - evaluating, [218](#)
 - model, [216](#)
 - parameter maps, [243](#)
 - query type, [317](#)
 - quick tours, [11](#)
 - quotation marks, [168](#)
 - in expressions, [168](#)
 - in macros, [168](#)
- ## R
- ragged hierarchies, [119](#), [208](#)
 - recording transactions, [295](#)
 - recursive relationships, [191](#), [342](#)
 - reflexive relationships, [191](#), [342](#)
 - regional settings
 - setting up a multilingual environment, [131](#)
 - Regular Aggregate property, [145](#)
 - relational metadata, [141](#)
 - SAP BW metadata, [222](#), [223](#)
 - regular dimensions, [326](#), [332](#), [336](#)
 - creating, [345](#)
 - hierarchies, [346](#)
 - role-playing, [339](#)
 - regular dimensions for relational metadata, [115](#)
 - converting, [130](#)
 - exploring, [127](#)
 - hierarchies, [117](#)
 - levels, [117](#), [119](#)

- merging, [127](#)
- scope relationships, [126](#)
- regular dimensions for SAP BW metadata, [205](#)
 - exploring, [213](#)
 - hierarchies, [205](#)
 - levels, [205](#), [209](#)
- related objects, [97](#), [127](#), [213](#)
- relational data source query subjects
 - evaluating, [104](#)
 - updating, [104](#)
- relational metadata, [77](#)
 - business rules, [155](#)
 - business view, [77](#)
 - import view, [77](#)
 - query items, [138](#)
 - star schemas, [180](#)
- relational modeling concepts, [320](#)
- relational models
 - migrating, [294](#)
- Relationship Editing Mode, [45](#)
- relationships
 - 0-1, [78](#)
 - 0-n, [78](#)
 - 1-1, [78](#)
 - 1-n, [78](#), [355](#)
 - ambiguous, [339](#)
 - cardinality, [78](#), [320](#)
 - changing, [82](#)
 - checking, [78](#), [339](#)
 - complex expressions, [82](#)
 - creating, [83](#)
 - custom, [82](#)
 - definition, [78](#)
 - detecting, [84](#)
 - determinants, [191](#)
 - from Oracle Designer, [74](#)
 - generating, [84](#)
 - importing, [78](#)
 - levels of granularity, [342](#)
 - many-to-many, [78](#), [79](#)
 - many-to-one, [78](#)
 - modifying, [82](#)
 - multiple, [191](#)
 - multiple valid, [339](#), [341](#)
 - n-n, [78](#)
 - one-to-many, [78](#)
 - one-to-one, [78](#)
 - reflexive and recursive, [191](#)
 - scope, [126](#)
 - shortcuts, [83](#), [185](#)
 - verifying, [78](#)
- remap objects, [290](#)
- RemoteCube
 - mapping to Framework Manager, [203](#)
- removed features, [13](#), [19](#)
- removing
 - object security, [259](#)
- renaming
 - projects, [286](#)
 - relationships, [82](#)
- reorder objects, [31](#)
- repairing
 - IBM Cognos ReportNet models, [367](#)
- repairing models, [252](#)
- report
 - dependencies, [288](#)
- reporting requirements, [21](#)
- reports
 - not showing changes to model, [270](#)
- reserved symbols
 - number sign, [168](#)
- reserved words, [609](#)
- resetting
 - Regular Aggregate property, [148](#)
 - Usage property, [148](#)
- resolving
 - ambiguous objects, [360](#)
 - split queries, [361](#)
- restricting
 - access, [256](#)
 - BLOBs, [305](#)
 - data retrieved, [304](#)
- reusing cached data, [313](#)
- role-playing dimensions, [191](#), [339](#)
- roles
 - adding, [256](#)
 - dimensions, [122](#), [123](#)
 - exploring, [271](#)
- roles for SAP BW metadata
 - dimensions, [210](#), [212](#)
- rollup processing, [311](#), [317](#)
- round-off errors, [379](#)

row limits, [304](#)

rules

aggregation for SAP BW metadata, [222](#)

business, [155](#)

business for SAP BW metadata, [237](#)

cardinality, [80](#)

rules of cardinality, [320](#)

S

sample models, [39](#)

SAP BW

conformed dimensions, [15, 17](#)

custom properties, [230](#)

guidelines for using fact data, [595](#)

importing key figures, [197](#)

security filter change in Framework Manager, [19](#)
Transformer, [602](#)

wrong language in query, [376](#)

SAP BW data

Cognos Planning, [593](#)

creating packages, [14](#)

SAP BW dimensions, [593](#)

SAP BW metadata, [197](#)

importing, [197](#)

SAP BW Query

permissions for accessing metadata, [198](#)

SAP BW Query key figures

externalizing, [603](#)

SAP BW query subjects, [367](#)

SAP BW structures, [201](#)

SAP BW variables, [209, 230](#)

save as

copying projects, [285](#)

saving automatically

options, [30](#)

schema, [317](#)

scope relationships, [114, 122, 123, 126, 205](#)

script files

fixing errors, [300](#)

scripts

running, [269](#)

searching

objects, [34, 36](#)

search prompts, [149, 151, 226, 227](#)

secured InfoCube

accessing, [197](#)

security

CSVIdentityName function, [258](#)

CSVIdentityNameList function, [259](#)

exploring, [271](#)

filter change in Framework Manager for SAP
BW, [19](#)

filters, [257](#)

levels, [378](#)

packages, [254](#)

types, [256](#)

segmented projects, [280, 282, 372](#)

copying, [285](#)

deleting, [287](#)

moving, [286](#)

synchronizing, [302](#)

segmenting

limitations, [281](#)

projects, [282](#)

segments, [280](#)

recommendations for using, [280](#)

self-joins, [191](#)

semi-additive, [148](#)

semi-additive measures, [146, 225](#)

Semi-Aggregate property, [148](#)

SAP BW metadata, [222, 225](#)

session parameters

modifying for relational metadata, [165](#)

modifying for SAP BW metadata, [245](#)

setting

governors, [304](#)

suppression, [263](#)

settings

font, [33, 47](#)

level of detail, [33, 47](#)

notation, [33, 47](#)

snap options, [33, 47](#)

test, [102, 129, 215](#)

shared dimensions, [336](#)

shortcuts, [333](#)

behavior, [304, 306](#)

security, [260](#)

shortcuts for relational metadata

dimensions and query subjects, [185](#)

relationships, [83, 185](#)

using, [184](#)

- shortcuts for SAP BW metadata
 - dimensions, [248](#)
 - using, [247](#)
- show dependencies, [289](#)
- single fact queries, [351](#)
- single quotation marks, [168](#)
- snap options
 - settings, [33](#), [47](#)
- snowflaked data sources, [343](#)
- snowflaked dimensions, [183](#)
- sorting
 - levels, [121](#)
 - members, [121](#)
 - objects, [34](#)
- sort objects, [31](#)
- sparse data
 - semi-additive measures, [146](#)
- sparse data and Analysis Studio, [81](#)
- split queries, [361](#)
- SQL, [351](#)
 - As View, [112](#), [113](#)
 - changing type, [107](#)
 - Cognos, [106](#), [109](#)
 - comments, [106](#)
 - determinants, [96](#)
 - editing, [106](#)
 - generation types, [112](#)
 - inner joins, [378](#)
 - Minimized, [112](#), [113](#), [191](#)
 - model query subjects, [113](#)
 - native, [106](#), [110](#)
 - pass-through, [106](#), [111](#)
 - query subjects, [85](#)
 - server, connecting, [56](#)
 - stand-alone native, [106](#), [111](#)
- SQL generation
 - controlling, [304](#)
- SQL Generation for Determinant Attributes
 - governor, [308](#)
- SQL Generation for Level Attributes
 - governor, [308](#)
- SQL Parameter Syntax
 - governor, [309](#)
- square brackets, [168](#)
- star dimensions, [183](#)
- star schema concepts, [343](#)
- star schema groups, [336](#)
 - creating, [348](#)
 - multiple conformed, [348](#)
- star schemas
 - creating using Model Design Accelerator, [48](#)
 - groupings, [180](#)
 - managing using Model Design Accelerator, [50](#)
 - multiple, [181](#)
- statistics
 - viewing, [36](#)
- stitched queries, [339](#)
- stored procedure query subjects, [88](#)
 - Composite, [89](#)
 - creating, [90](#)
 - definition, [25](#)
 - evaluating, [104](#)
 - example, [92](#)
 - Informix, [89](#)
 - modifying, [90](#)
 - updating, [104](#)
- stored procedures
 - data query updates database, [90](#)
- structures
 - SAP BW, [201](#)
- summaries
 - viewing, [36](#)
- summarize aggregations
 - relational metadata, [144](#)
- supported environments
 - multilingual, [131](#)
- supporting
 - multiple users, [273](#)
- suppression
 - multi-edge, [263](#)
 - null, [263](#)
- suppression options, [263](#)
- synchronizing
 - fixing errors, [300](#)
 - linked projects, [303](#)
 - metadata, [301](#)
 - projects, [301](#)
 - query subjects, [104](#)
 - segmented projects, [302](#)
- syntax
 - BmtScriptPlayer, [298](#)

T

tab files, [263, 266](#)

tasks

viewing, [36](#)

testing

changing settings, [102, 129, 215](#)

data source connections, [58](#)

projects, [251](#)

prompts, [154, 230](#)

query subjects, [74, 376](#)

relational measure dimensions, [128](#)

relational query sets, [101](#)

relational query subjects, [101](#)

relational regular dimensions, [128](#)

SAP BW measure dimensions, [213](#)

SAP BW regular dimensions, [213](#)

text box prompts, [149, 151, 226, 227](#)

text format type, [148, 225](#)

time-dependent hierarchies, [14](#)

time limits, [304](#)

time prompts, [149, 151, 226, 227](#)

trace

transaction history, [296](#)

transaction history

viewing, [296](#)

transactions

fixing, [279](#)

play back, [296](#)

Transformer

SAP BW, [602](#)

Transformer and query subjects, [263, 266](#)

Transformer version 7.x

externalized CSV files, [602](#)

Transformer version 8.x

externalized CSV files, [602](#)

translation tables

exporting, [135](#)

importing, [136](#)

troubleshooting, [375](#)

type-in prompts, [149, 151, 226, 227](#)

type-in SQL, [377](#)

U

UDA-SQL-0107, [376](#)

UDA-SQL-0114, [376](#)

unable to access service at URL, [375](#)

unbalanced hierarchies, [119, 207](#)

understanding

metadata, [271](#)

unexplained number calculations, [379](#)

union all query set, [97](#)

union query set, [97](#)

unique identifiers, [92](#)

objects, [38](#)

unique keys, [368](#)

unique levels, [115](#)

uniquely identified determinants, [92, 94](#)

updating

model versions, [266](#)

query subjects, [104](#)

upgrading

IBM Cognos ReportNet models, [365](#)

linked projects, [372](#)

models, [27, 365](#)

segmented projects, [372](#)

upgrading models

warnings, [371](#)

URI

unable to access service, [375](#)

Usage property

relational metadata, [141, 144](#)

SAP BW metadata, [222, 223](#)

users

adding, [256](#)

packages, [254](#)

using

multiple data sets, [292](#)

multiple data source connections, [53, 56](#)

using embedded strings

in expressions, [245](#)

in macros, [245](#)

using literal strings

in expressions, [245](#)

in macros, [245](#)

using quotation marks

in expressions, [245](#)

in macros, [245](#)

V

validating

relational query subjects, [104](#)

SAP BW query subjects, [218](#)

- valid relationships
 - multiple, [339](#)
- vargraphic datatype, [376](#)
- variables
 - guidelines for SAP BW fact data, [595](#)
 - SAP BW, [230](#)
 - stored procedure, [92](#)
 - using when externalizing, [605](#)
- vendor-specific database functions, [314](#)
- verifying
 - IBM Cognos ReportNet models, [365](#), [367](#)
 - models, [251](#)
 - packages, [251](#)
 - relationships, [78](#)
- versioned hierarchies, [200](#)
- versioning
 - models, [266](#)
- viewing
 - diagrams, [32](#), [44](#), [45](#)
 - model objects, [28](#), [43](#)
 - package inclusion, [272](#)
 - properties, [34](#)
 - related objects, [97](#), [127](#), [213](#)
 - SQL, [106](#)
 - statistics, [36](#)
 - summaries, [36](#)
 - tasks, [36](#)

W

- With clause, [304](#)

X

- XML files
 - importing, [75](#)