IBM Cognos Analytics
Version 11.1

Metadata Modeling Guidelines

©

Product Information

This document applies to IBM Cognos Analytics version 11.1.0 and may also apply to subsequent releases.

Copyright

Contents

Chapter 1. Metadata modeling

The purpose of metadata modeling is to present a user-friendly representation of the data that is available to your query audience while adding business value where required.

The metadata modeler's job is to ensure consistent and expected results for users who create or consume content in IBM® Cognos® Analytics.

This document discusses how to effectively leverage the modeling concepts supported by IBM Cognos Analytics, and ensure accurate and performant query results.

This document doesn't cover step-by-step instructions. Where appropriate, links to the related component documentation are provided.

Terminology clarification

This document spans multiple IBM Cognos Analytics metadata modeling tools that might each use slightly different terminology for objects.

For example, in Framework Manager query objects are called data source query subjects, and model query subjects. In data modules, parallel objects are called tables, table views, table copies, table unions, and so on. In general, all of these objects represent a table or view from a data source, and the columns within. For the purposes of this document, the terms tables and columns are used.

Planning your project

Gathering requirements, carefully choosing your data sources, and considering the best design approach produce better performing metadata.

Gather requirements

Gather the business requirements first. It's not recommended to simply see what data is available, and then work around it. Ideally, the source data should be structured and enriched for analytical applications. A common approach is to combine star schemas to represent a conformed dimensional warehouse that spans one or more subject areas in a business.

Ensure that your IBM Cognos Analytics applications are built on properly structured data sources that meet the users' query requirements and performance expectations. This will allow you to avoid queries that continually "restructure" and "enrich" data at run time. It's best to have processes that feed the source data, such as the extract, transform, and load (ETL) process, to procure and store the required data. Performance starts with the data source and any optimizations that can be implemented within it, such as aggregate tables, materialized views, precomputed calculations, indexing, and so on.

Consider the following, additional questions:

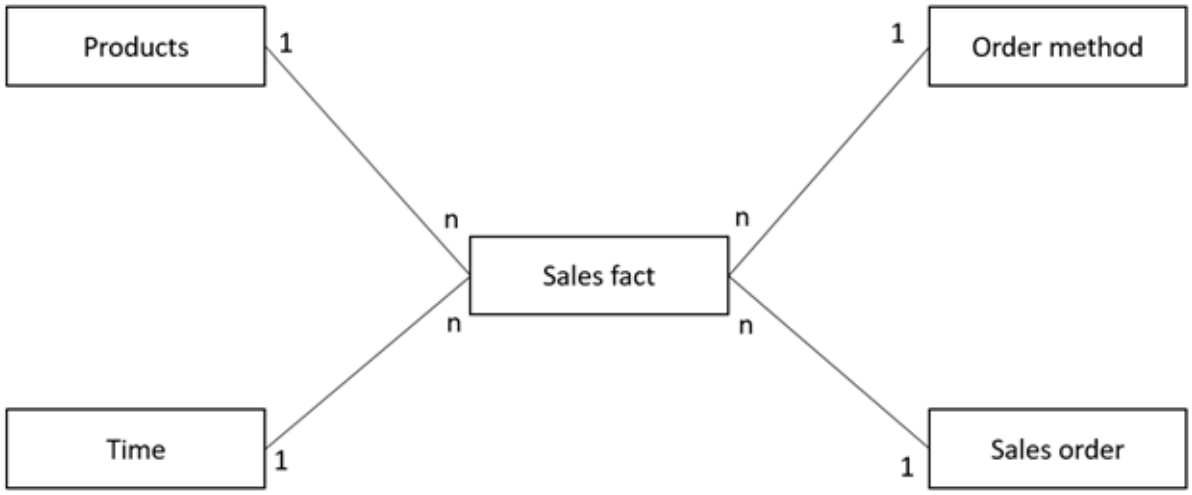
- What are the performance expectations?
- How current the data needs to be?
- What level of detail is required?

The answers to these types of questions dictate what data sources to use, what data the sources should contain, what level of detail is stored (for example, hour versus day, versus week, versus month), how often the data is refreshed, and so on.

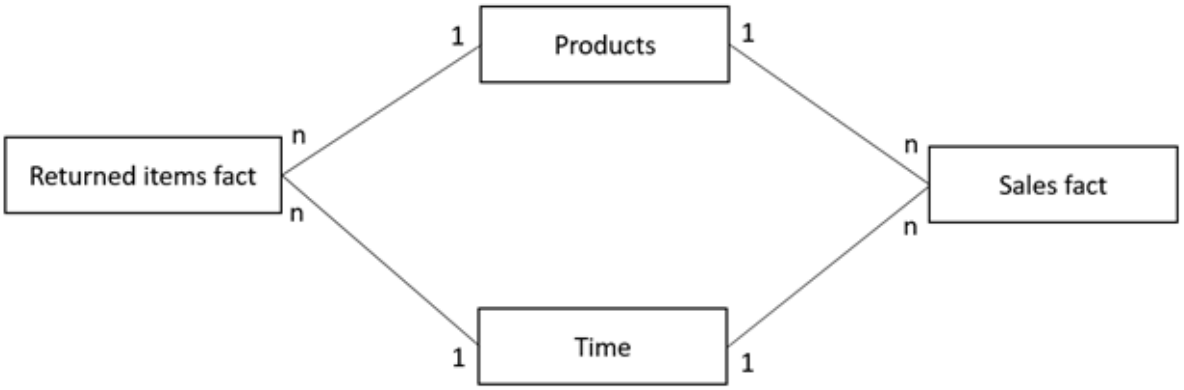
Choose data sources

Cognos Analytics performs best with star or snowflake schemas. These schemas consist of dimension tables that contain attributes, such as name, date, color, or city, which are used to categorize data. They also use fact tables that contain key performance indicators (KPI), otherwise known as measures or facts.

In the following example, `Products`, `Time`, `Order method` and `Sales order` are dimension tables, and `Sales fact` is the fact table.



Dimension tables that are referenced by multiple fact tables are known as shared or reference dimension tables. In the following example, `Products` and `Time` are shared dimensions to the `Returned items fact` and `Sales fact` tables.



With this type of structure for your enterprise data, the metadata modeling process is greatly simplified, which ensures best performance. Of course, performance can be impacted by several factors that include data volume, system resources, database vendor, or database optimizations, such as indexes. All of these elements must be considered when undertaking a Cognos Analytics project. Testing against the database structure directly, outside of Cognos Analytics, ensures that it performs as expected.

Cognos Analytics can query against many types of data sources, not just databases. Microsoft Excel or CSV files can be used. Or your requirements might see an OLAP source as the best option. For example, you might have users who are knowledgeable about dimensional functions and require an OLAP data source to accomplish their queries. You must consider what works best and satisfies the needs of users.

Data sources can also be mixed and matched. For example, you can use a CSV file and a database table in the same project and create a relationship join between them. However, caution should be used as performance might be an issue if these two sources aren't combined properly. Using certain optimization techniques that are discussed later in this document can help to improve performance when combining disparate data sources in a project.

Think about design

As mentioned before, Cognos Analytics works best with the star or snowflake schema data warehouse structure. The Cognos Analytics query service is optimized to recognize dimension and fact tables based

on the nature of the join between tables. The more clearly a metadata modeler can identify and configure fact and dimension tables, the higher is the success rate of expected and performant results.

Metadata modeling workflow

Metadata modeling is an iterative process, as you prepare the metadata for the purposes of reporting, dashboarding, and exploring data. There is a general starting point and a workflow to follow. The sections in this topic illustrate the workflow steps at a high level, along with links to more detailed content.

Import and verify metadata

Once your requirements are gathered and the supporting data sources are available, in the metadata modeling tool, you can import the required metadata to support the user requirements. You might be tempted to import everything, and then decide what to use. From a model maintenance, readability, and usability standpoint this is not recommended. Import only what you need, and then add more later as the requirements change.

For more information, see Chapter 2, “Import and verify metadata,” on page 5.

Remove ambiguity

Ambiguity in model design refers to potential misinterpretations of relationships and their cardinality by the Cognos Analytics query service. You can remove ambiguity in the model by ensuring that the correct join paths are used in your queries, and that the intended fact tables and dimension tables are always treated as such. This design practice produces the expected aggregation for your measures.

For more information and background on resolving ambiguity, see Chapter 3, “Remove ambiguity,” on page 7.

Consider model design

In this phase, you need to consider how to present objects to the users in a clear, logical, and concise manner. You can consolidate multiple tables into one view, consolidate logically grouped tables into one area of the model, add filters and calculations as required, and so on.

For more information, see Chapter 4, “Model design and presentation,” on page 17.

Identify and configure for multi-fact, multi-grain queries

There might be scenarios where facts from different fact tables are stored at different levels of granularity, which refers to the scale or level of detail that is present in the set of data. For example, the `Inventory Fact` table stores values at the month level while the `Sales Fact` table stores values at the day level. Different levels of granularity might introduce scenarios where one fact is inadvertently double-counted (aggregated more times than it should based on the nature of the data). As a metadata modeler, you should identify these potential scenarios and configure the model accordingly to prevent double-counting.

For more information, see Chapter 5, “Multi-fact, multi-grain queries,” on page 21.

Enhance the model with additional features

Based on the needs and requirements of users, metadata modelers might use various techniques and features to enhance the model. For example, some users might want to do relative date comparisons of the data. Each metadata modeling tool has a way to accomplish this type of request.

Framework Manager uses dimensionally modeled relational (DMR) models for this purpose. A modeler creates dimensional objects that allow users to drill up or down through the data based on defined hierarchies. Dimensional functions can also be used to extract and compare data from different time periods or segments of the business. In data modules, modelers can implement navigation paths and relative date calendars to accomplish similar user requirements.

Other features to enhance the model or its performance include, but are not limited to, ensuring minimized SQL and controlling the way data is aggregated.

For more information, see Chapter 6, “Enhance the model with additional features,” on page 23.

Consider performance

As you develop your model, you need to constantly test for performance. Performance starts with the data source that you report on. However, there are also some key optimizations that can be accomplished within Cognos Analytics, such as leveraging data caches, data sets, and join optimizations across heterogeneous data sources.

For more information, see Chapter 7, “Optimizing query performance,” on page 29.

Iterate

Again, as with any project, the metadata modeler develops the model based on requirements, tests often, and then iteratively changes the model until the desired results are achieved.

Chapter 2. Import and verify metadata

After identifying the metadata that can be used in your model, import only the required items to keep your model manageable. Also, verify the **Usage** and **Aggregate** properties on columns.

The following **Usage** settings are supported:

Identifier
Represents a key, code, or date.

Attribute
Represents context, such as name, color, geographic information.

Measure
Represents key performance indicators that are typically aggregated in the context of identifiers or attributes.

By default, the **Aggregate** property for an identifier or attribute is set to **Count**, and for a measure to **Sum** or **Total**. Even though a fact table might hold a numeric attribute or identifier (for example, an order number), it doesn't mean that the item in the fact table is a measure, and its aggregation should be set accordingly, such as **Count** or no aggregation. The same logic applies to a dimension where an attribute, such as population or square footage, might not necessarily warrant an aggregation setting of **Sum** or **Total**, but rather the setting of no aggregation.

You typically want to ensure that integer values that represent keys, codes, or non-rollup integer attributes in your data have the **Usage** property set to **Attribute** or **Identifier**, and the **Aggregate** property set to **Count**. It's not logical for these values to behave as measures, and be totaled or averaged.

For information on setting column properties in data modules, see ."Object properties" in the *IBM Cognos Analytics Data Modeling Guide*.

For information on setting query item properties in Framework Manager, see "Query items" in the *IBM Cognos Framework Manager User Guide*.

Data modules can denote that columns are from the domain of geographic (city, state, country, and so on) or temporal types (day, month, quarter, year, and so on). This extended metadata can be automatically leveraged in queries. For example, data values that represent geographic data automatically have their **Represents** property set to **Geographic location**. With time-based data, this property is automatically set to **Time**. Cognos Analytics also analyzes the data and allows for other functionality, such as data visualization recommendations. Verify the **Represents** property on columns to ensure that it's set as expected.

For Framework Manager models, you can enrich a published package to achieve the same artificial intelligence (AI) functionality as in data modules.

For more information, see "Enriching packages" in the *IBM Cognos Analytics Data Modeling Guide*.

Chapter 3. Remove ambiguity

Ambiguity in model design refers to potential misinterpretations of relationships and their cardinality by the Cognos Analytics query service.

The topics in this section provide important knowledge on how the query service interprets relationships, and how to model the metadata for expected and accurate results.

Cardinality and how Cognos Analytics uses it

Tables are related using relationships that denote the numerical number of related rows in each table. Common relationships are 1 to many, and 1 to 1.

Relationships between two tables reference one or more columns from both tables. Typically, the relationship reflects the referential integrity defined in a database (primary, unique, and foreign keys). When metadata is imported, IBM Cognos Analytics attempts to locate any available referential integrity to create default relationships.

The Cognos Analytics query service uses the cardinality of a relationship in the following ways:

- To identify tables that behave as facts (n side of the relationship) or dimensions (1 side of the relationship).
- To avoid double-counting of measures.
- To support loop joins that are common in star schema models.

A relationship can specify the minimum-maximum cardinality and optional cardinality.

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

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

A relationship with cardinality specified as 1:1 to 1:n is commonly referred to as 1 to many when focusing on the maximum cardinalities.

A minimum cardinality of 0 indicates that the relationship is optional. As an example, a relationship between `Customer` and `Sales` might be defined as 1:1 to 1:n by default. In this case, customers without sales aren't returned because cardinality on both sides isn't optional. To include customers without sales, use the 1:1 to 0:n cardinality. This cardinality indicates that queries will show the requested customer information even though there might not be any sales data present.

Relationships can be defined to describe the following scenarios:

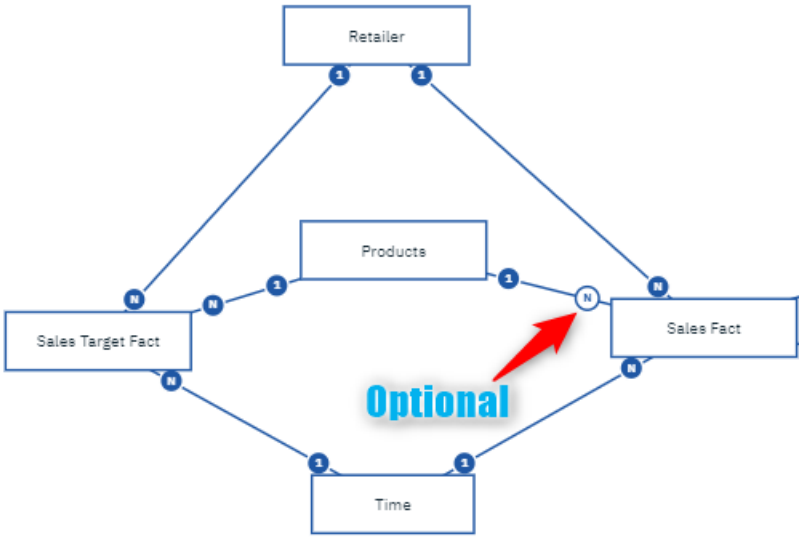
- 1:1 to 1:n (inner join)
- 0:1 to 1:n (right outer join)
- 0:1 to 0:n (full outer join)
- 1:1 to 0:n (left outer join)

Ensure that the cardinality is correctly defined in your metadata to avoid ambiguity. Tables with only 1 cardinality are always considered dimensions in the context of a query, while tables with only n cardinalities are always considered facts. Tables with a mix of 1 and n cardinalities are defined either as dimensions or facts, depending on the context of the query. For more information about context as it relates to queries, see "Cardinality in the context of a query" on page 8.

When generating queries, the Cognos Analytics query service follows these basic rules to apply cardinality:

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

In Framework Manager, the default annotation in the relationship diagram uses 1..1 or 0..1 and 1..n or 0..n to represent the minimum and maximum cardinalities. In data modules, 1 and n are displayed to show the maximum cardinalities in the relationship diagram, and optional cardinality is indicated by a white background versus a blue background in the cardinality annotation, as shown in the following screen capture.



Note: If you try to create a join between two tables where the data types of the keys don't match, you might be tempted to cast one of the keys to match the data type of the other key. This isn't recommended as performance can be negatively impacted. The data type mismatch should be resolved in the database so that the database optimization for primary and foreign keys can be leveraged. When you use a calculation in the modeling tools to cast a data type, a new data column is created at run time that doesn't exist in the database and won't be optimized.

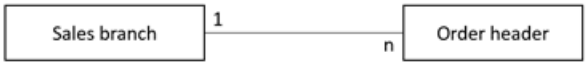
Cardinality in the context of a query

Depending on the context, cardinality can be interpreted differently by the IBM Cognos Analytics query service.

The following examples show possible interpretations.

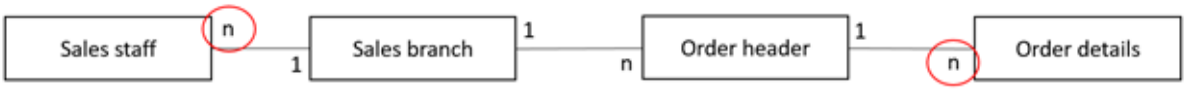
Example 1: Tables behaving as a dimension and a fact

In this example, `Sales Branch` behaves as a dimension relative to `Order header`, and `Order header` behaves as a fact relative to `Sales branch`.



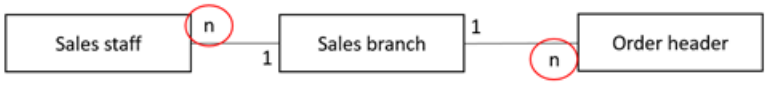
Example 2: Four tables in a query

In this example, all four query tables are included in a query. `Sales staff` and `Order details` are treated as facts. `Order header` and `Sales branch` are treated as dimensions. In this scenario, `Order header` is present on the many side and the 1 side of the relationship. In this context, the Cognos Analytics query service treats `Order header` as a dimension table, and avoids double counting of any measures in this table.



Example 3: Three tables in a query

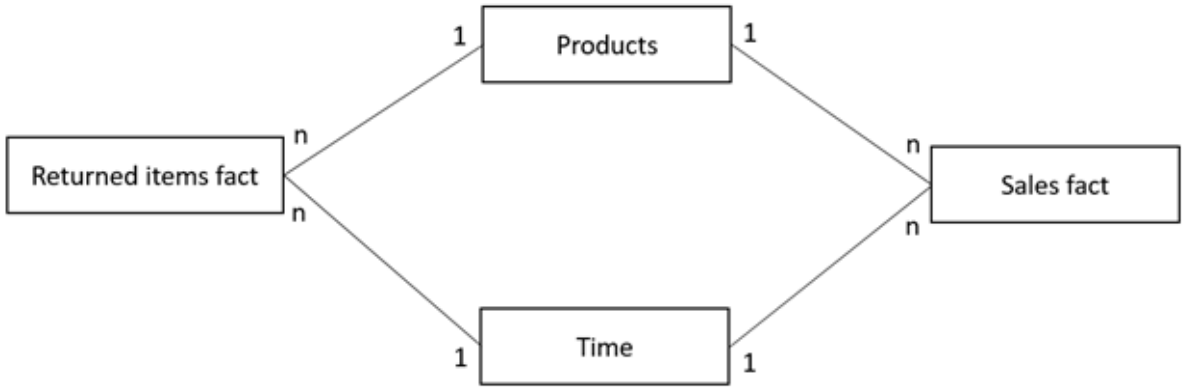
In this example, only three tables are included in a query from the previous example. `Order details` isn't used. `Order header` is now treated as a fact. `Sales staff` continues to be treated as a fact.



Stitch queries

When queries that request facts from multiple tables are performed in IBM Cognos Analytics, the query service performs what IBM Cognos calls a stitch query. Stitch queries consist of subqueries, one for each fact table, that are then merged together on their common attributes from a shared dimension table.

In the following example of a model, `Products` and `Time` are clearly shared dimensions, based on the cardinality defined between `Returned items fact` and `Sales fact`.



In the following results of the query that is based on this example, you see in the last row that in 2013 there were no returns for `Hibernator Pad`.

Year	Product	Quantity	Return quantity
2010	Hibernator Pad	98,303	1,320
2011	Hibernator Pad	137,367	1,980
2012	Hibernator Pad	153,179	1,682
2013	Hibernator Pad	117,060	

The following pseudo SQL shows that as the query was run, Cognos Analytics created two subqueries, one for `Sales` and one for `Returns`. Data from the subqueries is combined using a full outer join operation on the `Year` and `Product` columns. In 2013, there were sales but no returns for the `Hibernator Pad`, hence a null value is returned for the `Return quantity`.

```
Select
      coalesce(D2.Year1,D3.Year1) as Year1,
      coalesce(D2.Product_Name,D3.Product_name) as Product_name,
      D2.Quantity as Quantity,
      D3.Return_quantity as Return_quantity
from
    (Sub query 1) D2
full outer join
     (Sub query 2) D3
```

```
on
    ((D2.Year = D3.Year) and (D2.Product_name = D3.Product_name))
```

Examine the results of each subquery. First, look at the subquery results that retrieve data for the `Quantity` fact. The query returned the following four records.

Year	Product	Quantity
2010	Hibernator Pad	98,303
2011	Hibernator Pad	137,367
2012	Hibernator Pad	153,179
2013	Hibernator Pad	117,060

Now, examine the subquery results for the `Return quantity` fact. Notice that there are only three records. There were no returns for 2013.

Year	Product	Return quantity
2010	Hibernator Pad	1,320
2011	Hibernator Pad	1,980
2012	Hibernator Pad	1,682

However, in a stitch query, when there are more records in one subquery than in the other subquery, nulls are returned for the rows where there is no match, as seen in the following query result (which was presented earlier in this section):

Year	Product	Quantity	Return quantity
2010	Hibernator Pad	98,303	1,320
2011	Hibernator Pad	137,367	1,980
2012	Hibernator Pad	153,179	1,682
2013	Hibernator Pad	117,060	

In the pseudo SQL example earlier, the `coalesce` function is used to return the first non-null record set from the subqueries. If both are null, no record is returned. If one is null and the other is not, a record is returned, but the subquery that had no match displays a null value.

For more information about dealing with nulls in calculations in reporting, see this article (www.ibm.com/support/pages/node/6252027).

If dimensions and facts are incorrectly identified, stitch queries can be created unnecessarily, which can be costly to performance. Or the queries can be incorrectly formed, which can give incorrect results.

In some instances, fact detection and stitch queries are not desired. In these cases, you must know the data well and be sure that the relationships are 1 to 1, and the summary row aggregations would be incorrect. This typically occurs with combination analysis scenarios.

For more information on combination analysis, see this article (www.ibm.com/support/pages/node/6252021).

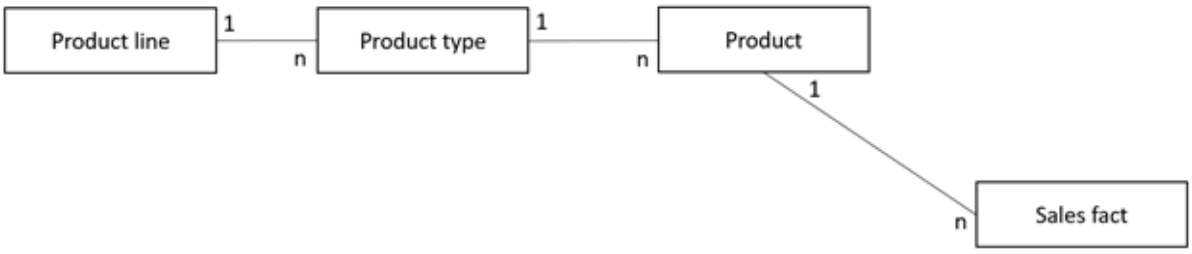
Verify relationships

The metadata modeling tools can detect and create relationships between tables during import. However, after import, it's always a good practice to ensure that the relationships are as you intend them to be to meet the query requirements.

Are the fact tables truly fact tables in that they have only n cardinality attached to them? Are the dimension tables truly dimension tables with only 1 cardinality attached?

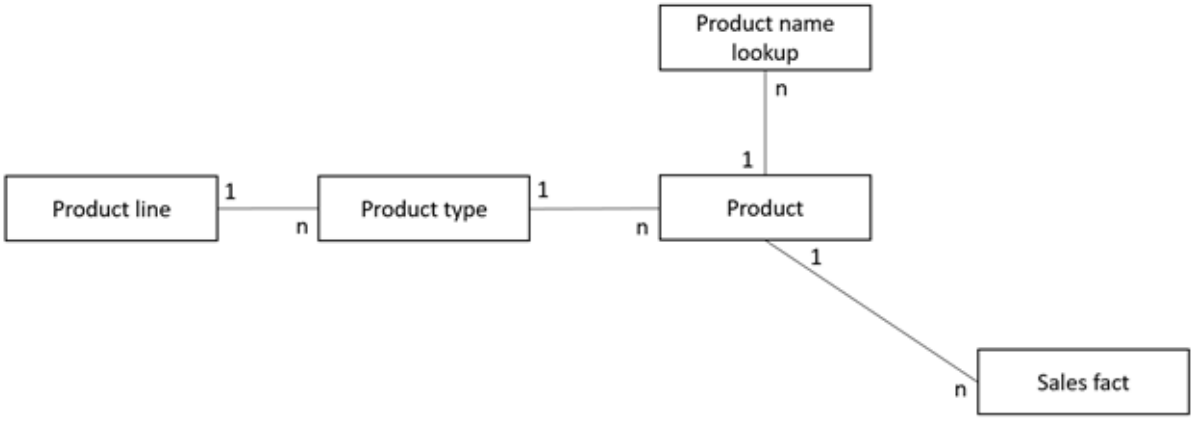
There might be some exceptions with dimension tables in that they might be snowflake dimensions. A snowflake dimension consists of multiple tables that all represent the overall dimension. The tables are normalized.

The following `Product` tables are an example of a snowflake dimension:

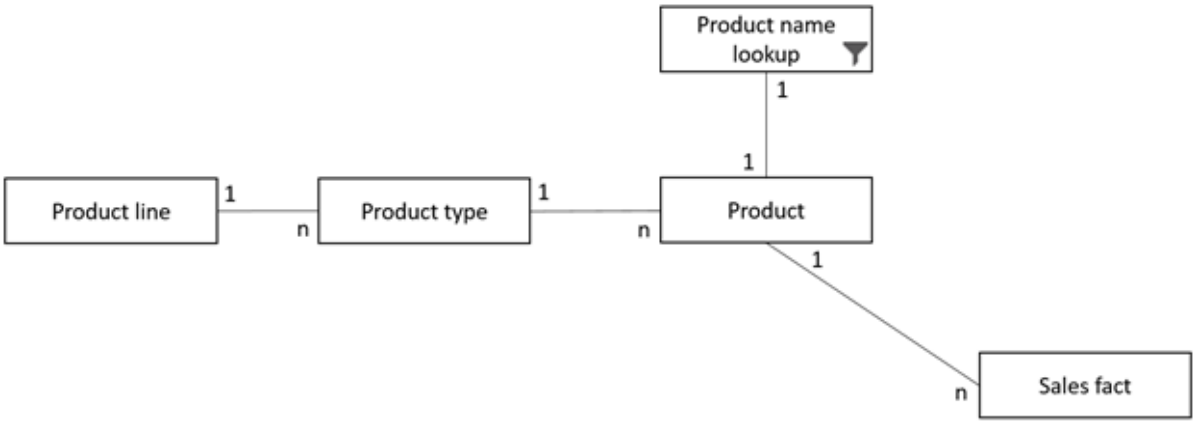


`Product line`, `Product type`, and `Product` all make up the `Product` hierarchy for the `Product` dimension. You might ask, since `Product type` and `Product` both have n cardinality attached, couldn't they be seen as fact tables in the context of a query. In this case, no. There is a 1 to many linear path to the `Sales fact` table. The Cognos Analytics query service doesn't see any of these product tables as fact tables in the context of a query.

Consider the following scenario:



In this scenario, if `Product`, `Product name lookup`, and `Sales fact` were all included in the query, both `Product name lookup` and `Sales fact` would be treated as fact tables and create an unnecessary stitch query. Is the relationship between `Product` and `Product name lookup` truly a 1 to many? Upon investigating the data, `Product name lookup` is a multilingual table that has multiple rows for each product in the `Product` table. So this is truly a 1 to many relationship. But based on the business requirement, we need to see only one language at a time, and the language chosen is based on the user's locale setting in Cognos Analytics. Therefore, we can add a filter to the `Product name lookup` table. The filter would return only one row per product, which changes the nature of the relationship to 1 to 1, as shown in the following scenario:



As you investigate your relationships, always ensure that dimensions are treated as dimensions, and facts as facts. A simple star schema database design makes this job easy. However, in many cases, there might

be some scenarios in the database design that need to be addressed in the model. Later in this document, we look at options to consolidate snowflake dimensions.

Resolve ambiguous relationships

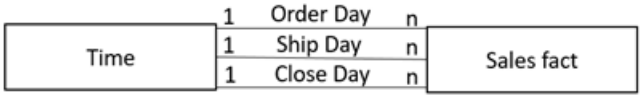
Ambiguous relationships occur when the data represented by a table or dimension can be viewed in more than one context or role, or can be joined in more than one way.

The most common ambiguous relationships are role-playing dimensions, loop joins, and reflexive and recursive relationships. For details about these types of relationships, see the subsections in this topic.

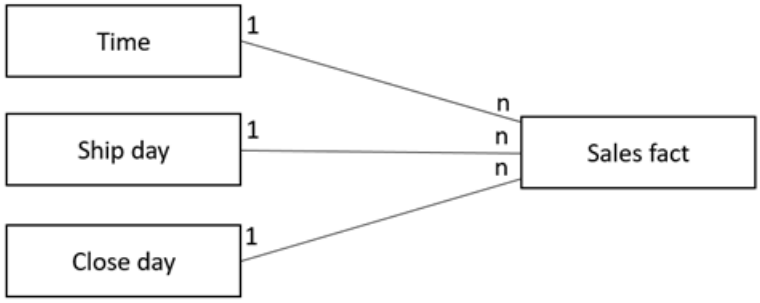
Role-playing dimensions

A table with multiple relationships between itself and another table is known as a role-playing dimension.

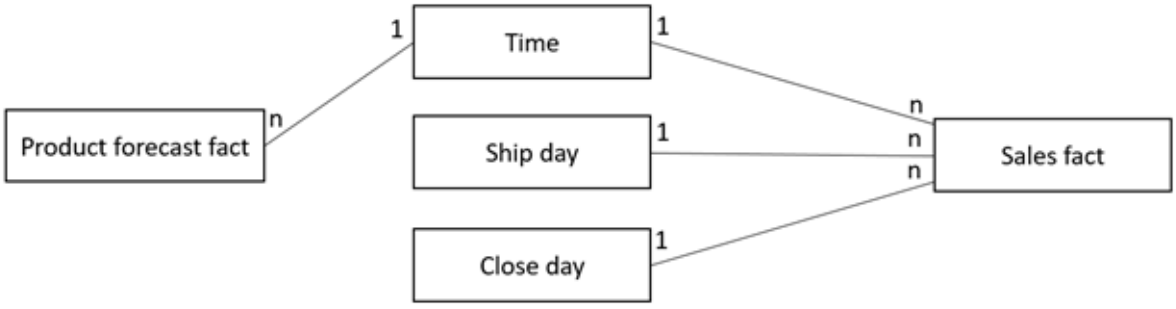
For example, the `Sales fact` in the following example has multiple relationships to `Time` on the keys `Order Day` , `Ship Day`, and `Close Day`.



Create a table for each role that you want the `Time` table to play. If your role-playing dimension requires only a subset of columns from the original table, you can remove unneeded columns for a cleaner presentation, and provide appropriate names, such as Date in `Time`, Ship Date in `Ship Day`, and Close Date in `Close Day`. You can name the keys appropriately as well. For example, rename `Day Key` to Ship Day Key in `Ship Day` and to Close Day Key in `Close Day`. Ensure that a single, appropriate relationship exists between each role-playing table and the fact table on the appropriate key. In the following example, the `Time` table is used to represent the `Order Day` for a sale. The other two role-playing dimensions, `Ship Day` and `Close Day`, are self-explanatory.



The role-playing dimensions that you create might or might not be applicable to other fact tables. For example, `Product forecast fact` might be applicable only to the `Time` dimension and would, therefore, be joined only to this dimension.



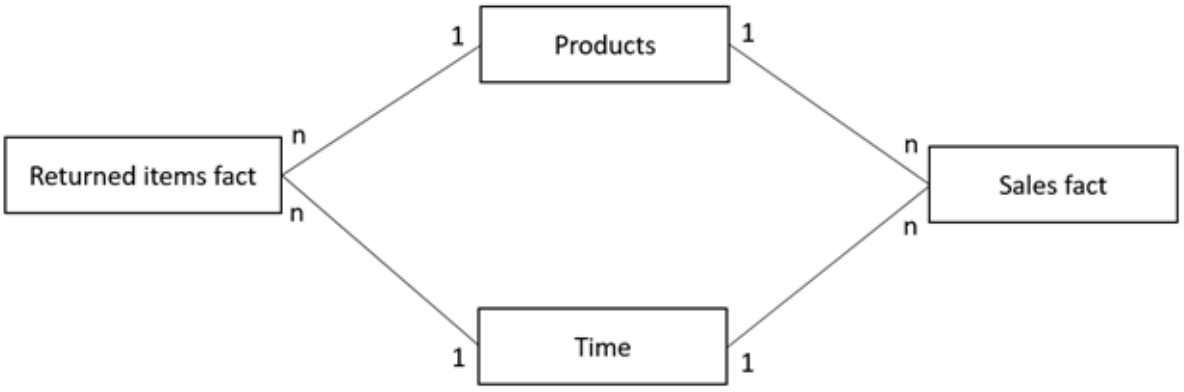
Now, `Time` can be used to compare `Product forecasts` and `Sales`.

Loop joins

Loop joins are caused when table relationships are ambiguously defined based on cardinality. This can produce unpredictable results. However, this issue doesn't apply to star schema loop joins when cardinality clearly identifies facts and dimensions.

IBM Cognos Analytics 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.

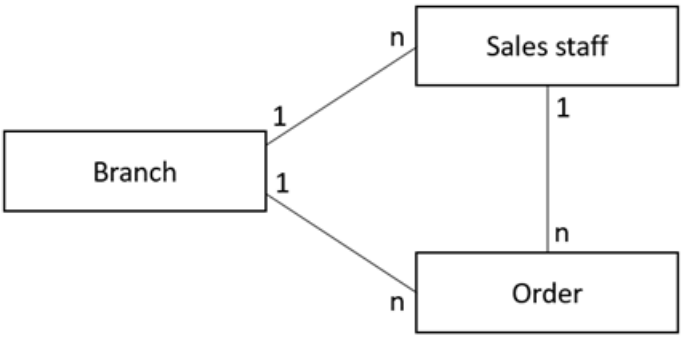
In the following example of a loop join multiple fact tables are joined to a common set of dimension tables.



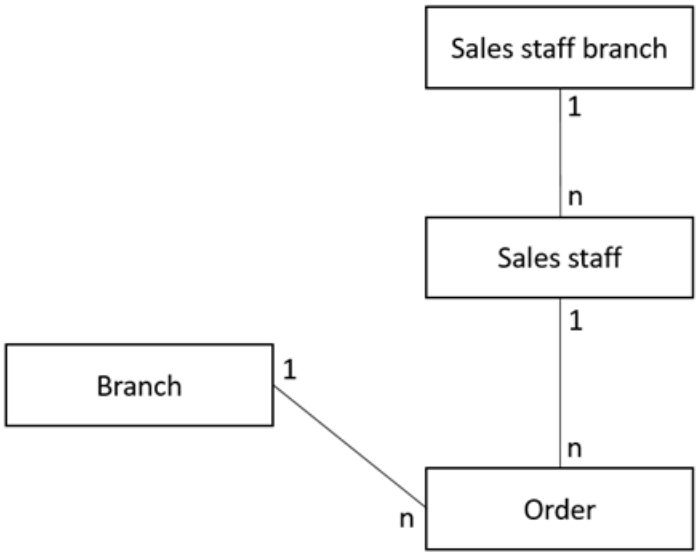
When tables are ambiguously defined based on cardinality, which means that it's not clear if a table is a fact or a dimension, and are part of a loop join, the joins that are used in a query are decided based on a number of factors, including the following ones:

- Location of the relationships
- Number of segments in join paths
- First join path in the alphabetical order (if all other factors are equal)

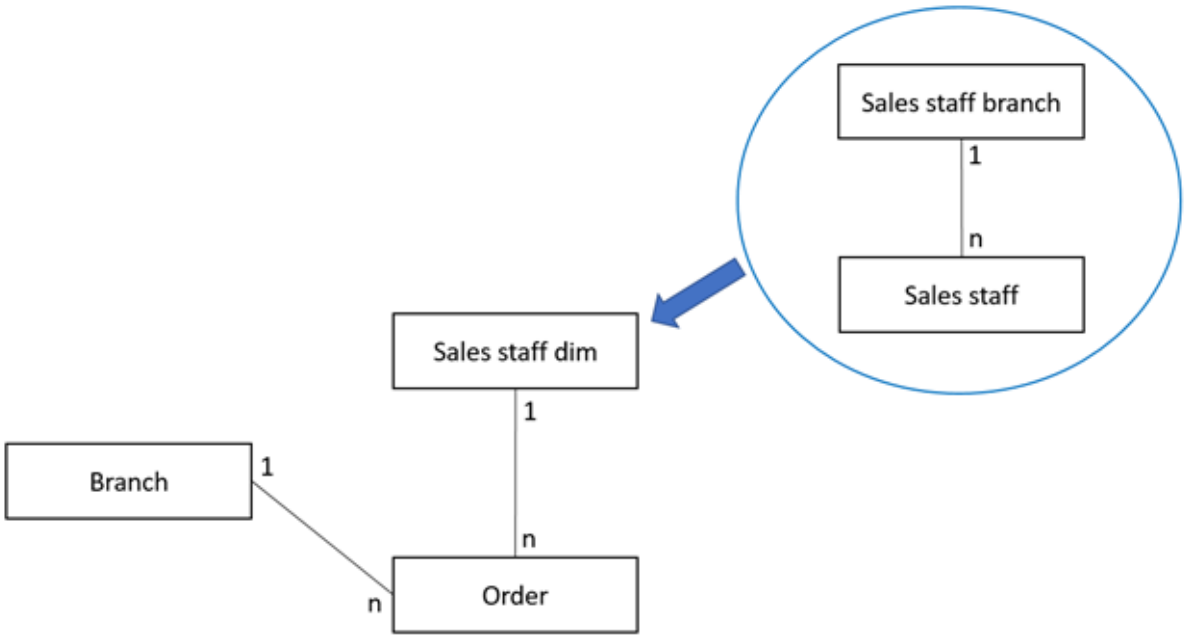
In the following scenario, the `Sales staff` table is ambiguously defined based on cardinality. The table has the 1 and n cardinality attached, and it's a part of a loop join. If columns from all three tables are selected in a query, it's unclear which join paths would be selected. Would it be `Branch` to `Sales staff` and `Order`, and the join between `Sales staff` and `Order` would be ignored? Would it be `Branch` to `Order` and `Sales staff` to `Order`, and the join from `Branch` to `Sales staff` would be ignored?



To remove this uncertainty, you could have `Branch` act as a role-playing dimension (in this case the `Sales staff branch`) to resolve the loop join, as shown in the following example:



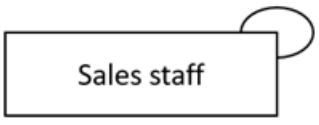
You could even combine `Sales staff branch` and `Sales staff` into one table that contains the `Staff` and `Branch` columns to simplify the presentation.



Reflexive and recursive relationships

Reflexive and recursive relationships imply two or more levels of granularity within a table with a fixed depth.

For example, the `Sales staff` table has a recursive relationship between `Sales_Staff_Code` and `Manager_Code`.



The following example shows how the data might look like in a table:

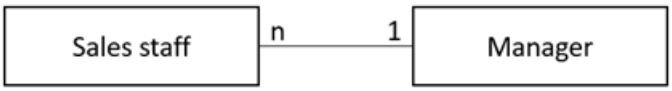
Table 1. Recursive relationship between Sales_Staff_Code and Manager_Code		
Sales_Staff_Code	**Sales_Staff_Name**	**Manager_Code**
1	Jane Smith	NULL
2	Martin Doe	1
3	Stephanie Sharaki	1

Jane Smith has a Sales_Staff_Code of 1, but no Manager_Code because she is the manager. Martin Doe and Stephanie Sharaki are both managed by Jane Smith.

To create a functioning reflexive relationship in your model, you can either create an alias shortcut (in Framework Manager only) or a copy of the table. The second option is preferred because you can name the columns accordingly, and create a relationship between the original table and the new one.

For example, create a table named Manager containing Sales_Staff_Code and Sales_Staff_Name that is based on the Sales staff table. Rename Sales_Staff_Name to Manager name in the Manager table in the model. Create a relationship with 1 to n cardinality between Manager and Sales staff that is joined on Sales_Staff_Code and Manager Code.

For a simple two-level structure, using a model table for Manager that is based on Sales staff, the model looks as follows:



For a reflexive, balanced hierarchy, repeat this structure for each additional level in the hierarchy. For example, Table 1 might also include the Director_Code, VP_Code, and so on.

You can go one step further and combine each level of the related hierarchy tables into a final Sales staff table that presents columns for all the levels in the hierarchy. This new, final table would be joined to your fact tables.

For large data volumes with many reflexive levels, performance can be impacted. Thoroughly test your work for performance to ensure that your needs and expectations are met. For performance reasons, it's recommended to flatten the hierarchy in the database into a single table that includes all the required levels in their own columns. An example is shown in the following table:

Table 2. Hierarchy flattened into a single table with all required levels in their own columns			
Sales_Staff_Code	**Sales_Staff_Name**	**Manager_Code**	**Manager_Name**
1	Jane Smith	100	Jane Smith
2	Martin Doe	100	Jane Smith
3	Stephanie Sharaki	100	Jane Smith

The same technique could be used for an unbalanced hierarchy with branches that terminate at different levels. The hierarchy would need to be flattened and balanced by padding the data for branches that terminate at higher levels, as shown in Table 2.

Chapter 4. Model design and presentation

You should always strive to present the metadata to users in a clear and concise manner, logically grouped and easy to read and understand.

Sometimes you might need to consolidate multiple tables into one view for a logical grouping. For example, if you have three tables that represent product data, you can present the data as one logical business table that simplifies query authoring. This new table transparently pulls the data from the underlying tables.

You might also want to implement calculations, filters, or prompts to meet the business query requirements. However, when doing this, consider precomputing columns in the source data versus continually reevaluating the logic in the model.

However, precomputing columns in the source data might not always be possible. For example, in some cases the source system stewards might not allow changes to the system. As a result, the Cognos Analytics application must continually compute expressions that you define in the model. Similarly, an application team might want to be able to rapidly adapt to changes in business requirements, and cannot wait for the warehouse to change. This can be a balancing act, however, in each situation performance should be considered first.

Framework Manager and data modules have a slightly different approach to the model design.

In Framework Manager, it's recommended to model multiple layers that include:

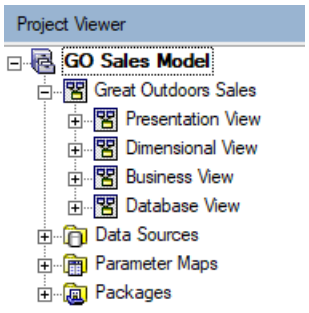
- The database layer for the imported tables.

  This layer is considered the metadata cache.
- The business layer for model enhancements and presentation.

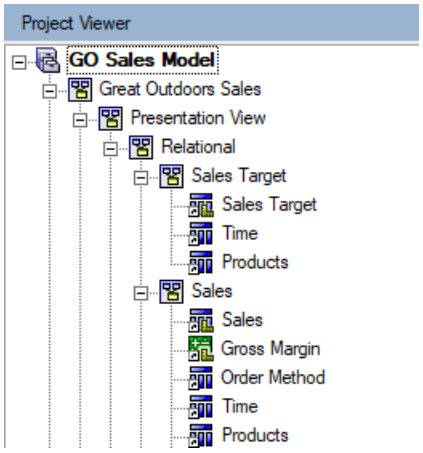
  This layer also acts as an insulation layer for reports to protect them from changes to the underlying database layer.
- The presentation layer that groups facts and related dimensions together by using star schema grouping.

  The **Star Schema Grouping** feature shows users which dimensions are shared across facts because they see the same dimension name in multiple star schema groupings. This helps the users to understand the scope of each dimension and its related fact tables. By understanding this one, simple rule, users who want to cross multiple fact tables in a query know that they must include a column from at least one shared dimension. For more information, see "Creating star schema groups" in the *IBM Cognos Framework Manager User Guide*.

The following screen capture shows the different modeling layers in Framework Manager:



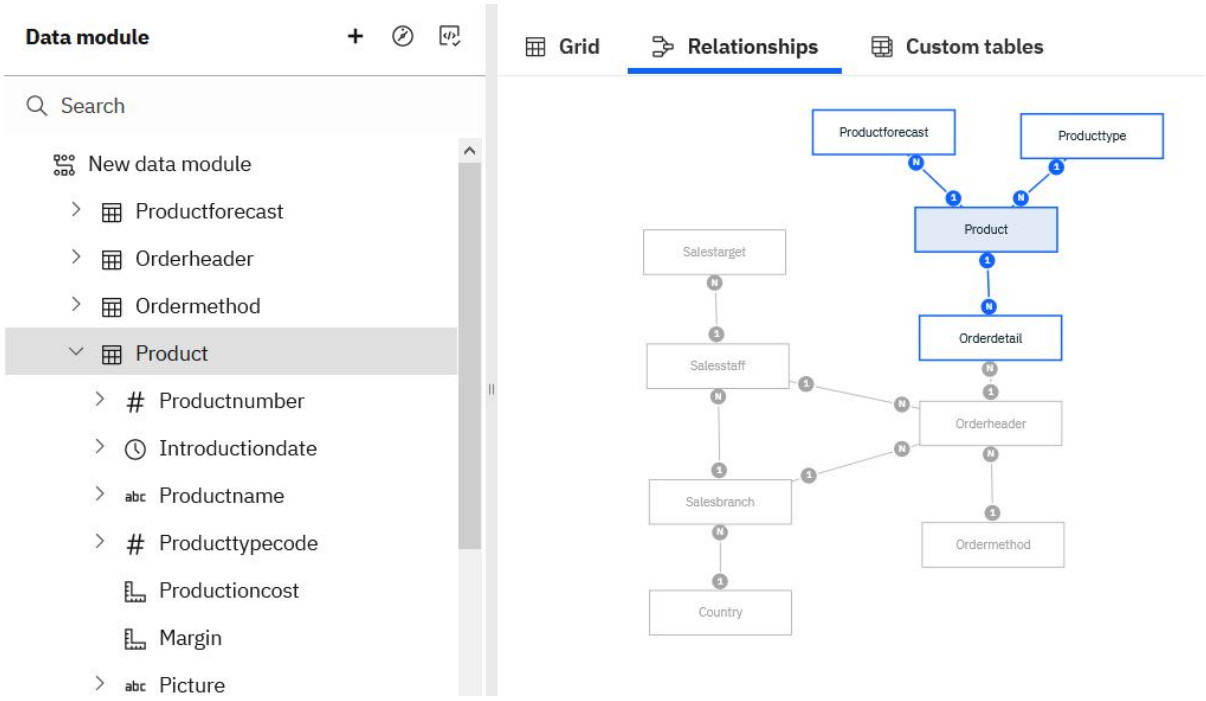
And the following screen capture shows a presentation view with star schema groupings. Shared dimensions between them can be identified based on the same name. For example, `Time` and `Products` appear in both `Sales Target` and `Sales` star schema groupings.



You might need to create and present more than one presentation layer. For example, one layer can be focused on business users who want to create their own dashboards, and another layer on professional report authors who need to address more complex requirements.

In data modules, the metadata cache is managed for you, bypassing the need for the database layer. You can focus on items to consolidate and the business value that you want to add for users. Although **Star Schema Grouping** is not a feature in data modules, users can be given read-only permissions for the data module to view its scope and examine the relationships between dimensions and facts. Unlike a Framework Manager model, a data module is available in the Cognos Analytics portal for all users with access to see.

The following screen capture shows an example of a data module in the modeling user interface:



In both modeling tools, you can create reusable objects and layers of abstraction. For example, you can create calculations or filters that can be reused in multiple locations, or copies of tables to be used in different business contexts, such as with role playing dimensions. You can develop objects in one model, and link to them from other models for reuse. This way of working enables a multi-modeler paradigm or simply reduces maintenance. For example, you might create and maintain all of your common dimensions in one model, and reference them in other models where they are joined to applicable fact tables. Changes that are made to the dimensions in the source model are propagated to the models that reference them.

Cognos Analytics supports multiple data sources in its query tools, such as Reporting or Dashboarding. In Framework Manager, you can create smaller, more manageable models rather than one large model that contains everything. Then, you can publish multiple, smaller packages that represent different aspects of the business, and visualize them in the reports or dashboards.

You can use multiple data modules as data sources in Dasboarding. However, in Reporting only one data module can be used as a source. If you need to use multiple data modules in Reporting, add these modules to a "parent" data module, and then use this module as a source.

The smaller your models are, the more manageable they are. Unless you need to query across multiple fact tables, you don't need to include all of those tables in one model. You can separate different aspects of the business into different models and reuse common dimension tables in each of them. And while you can layer model upon model, keep in mind that the more layers there are, the more difficult it is to manage, maintain, and troubleshoot the metadata.

Chapter 5. Multi-fact, multi-grain queries

Multiple-fact, multiple-grain queries occur when a table containing dimensional data is joined to multiple fact tables on different key columns.

For example, the Time dimension is joined to the Sales Fact on the Day Key (day grain), and to Sales Target Fact on the Month Key (month grain). The following tables, using some simple data, illustrate the concepts and query outputs.

Table 3. Time dimension table

Year	Quarter	Month	Day
2020	202001	20200101	Jan 1 2020
2020	202001	20200101	Jan 2 2020
2020	202001	20200101	Jan 3 2020

Table 4. Sales Fact table

Day Key	Revenue
Jan 1 2020	10
Jan 2 2020	10
Jan 3 2020	10

Table 5. Sales Target Fact table

Month Key	Sales Target
20200101	25

A dimensional table typically contains distinct groups, or levels, of attribute data with keys that repeat. In the previous example, the Time dimension illustrates this rule with repeating Year, Quarter, and Month values. Cognos Analytics automatically aggregates values to the lowest common level of granularity that is present in the query. The potential for double-counting arises when creating totals on columns that contain repeated data. For example, Sales Target Fact values, which are at the Month Key level, would repeat for every Day Key for Sales Fact values, which are at the Day Key level.

If you visualize data at a level of granularity below the lowest common level, in this case the Day level, the data of higher granularity, the Month level, is repeated, as shown in the Sales Target in the following table.

Table 6. Query results with Sales Target data incorrectly aggregated

Year	Month	Day	Revenue	Sales Target
2020	20200101	Jan 1 2020	10	25
2020	20200101	Jan 2 2020	10	25
2020	20200101	Jan 3 2020	10	25
Total			**30**	**75**

Please note that the **Total** value for Sales Target is 75, which is not correct since the Sales Target value for the month of January is only 25.

When the level of granularity of the data is modeled correctly, double-counting of the `Sales Target Fact` values is avoided, as shown in the query result in the following table.

Table 7. Query results with Sales Target data correctly aggregated

Year	Month	Day	Revenue	Sales Target
2020	20200101	Jan 1 2020	10	25
2020	20200101	Jan 2 2020	10	25
2020	20200101	Jan 3 2020	10	25
Total			**30**	**25**

For information, see “Prevent double counting” on page 22.

Non-shared dimension scenario

In the following query example, the `Order Method` table is introduced into the query. `Order Method` applies only to `Revenue` from the `Sales Fact` table, and not to `Sales Target` from the `Sales Target Fact` table. In this scenario, `Sales Target` is repeated for every row introduced by the `Order Method`, but the `Sales Target` values are not double-counted.

Table 8. Sales Target values are not double-counted

Year	Month	Day	Order Method	Revenue	Sales Target
2020	20200101	Jan 1 2020	Mail	4	25
2020	20200101	Jan 1 2020	Web	4	25
2020	20200101	Jan 1 2020	Visit	2	25
2020	20200101	Jan 2 2020	Mail	5	25
2020	20200101	Jan 2 2020	Visit	5	25
2020	20200101	Jan 3 2020	Mail	5	25
2020	20200101	Jan 3 2020	Web	5	25
Total				**30**	**25**

Prevent double counting

The modeling tools allow you to configure your models to account for scenarios where double counting might occur.

Some examples of such scenarios are documented in the topic Chapter 5, “Multi-fact, multi-grain queries,” on page 21.

To understand more about configuring your data modules to avoid double counting, see "Column dependencies" in the *IBM Cognos Analytics Data Modeling* guide.

To understand more about configuring your Framework Manager models to avoid double counting, see "Determinants" in the *IBM Cognos Framework Manager User Guide*.

Chapter 6. Enhance the model with additional features

As a modeler, you can use various techniques and features to enhance the model.

For example, you can add relative date analysis or data navigation, ensure that minimized SQL is used, and control how the data is aggregated.

The topics in this section discuss the techniques and features that you can use to enhance your model.

Relative date analysis and data navigation

The modeling tools allow for relative time analysis and data navigation, although each tool accomplishes these functions in a different way.

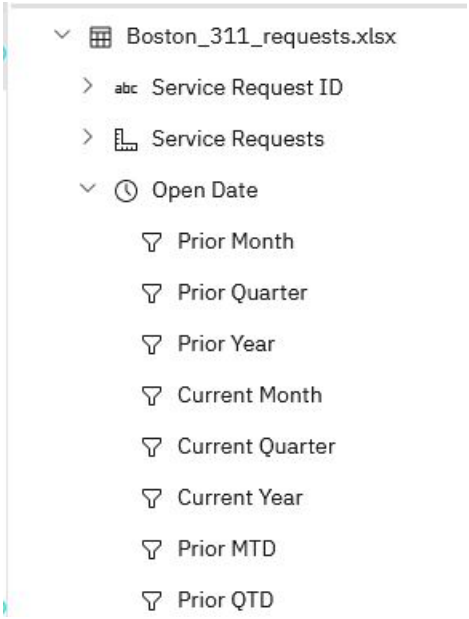
Note: This section assumes knowledge of dimensional data sources, such as OLAP and ROLAP, and the concept of cubes as a data structure, including dimensions, hierarchies, levels, members, measures, and so on.

Framework Manager supports dimensionally modeled relational (DMR) modeling, which is similar to ROLAP. This type of modeling allows Cognos Analytics to build a cube cache of the data to perform traditional, OLAP-style queries by using the multidimensial expression (MDX) query language, which is used to query dimensional sources. With DMR models, authors can use powerful dimensional functions for data analysis, comparison, and manipulation.

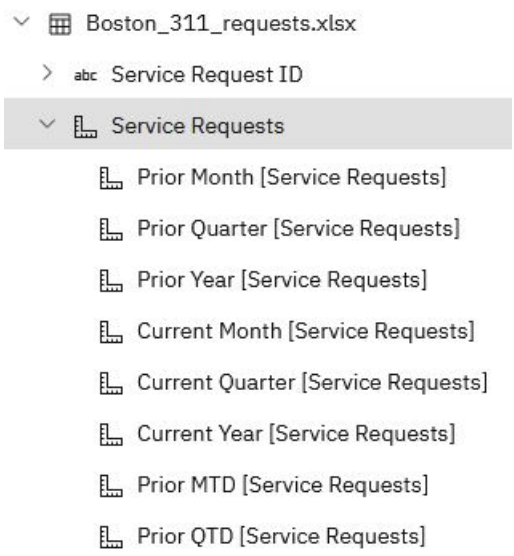
By using DMR, you can easily perform relative time analysis by dragging the members that you want to compare into the query, or use dimensional functions to retrieve the members for comparison. DMR also allows you to navigate up and down through the hierarchies of the dimensions, which is called drilling up and down. This makes for quick and powerful analysis of the data with the ability to drill down into the details.

For more information, see "Defining the Dimensional Representation of the Model" in the *IBM Cognos Framework Manager User Guide*.

In data modules, relative date analysis is accomplished in a purely relational fashion by constructing relative date filters and relative date measures. The following screen capture shows the relative date filters for the `Open Date` column in a sample data module:



And the following screen capture shows the relative date measures for the `Service Requests` measure in the same sample data module:



For more information, see "Relative date analysis" in the *IBM Cognos Analytics Data Modeling Guide*.

Data modules also allow you to navigate defined levels in your data using the navigation path feature. You can define logically related levels, such as Year, Month, Date, or non-related levels, such as Product, State, Customer, and so on. When navigation paths are enabled, users can drill up and down through the data in dashboards or explorations.

For more information, see "Creating navigation paths" in the *IBM Cognos Analytics Data Modeling Guide*.

Minimized SQL versus preventing join elimination

When a table is designed in the modeling tools, it can comprise multiple columns from multiple tables. When you query one column from this modeled table, you might expect to see SQL that omits the tables from which columns are not referenced. This concept is referred to as join elimination or minimized SQL.

Depending on the model design, minimized SQL might not occur, and the query acts like a view. In this case, all the joins of the underlying tables are enforced in a subquery before the parent query selects the single column in the final projection list. In some cases, this behavior might be desired because you want to enforce a join structure that controls the amount and type of data returned. This view behavior is also known as preventing join elimination.

For example, take the case of a `Product` dimension which includes four underlying tables, `Product dim` and three lookup tables that have a relationship to `Product dim`: `Product line lookup`, `Product type lookup`, and `Product lookup`. When you query `Product line` from `Product line lookup`, you might want only product lines returned, where there are also product types and products. In this case, you must ensure that the underlying join between these tables is enforced, provided inner joins are defined in the model. However, in some cases when you query `Product line`, you want to see all of the tables, regardless if they have product types and products associated. In this case, there are two ways to accomplish this requirement. The first is to configure a left outer join to `Product type lookup` from `Product line lookup`, with all the underlying joins still enforced in a subquery. The second option is to ensure that minimized SQL is generated by configuring the model to do so.

Let's look at the difference in the SQL for the `Product line` query. The following example shows non-minimized SQL (no join elimination).

```
WITH
"PRODUCT_LOOKUP0" AS
    (
    SELECT
        "PRODUCT_LOOKUP01"."PRODUCT_NUMBER" AS "PRODUCT_NUMBER",
        "PRODUCT_LOOKUP01"."PRODUCT_LANGUAGE" AS "PRODUCT_LANGUAGE",
```

```
        "PRODUCT_LOOKUP01"."PRODUCT_NAME" AS "PRODUCT_NAME",
        "PRODUCT_LOOKUP01"."PRODUCT_DESCRIPTION" AS "PRODUCT_DESCRIPTION"
    FROM
        "PRODUCT_LOOKUP" "PRODUCT_LOOKUP01"
    WHERE
        "PRODUCT_LOOKUP01"."PRODUCT_LANGUAGE" IN (
            'EN' )
    ),
         "Product_Line_Lookup_View_1" AS
    (
    SELECT
        "PRODUCT_LINE_LOOKUP0"."PRODUCT_LINE_EN" AS "PRODUCT_LINE_EN"
    FROM
        "PRODUCT_LINE_LOOKUP" "PRODUCT_LINE_LOOKUP0"
            INNER JOIN "PRODUCT_DIM" "PRODUCT_DIM0"
            ON "PRODUCT_LINE_LOOKUP0"."PRODUCT_LINE_CODE" = "PRODUCT_DIM0"."PRODUCT_LINE_CODE"
                INNER JOIN "PRODUCT_LOOKUP0"
                ON "PRODUCT_LOOKUP0"."PRODUCT_NUMBER" = "PRODUCT_DIM0"."PRODUCT_NUMBER"
                    INNER JOIN "PRODUCT_TYPE_LOOKUP" "PRODUCT_TYPE_LOOKUP0"
                    ON "PRODUCT_TYPE_LOOKUP0"."PRODUCT_TYPE_CODE" =
"PRODUCT_DIM0"."PRODUCT_TYPE_CODE"
    )
SELECT
    "Product_Line_Lookup_View_1"."PRODUCT_LINE_EN" AS "Product_Line"
FROM
    "Product_Line_Lookup_View_1"
GROUP BY
    "Product_Line_Lookup_View_1"."PRODUCT_LINE_EN"
```

Notice all the inner joins in the sub query for `Product_Line_Lookup_View_1`. The joins for all four tables are enforced.

Compare this SQL with the following example of minimized SQL (joins are eliminated) for the `Product line` query:

```
SELECT
    "Product_Line_Lookup_View_1"."PRODUCT_LINE_EN" AS "Product_Line"
FROM
    "PRODUCT_LINE_LOOKUP" "Product_Line_Lookup_View_1"
GROUP BY
    "Product_Line_Lookup_View_1"."PRODUCT_LINE_EN"
```

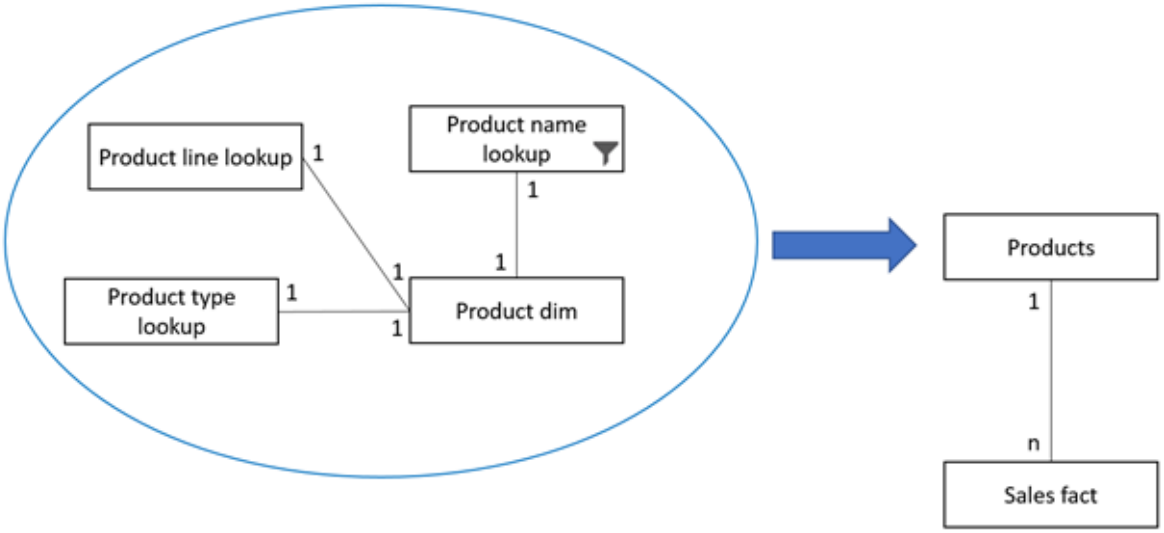
In this case, it's a simple column selection from a table.

If you then added an item from `Sales fact` to the query, the appropriate underlying joins from `Product line lookup` to `Product dim` to `Sales fact` would be used to aggregate the values from the `Sales fact` table.

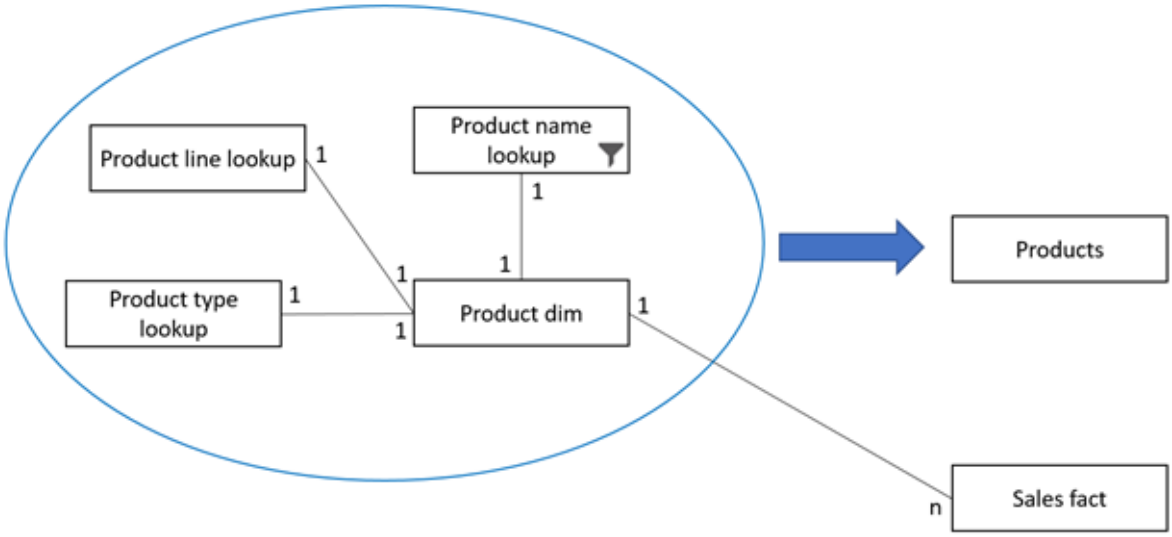
Minimized SQL in Framework Manager

In Framework Manager, you can configure a query subject to use minimized SQL, which is the default setting. For more information, see "Changing how the SQL is generated" in the *IBM Cognos Framework Manager User Guide*.

However, even if minimized SQL is configured in Framework Manager, if you create a model query subject, and then attach a relationship join to it, the model query subject always acts as a view. The joins are enforced as shown in the following model design scenario, which results in non-minimized SQL:



To generate minimized SQL, as shown in the following scenario, you wouldn't join from `Products` to `Sales fact`. Instead, you would join `Product dim` from the underlying product tables to the `Sales fact` table. When querying a single column, such as `Product line`, from `Products`, the SQL would be minimized as expected.



If you then added an item from `Sales fact` to the query, the appropriate underlying joins from `Product line lookup` to `Product dim` to `Sales fact` would be used to aggregate the values from the `Sales fact` table.

Minimized SQL in data modules

In data modules, if you want the minimized SQL or view behavior, use the table property **Item list**. A consolidated table with a relationship join attached to it uses this property to generate minimized SQL, or enforce the underlying joins.

For more information, see "Generating the query SQL" in the *IBM Cognos Analytics Data modeling* guide.

Aggregation and the order of operations

In the modeling tools and query tools, you can perform a calculation and then aggregate the results, or aggregate the values in the calculation first and then perform the calculation.

To control this functionality, you must set the aggregation properties accordingly. To calculate first and then aggregate, set the column property to **Sum** or **Total** (depending on the user interface). To aggregate first and then calculate, set the property to **Calculated**. The **Calculated** setting applies only to stand-alone (selectable) calculations, which are created outside of a table in the modeling tools, and in the query tools, such as Dashboarding or Reporting.

The following table shows the different outcomes when calculating values before or after aggregation.

Table 9. Calculating values before and after aggregation

Row number	**A**	**B**	**A * B (set to Sum)**	**A*B (set to Calculated)**
1	5	10	50	50
2	10	5	50	50
Total	**15**	**15**	**100**	**225**

In the A*B (set to Sum) column, the calculation is performed first and then the values are summed (50+50=100). In the A*B (set to Calculated) column, the detail rows are aggregated first (15 for each total) and then the results are multiplied (15*15 = 225).

For more information, see the related documentation for the following components:

- Reporting: "Summary functions" in the *IBM Cognos Analytics - Reporting User Guide*.
- Data Modules: "Calculations" in the *IBM Cognos Analytics Data Modeling Guide*.
- Framework Manager: "Setting the Order of Operations for Model Calculations" in the *IBM Cognos Framework Manager User Guide*.

Chapter 7. Optimizing query performance

Cognos Analytics is designed to take full advantage of your data infrastructure. The primary data access strategy is to delegate data processing, as much as possible, to a data server.

Therefore, in typical scenarios the volume of data is bounded by your data server capacity to respond to analytical queries within the threshold of your users' wait time tolerance. Typically, users don't like waiting more than a few seconds for a request when interacting with data.

Cognos Analytics generates Structured Query Language (SQL) queries to retrieve data from relational data servers. Users must wait while the data server responds to such queries. For example, when connecting to the SQL interface of a data server, Cognos Analytics generates SQL that is tailored for the type and version of the data server technology, and optimized to minimize the user wait time.

Usually, the number of rows that need to be transferred from the data server to Cognos Analytics is equal to the number of values to be displayed within Cognos Analytics. Even if your data server stores billions of records, if a bar chart in a Cognos Analytics dashboard displays five bars, only five rows of data should be retrieved. The data server computes all joins, aggregations, calculations, filters, and so on, that result in the five values that get displayed in the Cognos Analytics visualization.

It's possible to make requests that cannot be processed by an underlying data server. These types of requests might require processing by the Cognos Analytics query service instead. While this is not always possible, Cognos Analytics is designed to avoid generating SQL statements that return a large number of rows where only a small percentage of the row data is presented to users. Although such SQL statements might not be complex or expensive for the data server to process, they can result in large amounts of data that is transferred to the Cognos Analytics server for local processing, which might increase the wait times.

There might be times when retrieving data live from a data server is not desirable. You can avoid waiting for data server processing by enabling data caching or using data sets.

Query times can be greatly impacted by aggregations, calculations, joins and other such operations that require data processing either by Cognos Analytics or a data server. Keep in mind that dashboards are typically presenting summary perspectives of data, which could be more detailed (fine grained) at the source. A good first step in dashboard performance troubleshooting is identifying widgets that are accessing millions of rows of stored data, which then undergoes processing to reduce the number of rows to a small number of values to be displayed in a visualization. Most of that processing time could be avoided if the final result of data to be presented to the user were preprocessed. To address this issue, many data server technologies offer a concept known as a materialized view, which denotes a precomputed result of one query that is accessible to other queries. The same concept might be known under different names.

Materialized views can reduce user wait times in Cognos Analytics by offering data that is preaggregated at the key, frequently requested intersections of logical aggregation. For more information, see "Comparing materialized views in data servers to data caching in Cognos Analytics" on page 30.

Data cache

Modelers can configure the default cache behavior per query (table).

If the requirement is to have the current data per query for volatile data, setting the default to no cache makes sense. If the data changes infrequently, once a day for example, it makes sense to set the default to use the cache.

In Framework Manager, the default data cache behavior can be specified by using a model governor. This behavior is inherited by all queries that are constructed based on this model. The model data cache setting can be overridden in other Cognos Analytics components, for example, in Reporting by the query property **Use local cache**.

For more information about setting data caching in Framework Manager, see "In-memory caching" in the *IBM Cognos Framework Manager User Guide*.

For information about setting data caching in data modules, see ."Setting up data caching" in the *IBM Cognos Analytics Data Modeling Guide*.

Data sets

Data sets are created by extracting data from packages or data modules. Data sets can be used to gather a customized collection of items that you use frequently. As you make updates to your data set, dashboards and stories that use the data set are also kept up to date the next time you open them.

You define a data set by choosing one or more items (columns) from a package or data module, and apply filters to reduce the data. You're essentially specifying the rectangle of columns and rows of data that you need. The data is extracted and stored within the Cognos Analytics system.

Data sets can improve query performance and reduce the workload on your databases. The following are some reasons for using data sets:

- Improve query performance if your database is slow.
- Reduce the load on an overworked database (especially during peak periods).
- Retain a version of the data at a specific time.

For data sets created from relational packages or data modules, you have the option **Summarize detailed values, suppressing duplicates**. When you use this option, measure values are aggregated to the lowest grain that is explicitly included in the data set. For example, your data warehouse stores millions of records pertaining to each transaction where units were sold, but you're only interested in analyzing the total sales per region. If your data set contains only the `Region` and `Units Sold` columns, and you use this option, the data set will contain only as many rows as there are regions.

For more information, see "Data sets" in the *IBM Cognos Analytics Getting Started Guide*, and "Best practices for improving query performance on uploaded files" in the *IBM Cognos Analytics Managing Guide*.

Comparing materialized views in data servers to data caching in Cognos Analytics

Each instance of an IBM Cognos Analytics query service manages its own private cache of reusable result sets from SQL queries. Hence, a cluster of several application tier servers manages its own caches and doesn't share them. Meanwhile, queries that are sent from any of those instances can benefit from the materialized views a data server might use.

The query service holds result sets in a data cache that reflect the columns a Cognos Analytics user is requesting, with the level of aggregation and filtering applied. The order in which queries are run influences if any cached data can be reused or not. For example, a query projects COUNTRY and SUM(SALES) from T(table) where COUNTRY=UK. The result set of the query holds only the total sales for UK. If a new query is processed where COUNTRY=FR, the first result set cannot be used because the data cache doesn't cover (include) COUNTRY=FR. Hence, as users change context in a dashboard, or different queries request similar groupings but different measures, the data caches might not be able to avoid requerying the data server.

Contrast this with a materialized view in a data server. The materialized view would be defined to hold the grouped rows for all known countries. Hence, when COUNTRY=UK, it would quickly locate that row, likewise when COUNTRY=FR. It could also be used to compute SELECT SUM(SALES) from T. In effect, the materialized view could span a broader set of dimension categories and measures that service many different needs of the Cognos Analytics users.

The creation of a materialized view requires a database administrator (DBA) to define and deploy the materialized view. Using the data cache in Cognos Analytics doesn't require a DBA. If a DBA is not available to implement materialized views, relying on Cognos Analytics data caching or data sets, which have prejoined, filtered, and aggregated data, might be your only solution to consider.

Minimizing SQL query response times

Remember that less is faster. If all other factors are the same, a simpler SQL statement is satisfied in less time than a more complex SQL statement. Likewise, requests for more data take longer than requests for less data, if all other factors are equal.

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

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

Here are some preferred practices that many data server technology vendors suggest to improve run-time performance.

Avoid complex join and filter expressions

The complexity of expressions in the `WHERE` and `JOIN ON` clauses of an SQL statement can impede planning for the data server, query rewrites to materialized views, or other forms of query acceleration.

Reduce explicit or implicit data type conversions

Converting data types requires processing that can significantly increase query response times. Converting data types happens explicitly when you use a function like `CAST()` in a calculation or filter, or implicitly when certain operations occur on columns with different data types.

Ideally, a calculation expression that serves as a key in a relationship between tables resolves to the same data type as the corresponding key on the opposite side of the join relationship. This prevents constraining the data server from considering certain join strategies, such as a hash join, because of incompatible data types.

Avoid using SQL expressions to transpose values

Users who are familiar with SQL can construct expressions that attempt to massage database values for display. In several cases, such expressions can be replaced by using the available data type formatting, layout, and other presentation options.

The following example demonstrates how you can initiate multiple data type conversions, substrings, and concatenations to display a date value in a particular way rather than using the data format rendering option available in various authoring interfaces.

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

Avoid unnecessary outer joins

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

Make use of constraints on tables in data servers

Tables in a data server can have constraints that can be considered by the data server query engine for strategies such as join eliminations, query rewrites, and expression optimizations.

Primary key, unique key, and foreign key constraints (but not null and table constraints) can be declared for this purpose. Depending on the data server technology, these constraints can be declared as either non-enforced or enforced. In a normalized table design that includes snowflake schemas, non-primary key columns are functionally dependent on the primary key.

To plan SQL statements for the data server to process, the Cognos Analytics query service uses enforced constraints defined in a metadata model, such as determinants in Framework Manger or column dependencies in data modules, and relationships between tables. These objects are often created during one of the initial steps of creating a model, but more common is that they are manually defined by the metadata modeler. For more information, see "Determinants" in the *IBM Cognos Framework Manager User Guide* and "Column dependencies" in the *IBM Cognos Analytics Data Modeling Guide*.

Use indexes and table organization features

A common challenge for a database administrator is to anticipate the ways that applications attempt to navigate the database. This includes which tables the queries will combine and which tables predicates will be applied against.

Using a representative workload, the database administrator can review which tables are most frequently accessed, and in particular, which local set of columns is used to filter and group columns in tables. Using that knowledge, the database administrator can usually identify indexes or table organization strategies that enable the database to more efficiently select the required rows.

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

A metadata model can also be constructed on top of databases that expose application objects through SQL views. Such views must be reviewed by the database administrator with respect to the expressions within the view.

Join performance for heterogeneous data sources

To improve performance when querying heterogeneous data sources in IBM Cognos Analytics, you can specify join filters in data modules, Framework Manager, or Reporting.

By using this feature, you can push predicates (the filter values) from the table on the 1 side of the query into the filter expression of the SQL of the table on the n side of the query. For information about setting the join filters, see the related component documentation.

For information about setting the join filters in the different Cognos Analytics components, see the following documentation:

- Data modules: "Join optimization" in the *IBM Cognos Analytics Data Modeling Guide*
- Framework Manager: Optimizing joins by applying filters"Optimizing joins by applying filters" in the *IBM Cognos Framework Manager User Guide*
- Reporting: "Create a Join Relationship" in the *IBM Cognos Analytics - Reporting User Guide*

There are some additional considerations when working with multiple data sources. For more in-depth information on this subject, and to learn how to improve performance, see this article (www.ibm.com/support/pages/node/6258353).

Chapter 8. Data server changes and switching database vendors

When designing and maintaining metadata models, the modeler must consider changes to the underlying databases or the requirements to change the database vendors altogether.

Each modeling tool provides a way of updating changed metadata. For example, some column names in the database might change, or columns might be added or removed. You want to ensure that such changes are reflected in the model so that errors don't occur.

In more extreme cases, you might need to connect to or switch between multiple database vendors for one model. In these cases, ensure that the database structure is identical, including the following elements:

- Case sensitivity
- Matching names of schemas, tables, or columns
- Compatibles data types

For example, DB2 has the DATE data type which is not compatible with the TIMESTAMP data type in Oracle. The cleaner the match in structure and data type is, the smoother the vendor switch will be. Switching vendors is not an exercise to be taken lightly. Appropriate planning should be in place before implementing.

Framework Manager and data modules require slightly different methods for database updates and vendor switching.

Data modules

After you reload the schema metadata for the data server source, you can re-add tables to the data module to include changes from the database. If a new column is added, it appears in the data module. If a column is missing, the validation process flags an issue, and you can delete the column from the module. For more information, see "Updating columns in a data module" and "Reloading schema metadata" in the *IBM Cognos Analytics Data Modeling Guide*.

Use the **Show unused items** feature on the data module source to see newly introduced columns in a table. This feature is also useful when you remove columns from a table in the data module, and then want to add them back. In this case, the columns are highlighted in the **Sources** panel

If a table name changes, import the new table and delete the old one. However, any table view that was based on the old table needs to be recreated.

When you need to replace the data sources altogether, you can do so by relinking the source. For more information, see "Relinking sources" in the *IBM Cognos Analytics Data Modeling Guide*.

As with Framework Manager, the database structure, naming conventions, and data types must match between the old and relinked source.

Framework Manager

When a table column is changed, or a column is removed or added, you can update the data source query subject to reflect the change. For more information, see "Updating query subjects" in the *IBM Cognos Framework Manager User Guide*.

Once this happens, any model query subjects that are dependent on the data source query subject must also be updated to reflect changes. New columns can be added manually.

When a table name changes, you need to import the table as a new object, delete the old one, and then remap any dependent model query subjects to the newly imported table. For more information, see "Remapping objects to new sources" in the *IBM Cognos Framework Manager User Guide*.

If you plan to change database vendors or allow dynamic access to different vendors by using macros at run time, ensure that correct values are specified for the data source properties in the model. For example, the Content Manager data source name must match a data source that is configured in Cognos Analytics, the data source name must point to the intended database vendor, and the catalog and schema properties must be set up correctly. For more information, see "Data sources" in the *IBM Cognos Framework Manager User Guide*.

When the model was built using a business layer approach to insulate reports from changes to the database, and when there are data type or naming convention conflicts, you can import the new tables and remap the dependent model query subjects to them. Then, delete the old data source query subjects.