Modeling in Framework Manager for Predictable Queries and Results

Copyright

Cognos Enterprise Business Intelligence

This edition published April 2005

Table of Contents

Workflow for Modeling Relational Data Sources

This document discusses the items that are in bold. For all other items, see the Framework Manager User Guide for details.

- Import Metadata
- **Verify Imported Metadata**
  - **Relationships and Cardinality**
  - Usage Property
  - Regular Aggregate Property
- Customize Metadata
  - Multilingual/dynamic metadata
  - Broadly applied row level security filters
- **Simplify Model with Dimensional Concepts**
  - **Collapse hierarchical and master/detail relationships**
  - **Analyze and resolve usage of query subjects using rules of cardinality**
  - Resolve fact to fact relationships – you should handle these in the data source
- **Resolve Ambiguous Relationships**
  - **Role-playing dimensions**
  - **Recursive relationships**
  - Modeling multiple discrete join paths – this is still under investigation
- **Define Dimensional Information**
  - **Multi-Fact/Multi-Grain queries**
  - **Internal cardinality**
  - **Multiple Hierarchies**
- Organize the Model by Creating Business Views
- **Create Star Schema Groupings**
  - **Conformed dimensions**
  - **Query context**
- Apply Security
- Create Packages and Publish Metadata

All scenarios described in the following cases are produced using the database view of the gosales_goretailers normalized sample model, which is included with the Cognos ReportNet samples.

Introduction

Framework Manager is the metadata modeling tool that delivers the BI metadata to business intelligence reporting, analysis, and scorecarding applications. A key goal of metadata for business intelligence is to leverage existing metadata while adding value in order to provide an intuitive end user experience characterized by predictable queries and results.

This document addresses some fundamental concepts for modeling relational data sources in Framework Manager and outlines best practices for modeling metadata for use in business reporting with Cognos ReportNet. Cognos ReportNet enables reporting on transactional and dimensional relational sources as well as SAP BW. Framework Manager has many features that are designed to leverage a dimensionally designed relational data source. While a dimensional schema is not required to use Framework Manager or Cognos ReportNet, many features have been added to assist metadata modelers and report authors to leverage the benefits of dimensional schemas.

Verify Relationships and Cardinality

Cardinality affects how queries are written and thus affects the results of a query. Cognos ReportNet allows cardinality to be specified by the modeler or generated based on criteria in the database.

Detect Cardinality from the Data Source

When importing from a relational data source, cardinality is detected based on a set of rules specified by the modeler.

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 the first two options: primary and foreign keys as well as query items that represent uniquely indexed columns. When you use these options to import your metadata and generate relationships, remember that you are adding this information to your model (this is important to note for later sections of this paper).

Note: Optional relationships, full outer joins, and many-to-many relationships can be imported from your database and are supported for query execution. Recursive relationships can be imported from your database and will appear in the model but they are not supported for query execution.

Usage of Cardinality in Query Generation

Framework Manager Modeling Guidelines:

The general rule applied to cardinality is as follows:

1-to-n cardinality implies FACT

0-to-n or 1-to-n cardinality implies DIMENSION

These rules are applied in the scope w\of what is included in any given query.

Framework Manager supports both minimum/maximum cardinality as well as mandatory/optional. In cases where minimum/maximum cardinality is defined (as recommended by Cognos's best practices) it will be applied to identify facts and dimensions.

The basic rules that apply to the usage of cardinality are:

- Cardinality is applied in the context of a query -- only the cardinalities of items explicitly included in the query are evaluated.
- Query subjects defined with 1-to-many or 0-to-many cardinalities are identified as facts.
- Query subjects defined with 1-to-1 or 0-to-1 cardinalities are identified as dimensions.
- It is possible to have a query subject behave as a dimension in one query and as a fact in another.

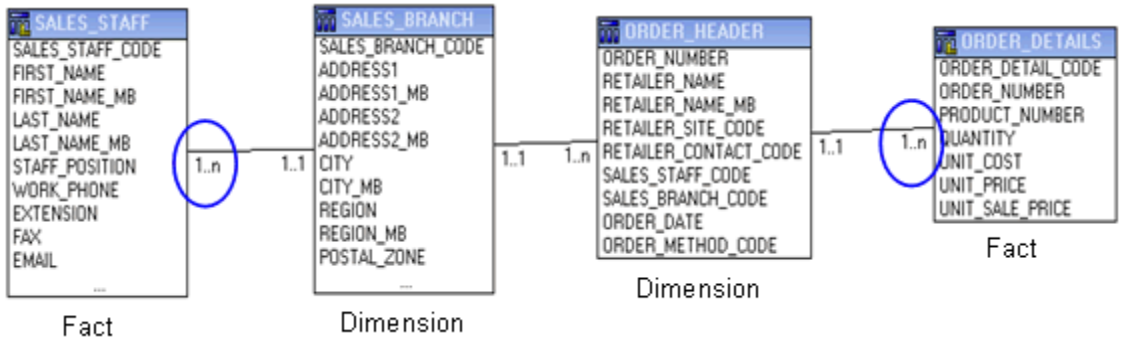
Note: When only mandatory/optional cardinality is employed, SQL generated will not be able to compensate for double counting that can occur when dealing with hierarchical relationships and different levels of granularity.

Identifying Facts and Dimensions

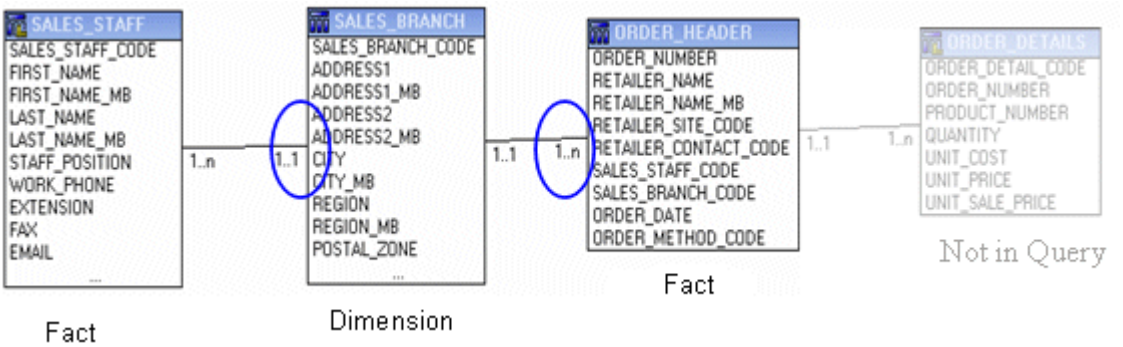
Within the context of a query, a query subject that has only maximum (n) cardinality on all its relationships to other query subjects can be regarded as a fact. The implication is that there are more rows of data in the fact query subject than in the related query subject on the minimum (1) side of the relationship. Any query subject having at least one relationship to another query subject with minimum cardinality (1) will be treated as a dimension.

Analyze a Schema for Facts and Dimensions

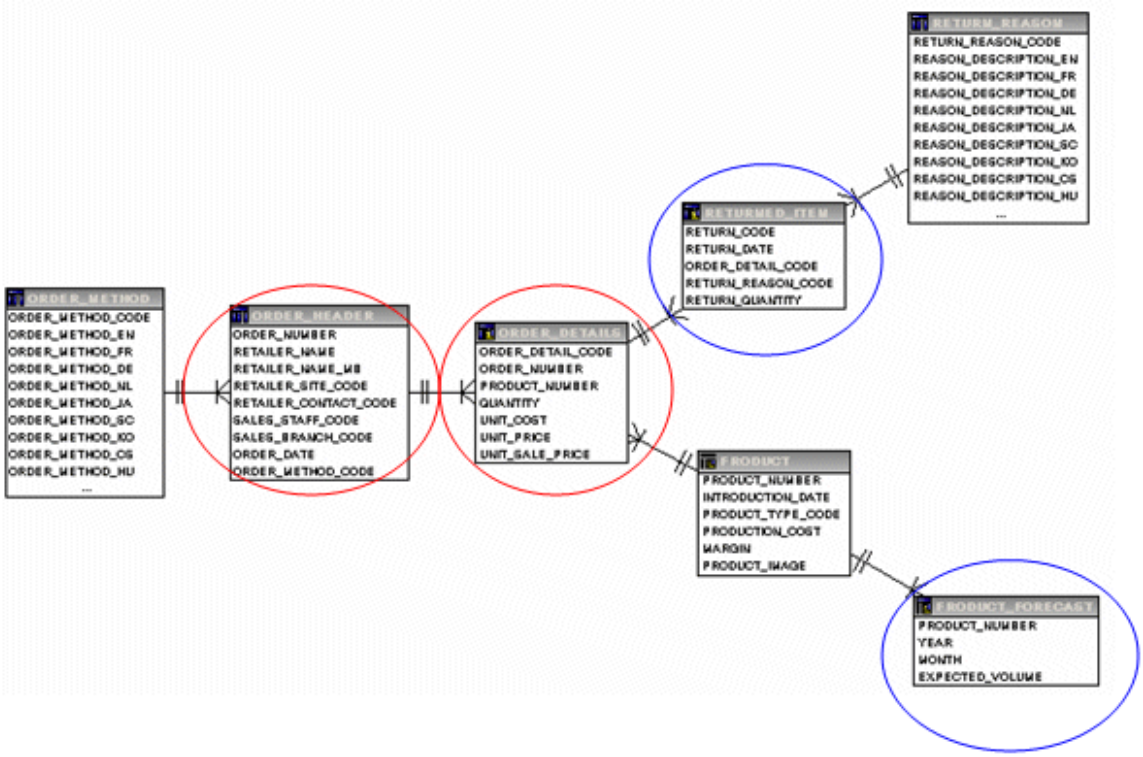
Example 1: In this example, all four query subjects are included in a query. The diagram shows that the query subjects having only maximum (n) cardinalities are treated as facts.



Example 2: In this example, only three query subjects are included in the query. The diagram shows that the Order Header query subject is now treated as a fact.



Example 3: In this example, query subjects whose cardinality indicates that they are always facts are circled in blue. Areas where the behavior is dependent on the context of the query are circled in red. All query subjects that are not circled behave as dimensions in all cases.



Tip: As part of the process of verifying the imported relationships, you should analyze sections of the import view of your model to identify areas where a query subject's role will change depending on the context of the query it is included in.

Simplify the Model with Dimensional Concepts

Framework Manager Modeling Guidelines

Use the cardinality rules from the previous section to identify areas of the model that ambiguously identify dimensions or facts.

Collapse groups of query subjects with hierarchical relationships into a single query subject per business concept (typically applies to dimensions).

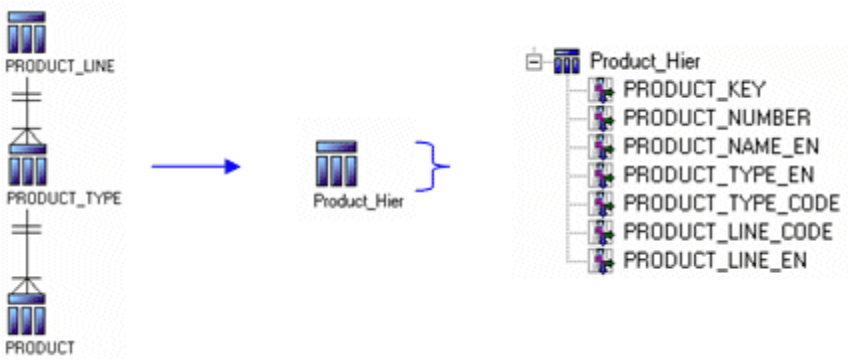
Collapse groups of query subjects with master/detail relationships into a single query subject per business concept (typically applies to facts).

The end result of simplifying the model should be a layer of query subjects that clearly represent the data in terms of business concepts and ensure predictable query generation.

Collapse Hierarchical Relationships

There are often cases in normalized or snowflaked data sources where several tables exist to describe a single business concept. Product is a very common example of this. For example, a normalized representation of Product includes three tables related by one to many relationships. Each Product Line has one or more Product Types. Each Product Type has one or more Products.

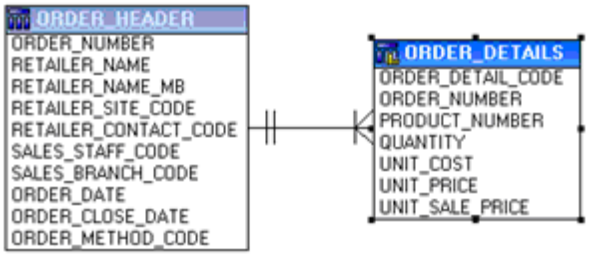
An end user may or may not be knowledgeable about the relationship between the individual levels of product, also the necessity to expand and select a query item from each query subject in the model makes more clicks for the end user. A modeler can make the decision to create a model query subject for product that not only simplifies using Product for the purpose of ad hoc query and reporting but also presents the levels of the hierarchy in order to visually cue the end user on the relationship between the levels. If desired, the modeler can use query item folders to clarify which attributes apply to each level. In the example below further clarification is not required.



Collapse Master/Detail Relationships

Another common scenario in data sources is the existence of master/detail tables containing facts. A good example of this is order header and order detail. For the purposes of inserting and updating data, this structure is beneficial. For reporting purposes, however, the concept of Orders exists separately from the physical representation of the data and, as discussed in the previous section, this scenario can also lead to unpredictability in query generation.

In order to simplify the model in this case, it is better to create a model query subject that combines the foreign keys of both Order Header and Order Details and includes all measures at the Order Detail level.



You should resolve ambiguously identified dimensions and facts. For more information and examples, see Appendix A.

Resolve Ambiguous Relationships

Framework Manager Modeling Guidelines

Cases of multiple relationships or recursive relationships generally imply that the data represented by a query subject can be viewed in more than one context or role and should be modeled accordingly. Two common examples of ambiguous relationships are:

Multiple Valid Relationships –typically occur between dimensions and facts. Create a shortcut or model query subject for each role with a single relationship specific to the role.

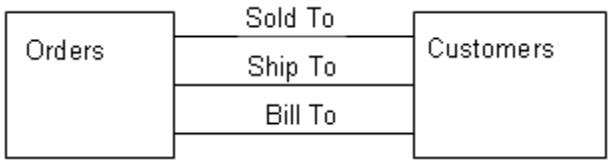
Recursive Relationships – typically imply two or more levels of granularity. As a minimum create a shortcut or model query subject each for the parent and the child roles. For a deep recursive hierarchy, we recommend that the hierarchy be flattened in the database and that you model the flattened hierarchy in a single query subject.

An ambiguous relationship occurs where there are multiple relationship paths between one or more query subjects leaving multiple options for how to write a query. This is a situation that can be responded to by the modeler or in the database by the DBA. If appropriate, the DBA can choose to remove extra relationships (although this can potentially result in another situation that will need to be addressed). Dealing with this situation in Framework Manager is a matter of determining which query path you want used.

Join ambiguity in Framework Manager can be resolved by using shortcuts or model query subjects as well as by organizing the model with folders or namespaces to specify the scope of joins. Use shortcuts when you want exact replicas of a query subject in more than one place. Use model query subjects when you want to create custom views of the same query subject. Scope causes the query engine to evaluate the joins on the basis of proximity to the query subject. (that is, first look for joins that exist in the same folder or namespace).

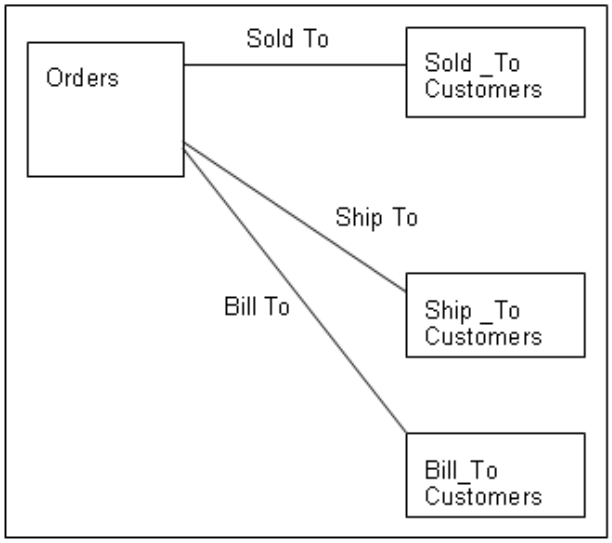
Multiple Valid Relationships

A table that has multiple valid relationships between itself and another table is known as a role-playing dimension. This is most commonly seen in dimensions such as Date and Customer. A good example is an Orders fact, which may have multiple relationships to the Customer dimension on keys such as sold_to, ship_to and bill_to.



How do you resolve the join ambiguity for Customer?

The answer in Framework Manager is part of the best practice for star schemas. Leave all the relationships in place in the Import View. Create a model query subject for each role. Rename the model query subjects and query items appropriately for their use. Ensure that a single appropriate relationship exists between each model query subject and the fact query subject.



This could also be done with shortcuts but renaming would only be possible at the query subject level. All query items would be identical for each shortcut.

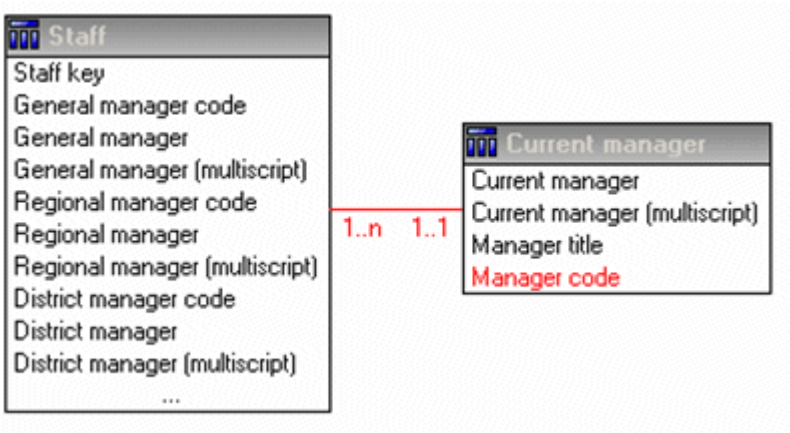
Recursive Relationships

Framework Manager imports recursive relationships but does not use them when executing queries. Recursive relationships are shown in the model for the purpose of representation only. To create a functioning recursive relationship, you must create a shortcut to the query subject or a model query subject based on the query subject (an alias) and then create a relationship between the query subject and the alias. It is necessary to repeat this for each level you wish to represent.

For a simple two-level structure, the model would look like this:

For each additional level it would be necessary to create another alias and join it appropriately.

Using a model query subject is often a more attractive option because you can specify which query items are included in the query subject. For example, the Staff query subject has a recursive relationship between Sales Staff Code and Manager Code. By creating a model query subject to represent Manager, you are able to select which query items apply to Manager and rename them in a meaningful way. Create a relationship with a 1..1, 1..n relationship between Staff and Manager.



Note: When dealing with a recursive relationship that represents a deep recursive hierarchy, we recommend that the hierarchy be flattened in the database and the flattened table be modeled as a single query subject.

Understand Dimensionally Optimized Queries

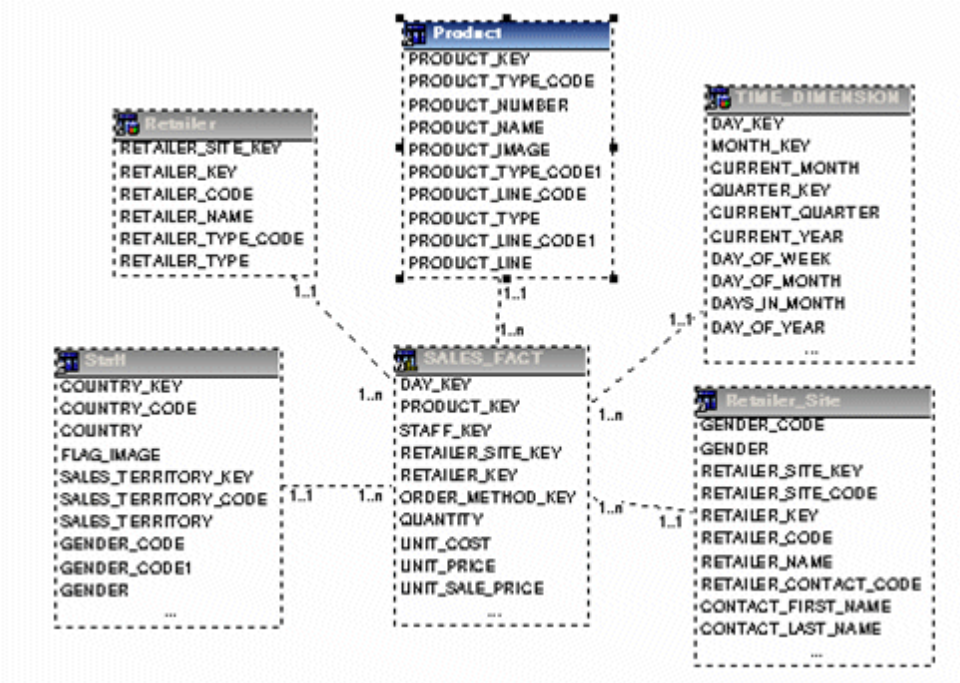
Dimensionally optimized queries are designed to enable multi-fact querying. The basic goals of multi-fact querying are:

Preserve data in cases where fact data does not align one hundred percent across common dimensions.

Prevent double-counting where fact data exists at different levels of granularity by ensuring each fact is represented in a single query with appropriate grouping.(Note: May require dimension information to be specified in some cases)

Single Fact Query

Example 1: A query on a single star schema grouping results in a single fact query.



In this example, Sales Fact is the focus of any query written, the dimensions providing attributes and descriptions to make the data in Sales Fact more meaningful. All relationships between dimensions and the fact are one-to-many. When you filter on the Month key and Product Name, the result is:

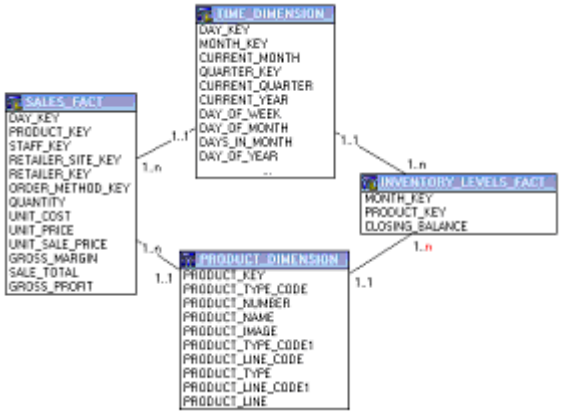
MONTH_KEY: between 200101 and 200103

PRODUCT_NAME_EN: Aloe Relief, Husky Rope 50

MONTH_KEY	PRODUCT_NAME_EN	QUANTITY
200101	Aloe Relief	448
200102	Aloe Relief	44
200103	Aloe Relief	764
Summary		**1,256**

Multi-fact/Multi-grain query on Conformed Dimensions

Example 2: Sales Fact and Inventory Levels Fact are both fact query subjects.



Note: This is a simplified representation, not an example of how this would appear in a model built with Cognos's best practices.

Observations:

- The data in the Sales Fact exists at a different level of granularity from the data in the Inventory Levels Fact. Sales is at the day level, Inventory Level is at the month level.
- In the same time period (200101-200103), there are no records retrieved for Husky Rope 50.

The Result

Individual queries on Sales Fact and Inventory Levels Fact by Month_key yield the following results. The data in Sales Fact is actually stored at the day level.

MONTH_KEY: between 200101 and 200103
PRODUCT_NAME_EN: Aloe Relief, Husky Rope 50

MONTH_KEY	PRODUCT_NAME_EN	QUANTITY
200101	Aloe Relief	448
200102	Aloe Relief	44
200103	Aloe Relief	764
Summary		**1,256**

MONTH_KEY: between 200101 and 200103
PRODUCT_NAME_EN: Aloe Relief, Husky Rope 50

MONTH_KEY	PRODUCT_NAME_EN	CLOSING_BALANCE
200101	Aloe Relief	12,664
200102	Aloe Relief	12,612
200103	Aloe Relief	11,908
200101	Husky Rope 50	9,914
200102	Husky Rope 50	9,776
200103	Husky Rope 50	9,298
Summary		**66,172**

A query on Sales Facts and Inventory Levels Facts will respect the cardinality between each fact table and its dimensions and write SQL to return all the rows from each fact table. The fact tables will be matched on their common keys (month and product), and where possible aggregated to the lowest common level of granularity (days will be rolled up to months). Nulls are often returned for this type of query because it is possible that a combination of dimensional elements in one fact table does not exist in the other.

MONTH_KEY: between 200101 and 200103

PRODUCT_NAME_EN: Aloe Relief, Husky Rope 50

MONTH_KEY	PRODUCT_NAME_EN	QUANTITY	CLOSING_BALANCE
200101	Aloe Relief	448	12,664
200102	Aloe Relief	44	12,612
200103	Aloe Relief	764	11,908
200101	Husky Rope 50		9,914
200102	Husky Rope 50		9,776
200103	Husky Rope 50		9,298
Summary		1,256	

This report shows the results from each fact table grouped by the conformed dimensions, month and product.

In this case, Husky Rope 50 was in inventory but was not sold between 200101 and 200103; for the same period Aloe Relief was in inventory and was sold.

The SQL

The SQL generated by Cognos ReportNet to produce this result is often misunderstood. The SQL is referred to as stitched SQL and uses two sub-queries, one for each star brought together by a full outer join on the common keys to preserve all dimensional members occurring on either side of the query.

The following example has been edited for length and is used as an example to capture the main features of stitched queries.

```
select
    coalesce(D2.MONTH_KEY,D3.MONTH_KEY) as MONTH_KEY,
    coalesce(D2.PRODUCT_NAME,D3.PRODUCT_NAME) as PRODUCT_NAME,
    D3.CLOSING_BALANCE as CLOSING_BALANCE,
    D2.QUANTITY as QUANTITY,
    RMIN(D2.QUANTITY1 order by coalesce(D2.PRODUCT_NAME,D3.PRODUCT_NAME) asc
    local) as QUANTITY1
from
    (select distinct
        TIME_DIMENSION.MONTH_KEY as MONTH_KEY,
        PRODUCT.PRODUCT_NAME as PRODUCT_NAME,
        INVENTORY_LEVELS_FACT.CLOSING_BALANCE as CLOSING_BALANCE
    from
        TIME_DIMENSION,
        PRODUCT,
        INVENTORY_LEVELS_FACT
    where
        (INVENTORY_LEVELS_FACT.MONTH_KEY = TIME_DIMENSION.MONTH_KEY) and
        (Product.PRODUCT_KEY = INVENTORY_LEVELS_FACT.PRODUCT_KEY)
    ) D3
full outer join
    (select
        TIME_DIMENSION.MONTH_KEY as MONTH_KEY,
        Product.PRODUCT_NAME as PRODUCT_NAME,
        XSUM(SALES_FACT.QUANTITY for
        TIME_DIMENSION.MONTH_KEY,Product.PRODUCT_NAME ) as QUANTITY
    from
        TIME_DIMENSION,
        PRODUCT,
        SALES_FACT
    where
        (TIME_DIMENSION.DAY_KEY = SALES_FACT.DAY_KEY) and
        (Product.PRODUCT_KEY = SALES_FACT.PRODUCT_KEY)
    group by
        TIME_DIMENSION.MONTH_KEY,
        Product.PRODUCT_NAME
    ) D2
on ((D3.MONTH_KEY = D2.MONTH_KEY) and (D3.PRODUCT_NAME = D2.PRODUCT_NAME))
```

What is coalesce?

Coalesce is simply an efficient means of dealing with query items from conformed dimensions. A coalesce is used to accept the first non-null value returned from either query subject. This 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 would give us results only if an item in inventory was sold. A right outer join would give us all the sales where the items were in inventory. A left outer join would give all the items in inventory that had sales. A full outer join is the only way to find out all of what was in inventory and what was sold.

Why is there an aggregate function for Quantity but not for Closing Balance?

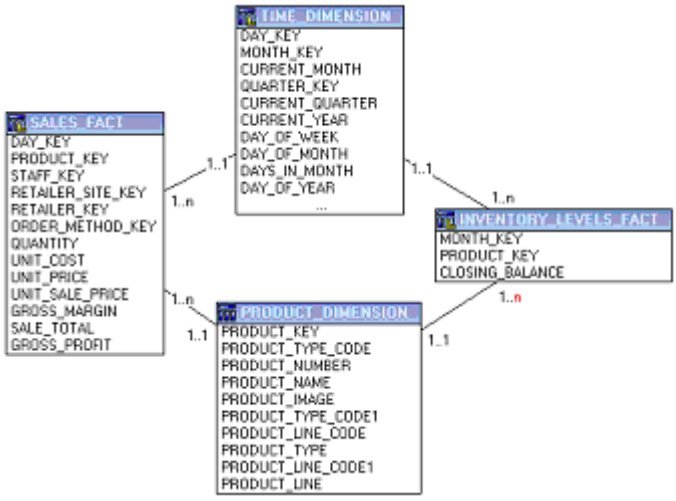
Because we have specified dimensional information on the conformed dimensions, Cognos ReportNet understands that the data in the inventory fact levels table is at a higher level than the data in Sales Fact table and that aggregation is required to bring the data in Sales Fact to a common level of granularity for reporting. If you turn off aggregation on the report, you can see the records reported at differing levels of granularity.

Why Use Dimensional Optimization?

When a Cognos ReportNet model is defined using minimum and maximum cardinality, it uses the cardinality of relationships to identify the facts and dimensions. In the example below there is relationship ambiguity and the potential for double counting.

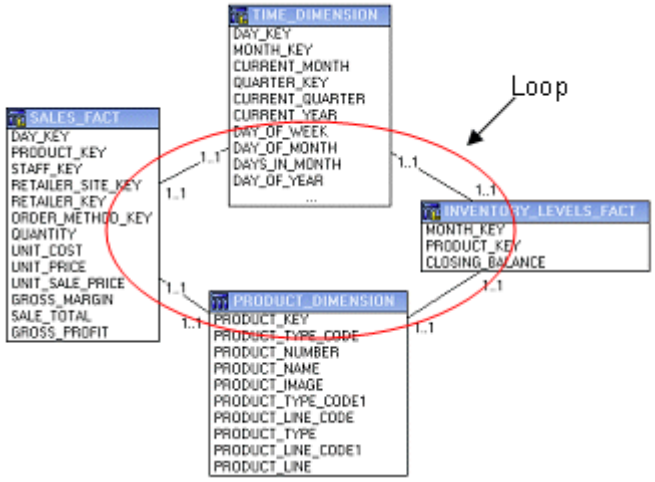
Relationship ambiguity exists when there is more than one join path between query subjects. For example, Time and Product could be joined through Sales Fact or Inventory Levels Fact with different results.

The potential for double-counting exists because the Inventory Levels Fact and the Sales Fact join to the Time Dimension on different keys (Month Key and Day Key respectively).



The result without minimum/maximum cardinality

If the cardinality were modified to use only mandatory (1) and/or optional (0) cardinality for all relationships between query subjects the result of a query on Inventory Levels Fact and Sales Fact with Time and/or Product will have incorrect totals. The Closing Balance is double-counted, once for each day in the month and the quantity is incorrect.



PRODUCT_NAME_EN: Aloe Relief, Husky Rope 50

MONTH_KEY: between 200101 and 200103

PRODUCT_NAME_EN	MONTH_KEY	QUANTITY	CLOSING_BALANCE
Aloe Relief	200101	458,366	120,915,872
Aloe Relief	200102	414,008	108,765,888
Aloe Relief	200103	458,366	113,697,584
Husky Rope 50	200101	228,594	69,150,150
Husky Rope 50	200102	206,472	61,588,800
Husky Rope 50	200103	228,594	64,853,550
Summary		**1,994,400**	**538,971,844**

The SQL

If you look at the SQL, you can see that Cognos ReportNet has detected that a loop join exists and has not included the relationship between time and sales because it was not necessary to complete the join path.

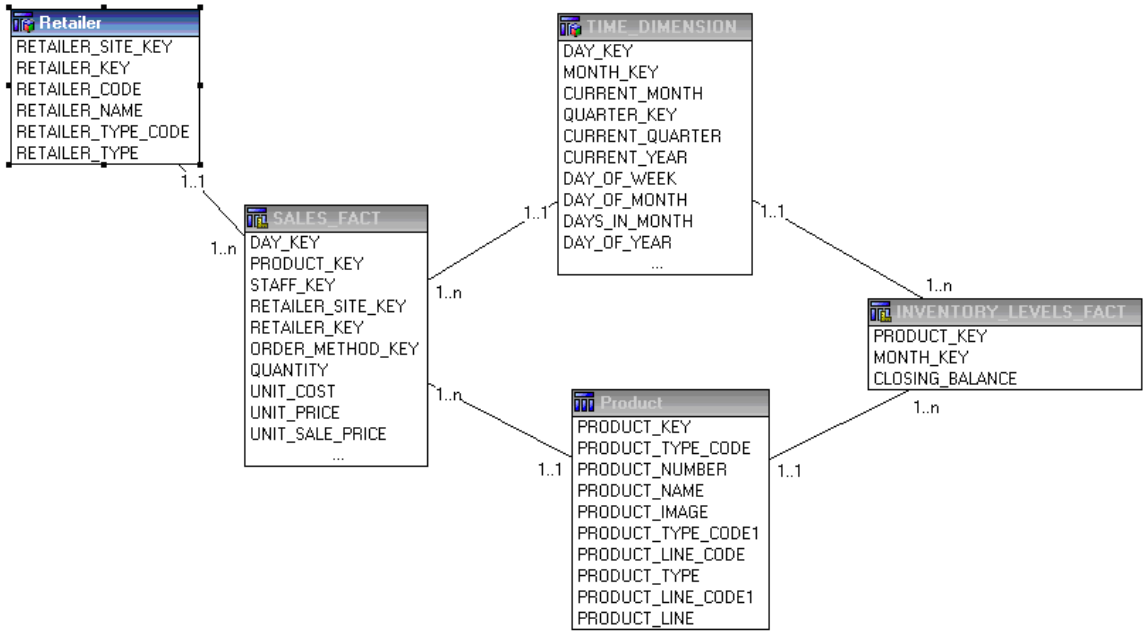
```
select
    PRODUCT_DIMENSION.PRODUCT_NAME_EN as PRODUCT_NAME_EN,
    TIME_DIMENSION.MONTH_KEY as MONTH_KEY,
    XSUM(SALES_FACT.QUANTITY for
    PRODUCT_DIMENSION.PRODUCT_NAME_EN,TIME_DIMENSION.MONTH_KEY ) as QUANTITY,
    XSUM(INVENTORY_LEVELS_FACT.CLOSING_BALANCE for
    PRODUCT_DIMENSION.PRODUCT_NAME_EN,TIME_DIMENSION.MONTH_KEY ) as
    CLOSING_BALANCE,
    XSUM(XSUM(SALES_FACT.QUANTITY for
    PRODUCT_DIMENSION.PRODUCT_NAME_EN,TIME_DIMENSION.MONTH_KEY ) at
    PRODUCT_DIMENSION.PRODUCT_NAME_EN,TIME_DIMENSION.MONTH_KEY ) as QUANTITY1,
    XSUM(XSUM(INVENTORY_LEVELS_FACT.CLOSING_BALANCE for
    PRODUCT_DIMENSION.PRODUCT_NAME_EN,TIME_DIMENSION.MONTH_KEY ) at
```

```
    PRODUCT_DIMENSION.PRODUCT_NAME_EN,TIME_DIMENSION.MONTH_KEY ) as
    CLOSING_BALANCE1
from
    "GO Sales DWH".GOSLDW.dbo.PRODUCT_DIMENSION PRODUCT_DIMENSION,
    "GO Sales DWH".GOSLDW.dbo.TIME_DIMENSION TIME_DIMENSION,
    "GO Sales DWH".GOSLDW.dbo.SALES_FACT SALES_FACT,
    "GO Sales DWH".GOSLDW.dbo.INVENTORY_LEVELS_FACT INVENTORY_LEVELS_FACT
where
    (PRODUCT_DIMENSION.PRODUCT_NAME_EN in ('Aloe Relief','Husky Rope 50')) and
    (TIME_DIMENSION.MONTH_KEY between 200101 and 200103) and
    (PRODUCT_DIMENSION.PRODUCT_KEY = INVENTORY_LEVELS_FACT.PRODUCT_KEY) and
    (TIME_DIMENSION.MONTH_KEY = INVENTORY_LEVELS_FACT.MONTH_KEY) and
    (PRODUCT_DIMENSION.PRODUCT_KEY = SALES_FACT.PRODUCT_KEY)
group by
    PRODUCT_DIMENSION.PRODUCT_NAME_EN,
    TIME_DIMENSION.MONTH_KEY
order by
    PRODUCT_NAME_EN asc,
    MONTH_KEY asc
```

Multi-fact/Multi-grain Query on Conformed and Non-Conformed Dimensions

Example 3:

The scenario where a non-conformed dimension is added to the query changes the nature of the result returned by the stitched query. It is no longer possible to aggregate records to a lowest common level of granularity as 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 would look like this.

MONTH_KEY	PRODUCT_NAME	RETAILER_TYPE	QUANTITY
200401	Aloe Relief	Sports Store	96
200403	Aloe Relief	Department Store	292
200403	Aloe Relief	Direct Marketing	52
200403	Aloe Relief	Outdoors Shop	556
200401	Aloe Relief	Outdoors Shop	308
200403	Aloe Relief	Warehouse Store	36
200402	Aloe Relief	Outdoors Shop	270

MONTH_KEY	PRODUCT_NAME	CLOSING_BALANCE
200401	Husky Rope 50	9,914
200402	Husky Rope 50	9,776
200403	Husky Rope 50	9,298
200401	Aloe Relief	12,664
200402	Aloe Relief	12,612
200403	Aloe Relief	11,908

Querying the same items from both star schemas yields the following result:

MONTH_KEY	PRODUCT_NAME	CLOSING_BALANCE	RETAILER_TYPE	QUANTITY
200401	Aloe Relief	12,664	Outdoors Shop	308
	Aloe Relief		Sports Store	96
	Husky Rope 50	9,914		
200401		**9,914**		**404**
200402	Aloe Relief	12,612	Outdoors Shop	270
	Husky Rope 50	9,776		
200402		**9,776**		**270**
200403	Aloe Relief	11,908	Department Store	292
	Aloe Relief		Direct Marketing	52
	Aloe Relief		Outdoors Shop	556
	Aloe Relief		Warehouse Store	36
	Husky Rope 50	9,298		
200403		**9,298**		**936**
Summary		**9,298**		**1,610**

In this result, the higher level of granularity for records from Sales Fact results in more records being returned for each month and product combination. There is now a 1:n relationship between the rows returned from Inventory Levels and Sales Fact. Cognos ReportNet understands this and does not double-count the rows from Inventory Levels.

By comparing this to the result returned in Example 2 you can see that the summaries are the same but more records are returned. (Note: Closing Balance is an end of period value and cannot be aggregated across time.) Also, it is no longer possible to relate one value from the Closing Balance item to one value from the Quantity item.

Grouping on the Month key demonstrates that the result in this example is based on the same data set as the result in Example 2 but with a greater degree of granularity.

The SQL

The stitched SQL generated for this example is very similar to that generated in Example 2. The main difference is the addition of Retailer Type. Retailer Type is not a conformed dimension and affects only the query against the Sales Fact table.

```
select
   coalesce(D2.MONTH_KEY,D3.MONTH_KEY) as MONTH_KEY,
   coalesce(D2.PRODUCT_NAME,D3.PRODUCT_NAME) as PRODUCT_NAME,
    D2.CLOSING_BALANCE as CLOSING_BALANCE,
   D3.RETAILER_TYPE as RETAILER_TYPE,
   D3.QUANTITY as QUANTITY
from
   (select TIME_DIMENSION.MONTH_KEY as MONTH_KEY,
      Product.PRODUCT_NAME as PRODUCT_NAME,
       Retailer.RETAILER_TYPE1 as RETAILER_TYPE,
      XSUM(SALES_FACT.QUANTITY for TIME_DIMENSION.MONTH_KEY,
      Product.PRODUCT_NAME,Retailer.RETAILER_TYPE1 ) as QUANTITY,
      RSUM(1 at TIME_DIMENSION.MONTH_KEY ,Product.PRODUCT_NAME,
      Retailer.RETAILER_TYPE1 for TIME_DIMENSION.MONTH_KEY,
      Product.PRODUCT_NAME order by TIME_DIMENSION.MONTH_KEY
      asc,Product.PRODUCT_NAME asc, Retailer.RETAILER_TYPE1 asc local) as sc
   from TIME_DIMENSION,Product,Retailer,SALES_FACT
   where
      (TIME_DIMENSION.DAY_KEY = SALES_FACT.DAY_KEY) and
      (Product.PRODUCT_KEY = SALES_FACT.PRODUCT_KEY) and
       (Retailer.RETAILER_KEY1 = SALES_FACT.RETAILER_KEY)
   group by
      TIME_DIMENSION.MONTH_KEY,
      Product.PRODUCT_NAME,
      Retailer.RETAILER_TYPE1
   ) D3
full outer join
   (select TIME_DIMENSION.MONTH_KEY as MONTH_KEY,
      Product.PRODUCT_NAME as PRODUCT_NAME,
      INVENTORY_LEVELS_FACT.CLOSING_BALANCE as CLOSING_BALANCE,
      RSUM(1 at TIME_DIMENSION.MONTH_KEY,
      Product.PRODUCT_NAME,INVENTORY_LEVELS_FACT.CLOSING_BALANCE for
      TIME_DIMENSION.MONTH_KEY, Product.PRODUCT_NAME order by
      TIME_DIMENSION.MONTH_KEY asc, Product.PRODUCT_NAME asc,
      INVENTORY_LEVELS_FACT.CLOSING_BALANCE asc local) as sc
   from TIME_DIMENSION, Product, INVENTORY_LEVELS_FACT
   where (INVENTORY_LEVELS_FACT.MONTH_KEY = TIME_DIMENSION.MONTH_KEY) and
      (Product.PRODUCT_KEY = INVENTORY_LEVELS_FACT.PRODUCT_KEY)
   group by
      TIME_DIMENSION.MONTH_KEY,
      Product.PRODUCT_NAME,
      INVENTORY_LEVELS_FACT.CLOSING_BALANCE
   ) D2
on (((D3.MONTH_KEY = D2.MONTH_KEY) and (D3.PRODUCT_NAME = D2.PRODUCT_NAME)) and
(D3.sc = D2.sc))
```

No coalesce because this is not a conformed dimension.

Grouping includes retailer type as well as month and product.

Define Dimensional Information

Framework Manager Modeling Guidelines

The primary purpose of dimensional information in Framework Manager is to prevent double-counting.

By specifying the internal cardinality or granularity of the data in a query subject, it is possible to enhance the model and allow the query engine to anticipate situations in which double-counting could occur.

When specifying dimensional information consider the following:

Identify the levels that are relevant for reporting purposes

Identify the key(s) for those levels and if those keys uniquely identify the members of their respective levels.

Identify any attributes associated to levels

Identify the hierarchy of the levels – the relationship between levels.

Multiple hierarchies may exist, it is recommended that these be explicitly modeled with additional query subjects.

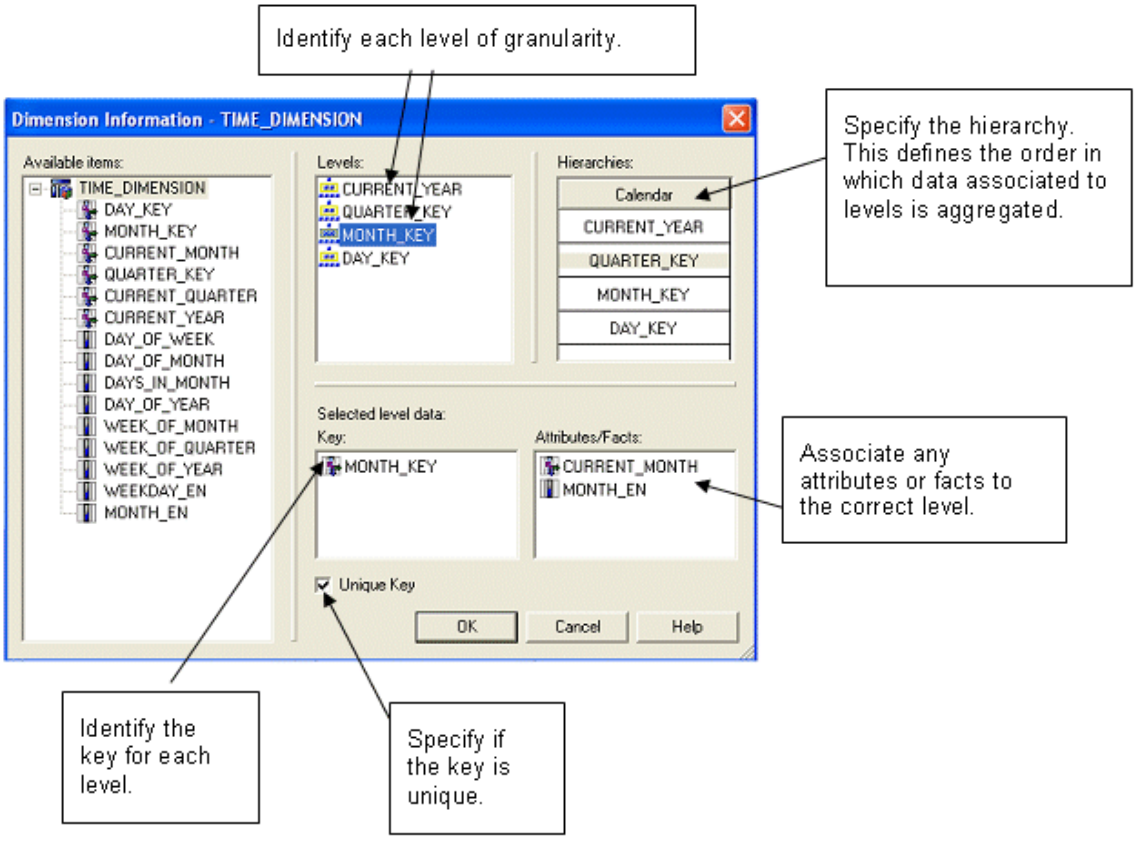
Star style dimension tables are perfect candidates for dimensional information because they typically have more than one level and they include attributes that are associated to more than one level.

When and How to Use Dimensional Information

Use dimensional information to specify the relationship between levels in a multi-level dimension.

For example, there is a time dimension table that contains keys for days, months, quarters, and years. Day key is the primary key of the table, however, there are also keys for month, quarter and current_year, each of which can uniquely identify the data at their respective levels and could relate directly to facts in the database.

In order to define the behavior expected when querying at a one or more levels of time, dimensional information is used. Levels are defined for years, quarters, months, and days. A key is defined for each level and in this example, that key is sufficient to uniquely identify the level. A hierarchy is defined to specify the relationships between the levels: days roll up to months, months to quarters, and quarters to years. Attributes and facts can also be defined for each level. Any attribute or fact that is not explicitly associated to a level will be treated by default as if it were associated to the lowest level (day).



When performing multi-fact queries between facts at different levels of granularity, dimension information on conformed dimensions ensures correct behavior when a multi-fact query is submitted and each fact is recorded for a different time period (daily versus monthly, for example). The query engine will use its understanding of the relationship between days and months from the defined dimensional information to ensure that data is rolled up correctly.

How to Avoid Double-Counting

There are several common scenarios that can lead to double-counting:

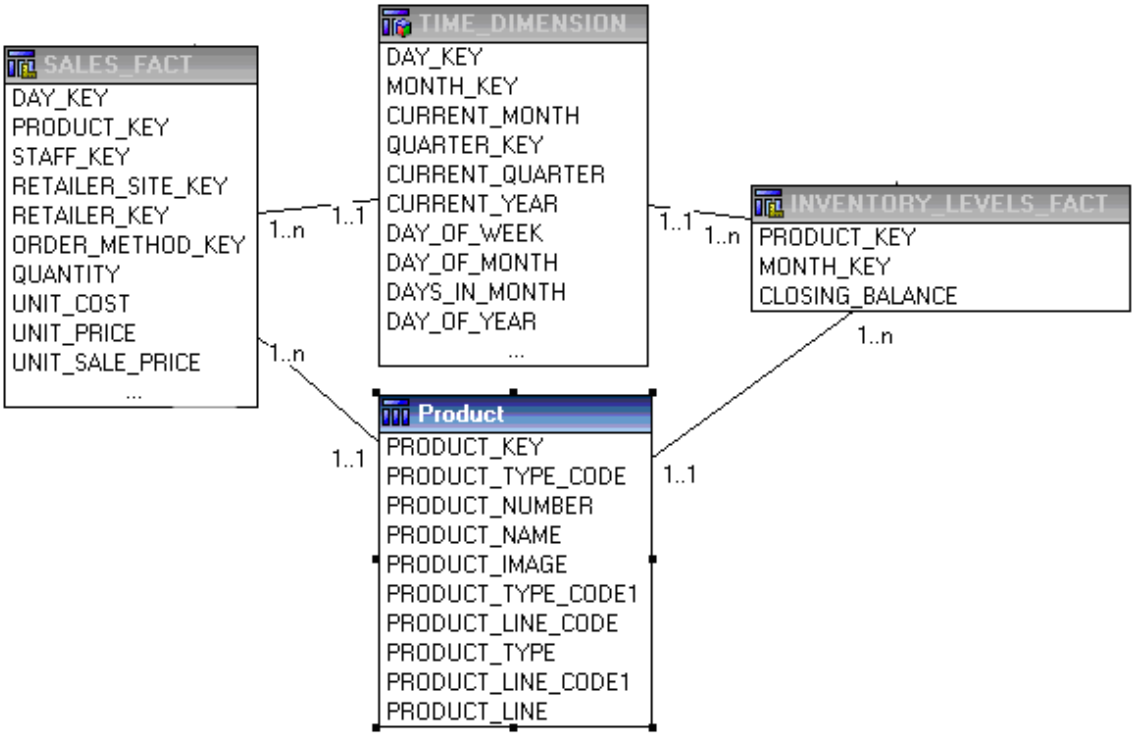
- multi-fact, multi-grain queries
- aggregate tables
- multiple hierarchies

Multi-Fact/Multi-Grain Queries

Multi-fact queries where de-normalized dimensions are related to fact tables on different keys (and therefore levels). A de-normalized dimension table typically has internal cardinality -- distinct levels of attribute data with keys that have a hierarchical relationship to each other.

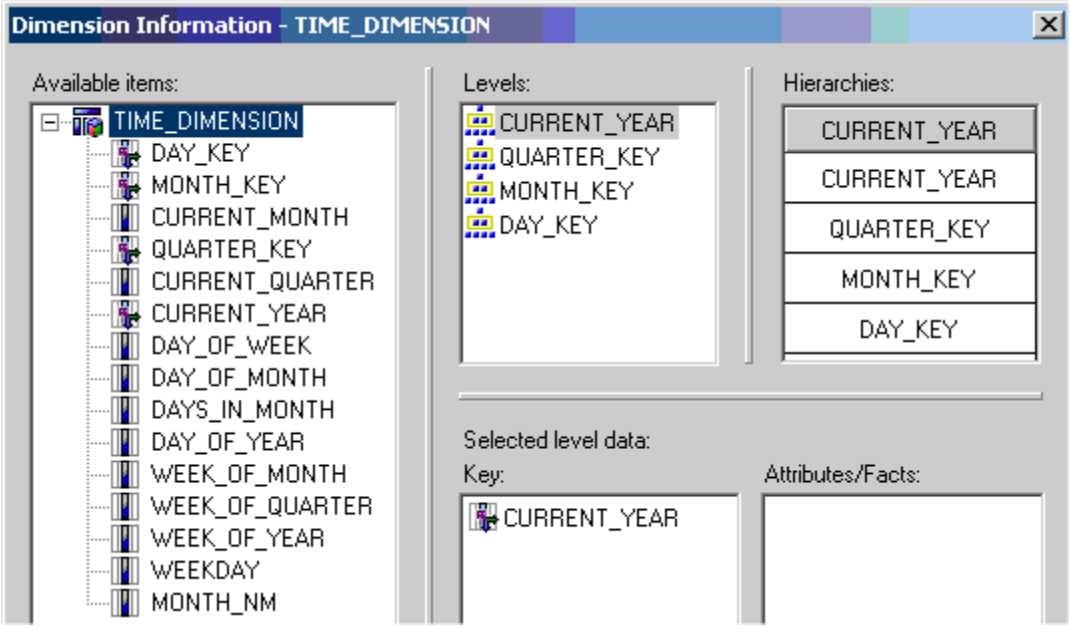
Example: Multi-Fact/Multi-Grain Query with Dimensional Information

A multi-fact/multi-grain query is typically the result of multiple star schemas sharing conformed dimensions but using keys for different levels of the same hierarchy. There is potential for double counting the since an aggregation is required to bring quantitative items from both fact tables to the same level of granularity for reporting.



The example above is the same as the one used in Example 2 and shows two fact tables sharing a time dimension and a product dimension. The Time dimension has four levels: Year, Quarter, Month, and Day. It has a relationship to Inventory Levels Fact on the Month key and to Sales Fact on Day key. The Product dimension has three levels: Product Line, Product Type, and Product. It has relationships to both fact tables on the Product key.

A report on these four tables will, by default, be aggregated to retrieve records from each fact table at the lowest common level of granularity, in this case at the month and product level. Without specifying dimensional information on the time dimension, it is possible that incorrect aggregation could occur – for example monthly fact values could be rolled up based on the lower time level of days. By using dimensional information to specify that days roll up to months, months to quarters, and quarters to years, any request to aggregate based on time will ensure that days are rolled up to months.



To see the SQL and the results generated for this case, refer to the example in Example 2.

The SQL generated in Example 2 is often mistaken by the modeler as incorrect or inefficient because of a full outer join shown in the Cognos SQL. Many modelers may be tempted to remove the full outer join by changing the SQL type of the query subjects to native SQL or by changing the relationships between the fact query subjects and the dimension query subjects to one-to-one. Once these changes have been made, they will find that the results are incorrect as shown below.

Example: Incorrect Result without Dimensional Information

By not specifying dimensional information on the dimension query subject, Time for example, you will find that the closing balance totals are multiplied by the number of days in the month (or rows in the time dimension per month). This is an incorrect result.

MONTH_KEY: between 200101 and 200103

PRODUCT_NAME_EN: Aloe Relief, Husky Rope 50

MONTH_KEY	PRODUCT_NAME_EN	QUANTITY	CLOSING_BALANCE
200101	Aloe Relief	448	392,584
200102	Aloe Relief	44	353,136
200103	Aloe Relief	764	369,148
200102	Husky Rope 50		273,728
200101	Husky Rope 50		307,334
200103	Husky Rope 50		288,238
Summary		**1,256**	**1,984,168**

Please note that if the underlying Time Dimension table has a primary key index on the Day level id, then Cognos ReportNet will detect this and render the correct results because it will understand that the month level join will not be unique.

Is the relationship between the Time dimension and Inventory Levels fact a many-to-many relationship?

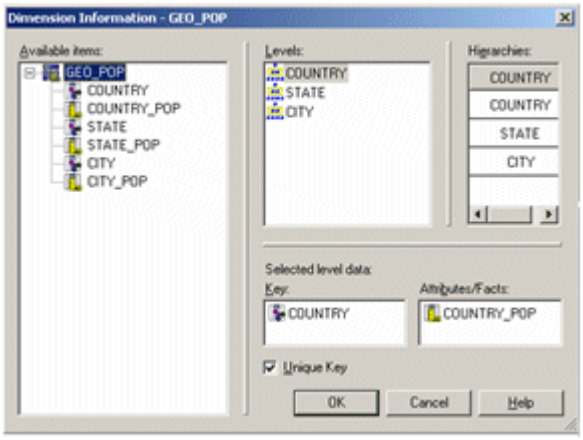
The join between the Time dimension and Inventory Levels fact is actually a many-to-many relationship because there are duplicate months in the time dimension (one per day). By specifying the dimensional information, we are giving Cognos ReportNet information about the internal cardinality of the Time dimension that allows us to specify the relationship as one-to-many. At runtime this will force the months to be distinctly joined and will avoid any double-counting issues.

Aggregate Tables

Aggregate tables contain primarily quantitative data aggregated to a single level or multiple levels. They are not frequently encountered and are not often recommended when creating a warehouse but they do exist.

Example: Aggregate Tables With Dimensional Information

A query subject has internal cardinality and there are quantitative items that are associated to levels above the lowest level. In this example, the GEO_POP query subject has populations for each level of the geographic hierarchy.



Once dimensional information is specified, the query is written to prevent aggregating repeated values for a given key.

```
select
    GEO_POP.COUNTRY as COUNTRY,
    XSUM(GEO_POP.COUNTRY_POP for GEO_POP.COUNTRY ) as COUNTRY_POP
from
    (select
        GEO_POP.COUNTRY as COUNTRY,
        XMIN(GEO_POP.COUNTRY_POP for GEO_POP.COUNTRY ) as COUNTRY_POP
    from
        GEO_POP
    group by
        GEO_POP.COUNTRY
        ) GEO_POP
    group by
        GEO_POP.COUNTRY
```

COUNTRY	COUNTRY_POP
Canada	30,000,000
Summary	**30,000,000**

Example: Incorrect Result Without Dimensional Information

Without specifying dimensional information, a query against any of the higher levels with its population will result in double counting. In this example, populations for Cities get added to the

already summarized values for State, which in turn get added to the summary value for the Country.

Note: If you do not associate an item to a level, Framework Manager will assume that it is associated to the lowest level (that is, that it can be safely aggregated using all rows).

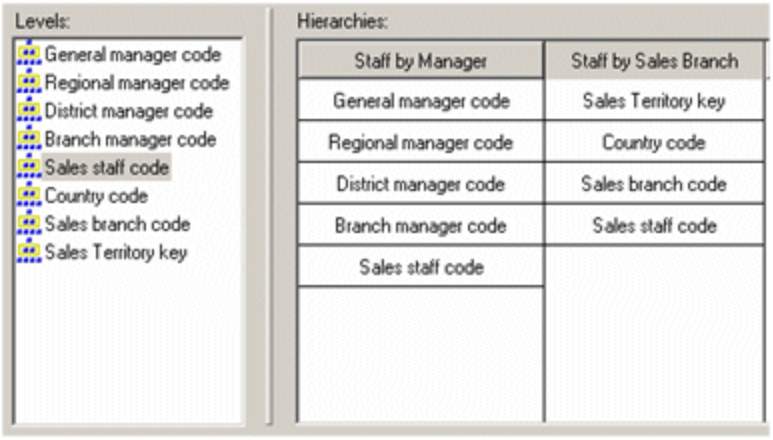
```
select
    GEO_POP.COUNTRY as COUNTRY,
    XSUM(GEO_POP.COUNTRY_POP for GEO_POP.COUNTRY ) as COUNTRY_POP
from
    GEO_POP GEO_POP
group by
    GEO_POP.COUNTRY
```

COUNTRY	COUNTRY_POP
Canada	150,000,000
Summary	**150,000,000**

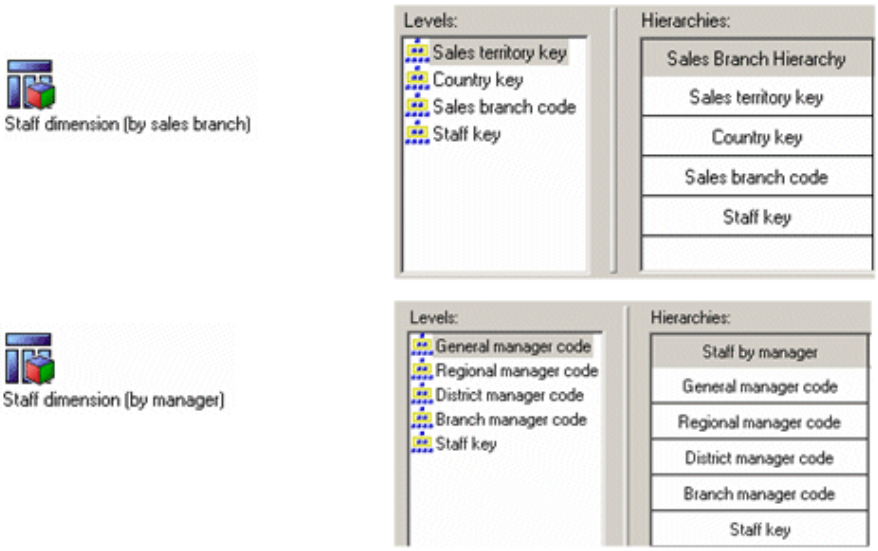
Multiple Hierarchies

You can specify multiple hierarchies implicitly or explicitly in Framework Manager. With multiple hierarchies specified implicitly, the query engine will select a hierarchy at runtime based on the query items selected by the user.

If the user selects more query items from one hierarchy than the other, that is the hierarchy to be applied to any aggregations performed on the data automatically.



If you wish to allow your users to be aware of and to select the hierarchy that they wish to use, we recommend that you create separate model query subjects for each hierarchy. The advantage is that you can customize query item names and include only those items that are relevant to the particular hierarchy.



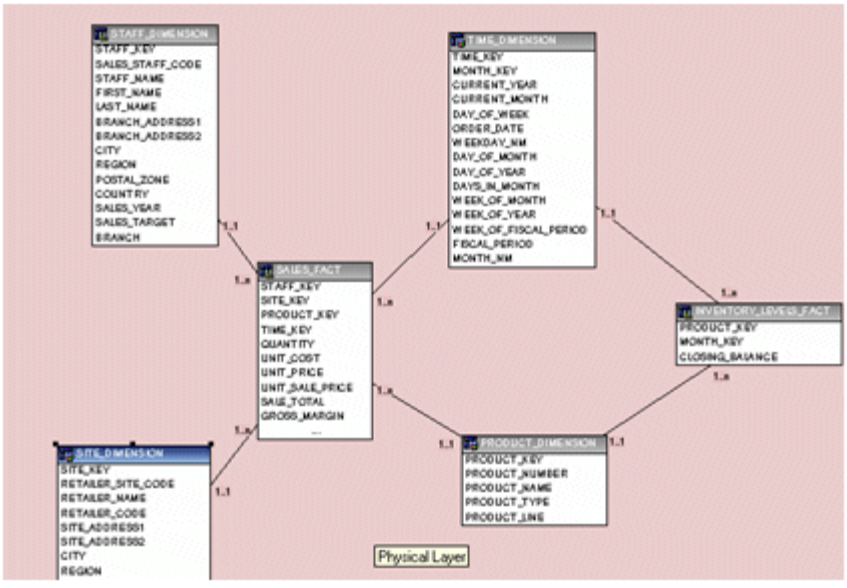
Create Star Schema Groupings

Framework Manager Modeling Guidelines

Star schema groupings are the key to providing context for queries. By creating star schema groupings in the business view of the model, it is possible to indicate to end users which dimensions relate to which facts and which dimensions are common to multiple facts. Star schema groupings also enable multi-fact querying.

Multiple Conformed Star Schemas

In dimensionally modeled data, it is likely that you will see fact query subjects that share dimension query subjects, resulting in what looks like a web in your diagram. Join ambiguity is an issue in this situation, when you use multiple dimensions without any items from the fact table.



For example, with the relationships above, how do you write a report that lists Products and Year? The business question may be which products were in inventory in 2002 or it may be which

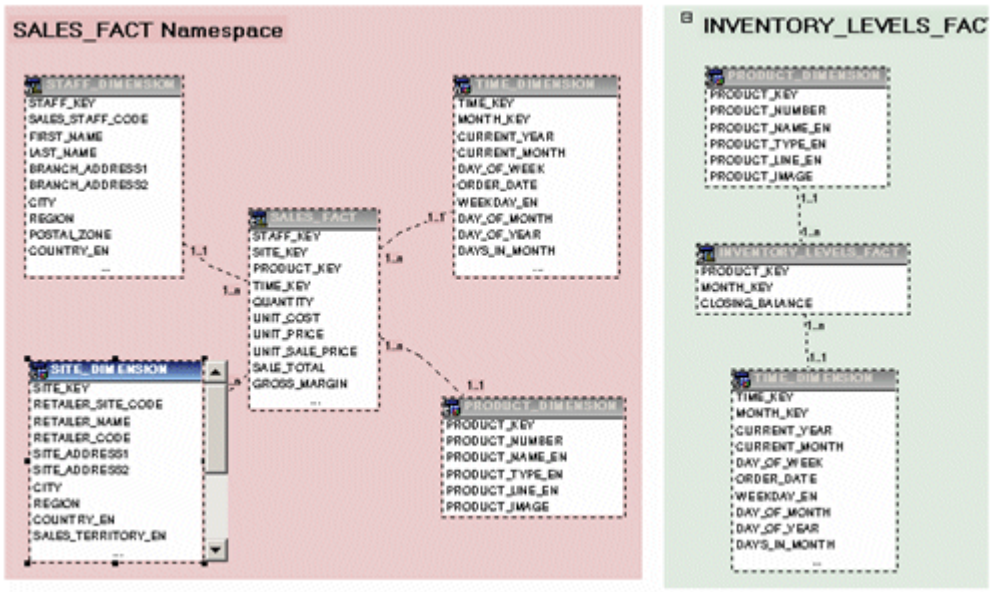
products were sold in 2002. Although this query involves only the time and product dimensions, these dimensions are related via multiple fact tables. Given the option to relate these dimensions on the Orders fact table or the Returns fact table how do you let the user choose which context they are interested in?

Set the Context of a Query

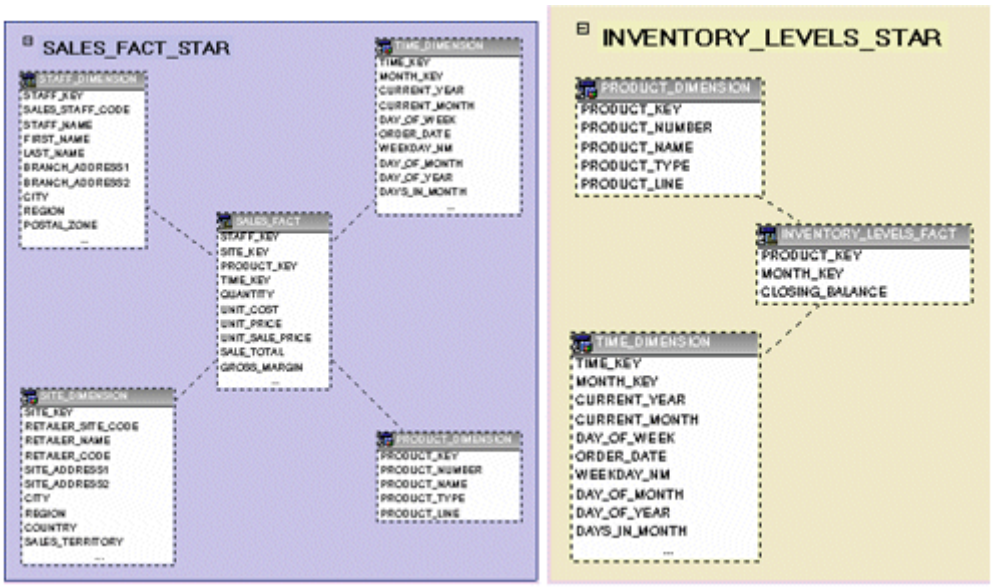
The way to resolve this join ambiguity is to create star schema groupings:

- select a fact table
- choose Create Star Schema Grouping from the menu
- choose the dimensions you want from a list in the wizard
- choose the options to create shortcuts for all query subjects and move to a new namespace

By doing this for all star schemas, you resolve join ambiguity by placing shortcuts to the fact and all dimensions in a single namespace along with shortcuts to the relationships between them. The shortcuts for conformed dimensions in each namespace will be identical and will be references to the original object, thus it will be truly conformed.

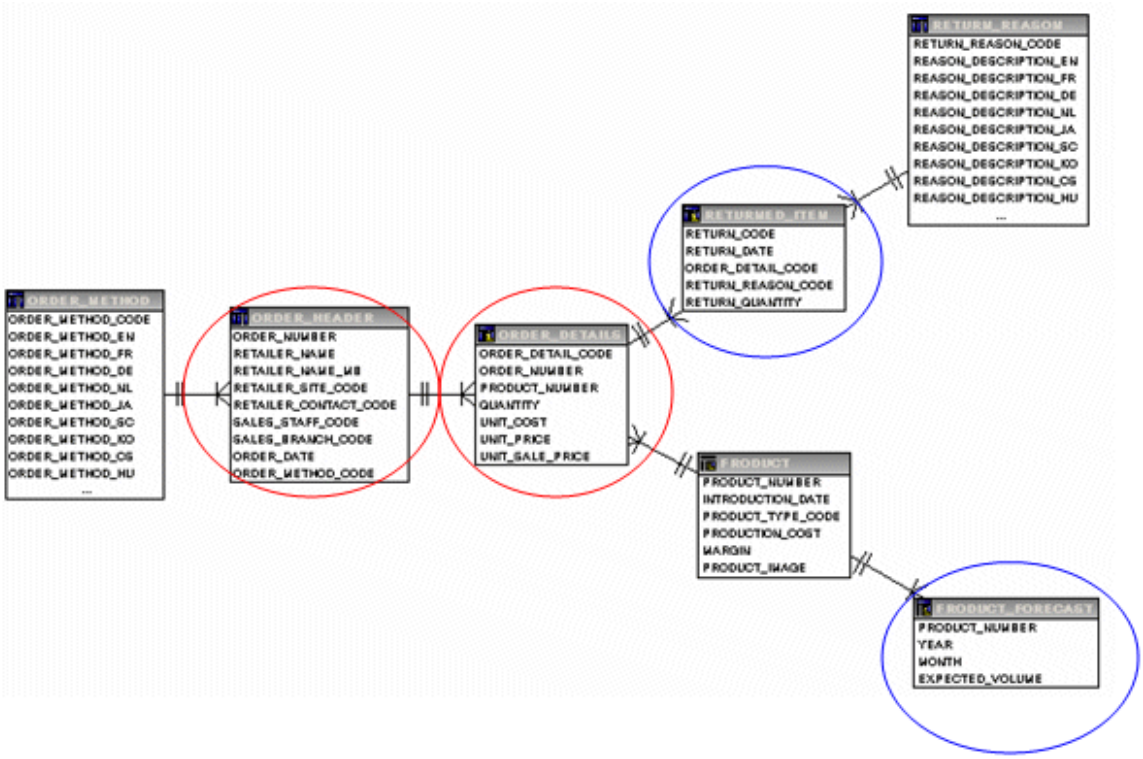


With a namespace created for each star schema, it will now be clear to a business user that to write a report on Products Sold in 2002, they will choose Product and Year from the Sales Fact Namespace. Because a relationship between Product, Time and Sales Fact is the only relationship available in this context, it will be used to return the data.


SALES_FACT_STAR
TIME_DIMENSION
TIME_KEY
MONTH_KEY
CURRENT_YEAR
CURRENT_MONTH
DAY_OF_WEEK
ORDER_DATE
WEEKDAY_NM
DAY_OF_MONTH
DAY_OF_YEAR
DAYS_IN_MONTH
STAFF_DIMENSION
STAFF_KEY
SALES_STAFF_CODE
STAFF_NAME
FIRST_NAME
LAST_NAME
BRANCH_ADDRESS1
BRANCH_ADDRESS2
CITY
REGION
POSTAL_ZONE
SALES_FACT
STAFF_KEY
SITE_KEY
PRODUCT_KEY
TIME_KEY
QUANTITY
UNIT_COST
UNIT_PRICE
UNIT_SALE_PRICE
SALE_TOTAL
GROSS_MARGIN
SITE_DIMENSION
SITE_KEY
RETAILER_SITE_CODE
RETAILER_NAME
RETAILER_CODE
SITE_ADDRESS1
SITE_ADDRESS2
CITY
REGION
COUNTRY
SALES_TERRITORY
PRODUCT_DIMENSION
PRODUCT_KEY
PRODUCT_NUMBER
PRODUCT_NAME
PRODUCT_TYPE
PRODUCT_LINE
INVENTORY_LEVELS_STAR
PRODUCT_DIMENSION
PRODUCT_KEY
PRODUCT_NUMBER
PRODUCT_NAME
PRODUCT_TYPE
PRODUCT_LINE
INVENTORY_LEVELS_FACT
PRODUCT_KEY
MONTH_KEY
CLOSING_BALANCE
TIME_DIMENSION
TIME_KEY
MONTH_KEY
CURRENT_YEAR
CURRENT_MONTH
DAY_OF_WEEK
ORDER_DATE
WEEKDAY_NM
DAY_OF_MONTH
DAY_OF_YEAR
DAYS_IN_MONTH

Appendix A

Why Resolve Ambiguously Identified Dimensions and Facts?



If the areas circled in red are not resolved then it is possible that unpredictable or incorrect queries could result. Side effects of these queries can range from double counted results to result sets that are incorrectly related and queries that are inefficient.

Several common manifestations of this are discussed below:

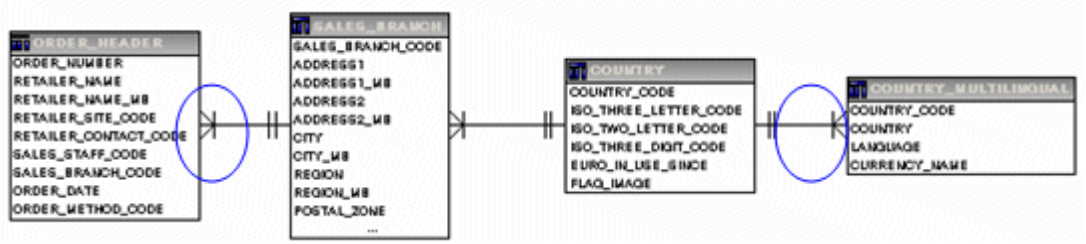
- Queries with incorrectly related data
- Incorrect summaries for queries with multiple facts

Resolving the areas circled in red can be accomplished by using a combination of data source and model query subjects. There are some cases that can even be resolved with the addition of filters to the query subjects that effectively change the cardinality from maximum (n) to minimum (1).

Resolve Queries That Are Incorrectly Split

Using the rule described above for identifying a fact, we can see that in this scenario Order Header and Country Multilingual are behaving as facts. In reality, the Country Multilingual query subject contains only descriptive information and seems to be more of a lookup table. From a dimensional or business modeling perspective, Country Multilingual is an extension of country.

Question: Why is leaving the model like this a problem?



Answer:

Test this model by authoring a report on the number of orders per city, per country. Using this model will return an incorrect result. The numbers are correct for the cities but Amsterdam is not in Australia. 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 we look when we see something like this is in the SQL.

In this case we see a stitched query, (for more information see the section on Understanding Dimensionally Optimized Queries) 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, using the relationships that relate the facts to each other as well as dimensional information for the conformed (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 on the conformed dimension(s).

```
select
    D3.COUNTRY as COUNTRY,
    D2.CITY as CITY,
    D2.ORDER_NUMBER as ORDER_NUMBER
from
    (select
        SALES_BRANCH.CITY as CITY,
         XCOUNT(ORDER_NUMBER for SALES_BRANCH.CITY ) as ORDER_NUMBER,
        RSUM(1 at SALES_BRANCH.CITY order by SALES_BRANCH.CITY asc local) as sc
    from
        ORDER_HEADER,
        SALES_BRANCH
    where
        (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
        COUNTRY_MULTILINGUAL
    where
```

This query has no coalesce, which is the first indication of trouble.

RSUM indicates an attempt to create a valid key.

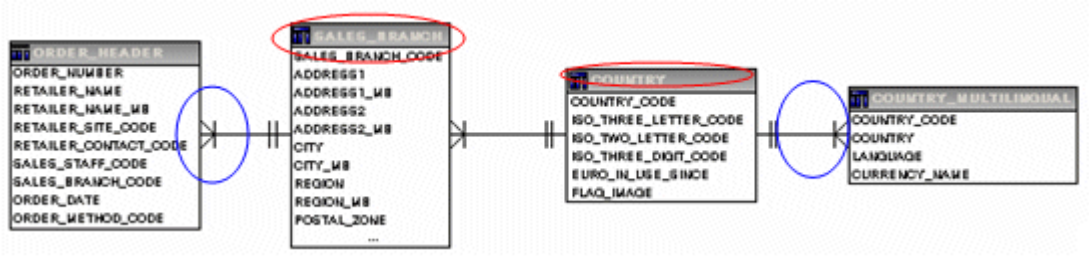
```
      (COUNTRY_MULTILINGUAL."LANGUAGE" = 'EN')
   group by
      COUNTRY_MULTILINGUAL.COUNTRY
   order by
      COUNTRY asc
   ) D3
   on (D2.sc = D3.sc)
order by
   COUNTRY asc,
   CITY asc
```

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 would have to go to the model.

In this case, the query items used in the report came from four 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.



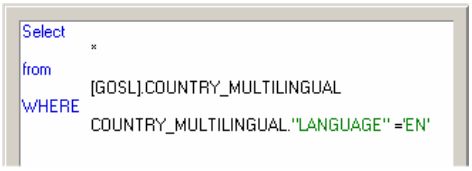
Problem:

The query splits because the query engine sees this as a multi-fact query and 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 of the data.

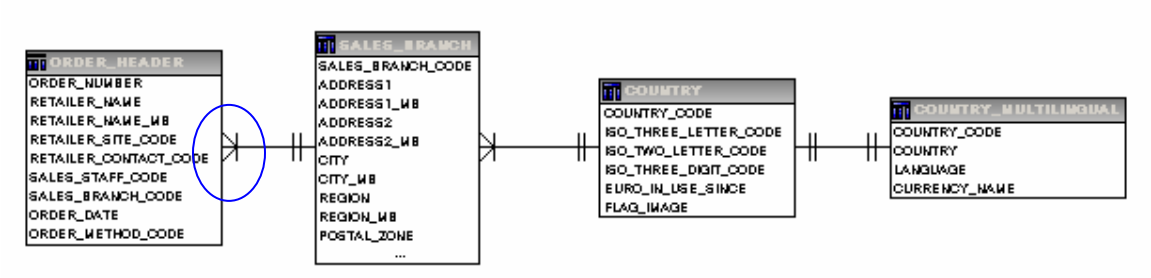
Solution 1:

Add a filter to Country Multilingual that changes the cardinality of the relationship to 1..1 = 1..1.



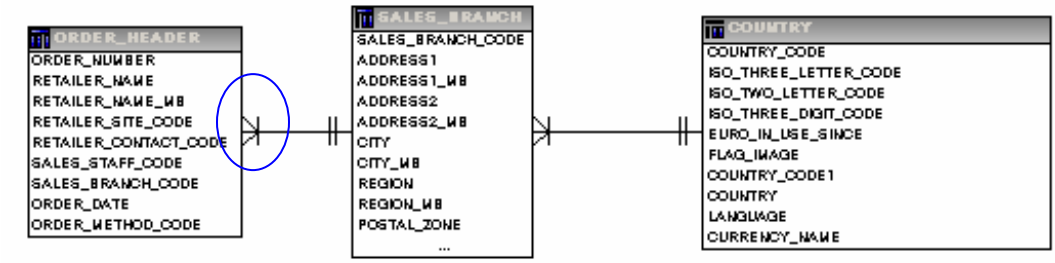
OR: Add the filter on the relationship and change the cardinality to 1..1=1..1

This 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

And the SQL is no longer a stitched query

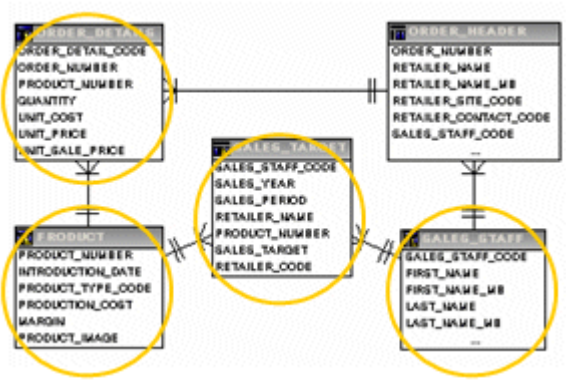
```
select
    COUNTRY.COUNTRY as COUNTRY,
    SALES_BRANCH.CITY as CITY,
    XCOUNT(ORDER_HEADER.ORDER_NUMBER for COUNTRY,CITY ) as Number_of_Orders
from
    (select
        COUNTRY.COUNTRY_CODE as COUNTRY_CODE,
        COUNTRY_MULTILINGUAL.COUNTRY as COUNTRY
    from
        COUNTRY,
        COUNTRY_MULTILINGUAL
    where
        (COUNTRY_MULTILINGUAL."LANGUAGE" = 'EN') and
        (COUNTRY.COUNTRY_CODE = COUNTRY_MULTILINGUAL.COUNTRY_CODE)
    ) COUNTRY,
    SALES_BRANCH,
    ORDER_HEADER
where
    (SALES_BRANCH.SALES_BRANCH_CODE = ORDER_HEADER.SALES_BRANCH_CODE) and
    (COUNTRY.COUNTRY_CODE = SALES_BRANCH.COUNTRY_CODE)
```

```
group by
   COUNTRY.COUNTRY, SALES_BRANCH.CITY
order by
   COUNTRY asc, CITY asc
```

Resolve Queries That Are Split in the Wrong Place - Blind Spots

There are cases where a join is ignored, which gives an incorrect result.

A report on Target vs Actual Sales by Product and Sales Staff



Write a report that reports Sales Target and Actual Sales grouped by Product and Sales Staff. The result is as follows, it seems that Actual Sales are a bit too large.

PRODUCT_NAME	LAST_NAME	ACTUAL_SALES	SALES_TARGET
Aloe Relief	Wiesinger	69,346.28	1,000
Husky Rope 50	Wiesinger	1,198,642.74	3,300

To verify the result, query each fact separately.

PRODUCT_NAME	LAST_NAME	ACTUAL_SALES
Aloe Relief	Wiesinger	1,413.96
Husky Rope 50	Wiesinger	4,610.36

PRODUCT_NAME	LAST_NAME	SALES_TARGET
Aloe Relief	Wiesinger	1,000
Husky Rope 50	Wiesinger	3,300

This shows that you are getting a much larger result for Actual Sales than you should.

Taking a look at the SQL shows

```
select
   coalesce(D2.PRODUCT_NAME,D3.PRODUCT_NAME) as PRODUCT_NAME,
   D3.LAST_NAME as LAST_NAME,
   D3.c3 as c3,
   D2.ACTUAL_SALES as ACTUAL_SALES,
   D3.SALES_TARGET as SALES_TARGET
from
   (select
      PRODUCT.PRODUCT_NAME as PRODUCT_NAME,
      SALES_STAFF.LAST_NAME as LAST_NAME,
      SALES_STAFF.SALES_STAFF_CODE as c3,
      XSUM(SALES_TARGET.SALES_TARGET for
      PRODUCT.PRODUCT_NAME,SALES_STAFF.LAST_NAME,SALES_STAFF.SALES_STAFF_CODE )
      as SALES_TARGET
   from PRODUCT, SALES_STAFF, SALES_TARGET
```

Only one of the dimension columns has a coalesce. This is an indication that the query has been improperly split.

```
where
   (PRODUCT.PRODUCT_NAME in ('Aloe Relief','Husky Rope 50')) and
   (SALES_STAFF.SALES_STAFF_CODE = 12) and
   (PRODUCT.PRODUCT_NUMBER = SALES_TARGET.PRODUCT_NUMBER) and
   (SALES_STAFF.SALES_STAFF_CODE = SALES_TARGET.SALES_STAFF_CODE)
group by
   PRODUCT.PRODUCT_NAME,
   SALES_STAFF.LAST_NAME,
   SALES_STAFF.SALES_STAFF_CODE
) D3
full outer join
(select
   PRODUCT.PRODUCT_NAME as PRODUCT_NAME,
   XSUM((ORDER_DETAILS.QUANTITY * ORDER_DETAILS.UNIT_SALE_PRICE) for
   PRODUCT.PRODUCT_NAME ) as ACTUAL_SALES
from PRODUCT, ORDER_DETAILS
where
   (PRODUCT.PRODUCT_NAME in ('Aloe Relief','Husky Rope 50')) and
    (PRODUCT.PRODUCT_NUMBER = ORDER_DETAILS.PRODUCT_NUMBER)
group by
   PRODUCT.PRODUCT_NAME
) D2
on (D3.PRODUCT_NAME = D2.PRODUCT_NAME)
```

The grouping is correct for the Sales Target side of the query. This explains why the Sales Target is accurate.

The quantity side of the stitched query is only grouped by Product, which explains why the Actual Sales are too large.

Why did this happen?

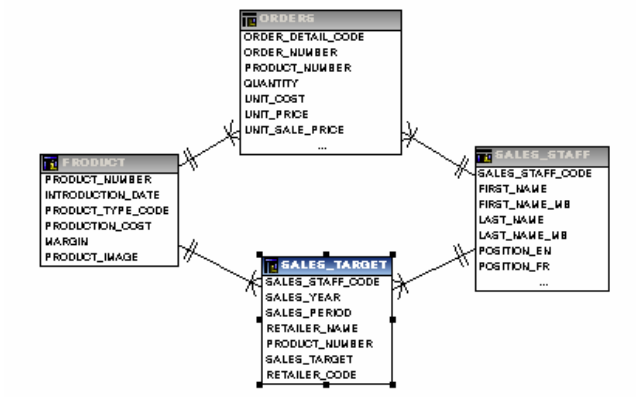
Because nothing was selected from Order Header, it was not included in the query so there was no way to link Sales Staff to Order Details. Linking Sales Staff to Order Details was not necessary to connect all the tables, there was already a relationship between Sales Staff and Sales Target.

There are two ways to resolve this:

- Include an item from Order Header (but how would an end user know to do that?)
- Model Order Header and Order Details as a single query subject. This is the correct answer from both a dimensional and business modeling standpoint. Also, a simpler model makes it harder to write a report that generates bad SQL.

Fix it in the model

Consolidate query subjects where logical groupings are possible. In this case combine the Order Details and Order Header query subjects.



Now, for the same report the result looks like this:

PRODUCT_NAME	LAST_NAME	ACTUAL_SALES	SALES_TARGET
Aloe Relief	Wiesinger	1,413.96	1,000
Husky Rope 50	Wiesinger	4,610.36	3,300

```
select
     coalesce(D2.PRODUCT_NAME,D3.PRODUCT_NAME) as PRODUCT_NAME,
    coalesce(D2.LAST_NAME,D3.LAST_NAME) as LAST_NAME,
     D2.ACTUAL_SALES as ACTUAL_SALES,
    D3.SALES_TARGET as SALES_TARGET
from
    (select
        PRODUCTS.PRODUCT_NAME as PRODUCT_NAME,
        SALES_STAFF.LAST_NAME as LAST_NAME,
        XSUM((ORDERS.QUANTITY * ORDERS.UNIT_SALE_PRICE) for
        PRODUCTS.PRODUCT_NAME,SALES_STAFF.LAST_NAME ) as ACTUAL_SALES
    from
        PRODUCTS, SALES_STAFF, ORDERS
    where
        (PRODUCTS.PRODUCT_NAME in ('Aloe Relief','Husky Rope 50')) and
        (SALES_STAFF.LAST_NAME = 'Wiesinger') and
         (SALES_STAFF.SALES_STAFF_CODE = ORDERS.SALES_STAFF_CODE) and
        (PRODUCTS.PRODUCT_NUMBER = ORDERS.PRODUCT_NUMBER)
    group by
        PRODUCTS.PRODUCT_NAME,
        SALES_STAFF.LAST_NAME
    ) D2
    full outer join
    (select
        PRODUCTS.PRODUCT_NAME as PRODUCT_NAME,
        SALES_STAFF.LAST_NAME as LAST_NAME,
        XSUM(SALES_TARGET.SALES_TARGET for
        PRODUCTS.PRODUCT_NAME,SALES_STAFF.LAST_NAME ) as SALES_TARGET
    from
        PRODUCTS, SALES_STAFF, SALES_TARGET
    where
        (PRODUCTS.PRODUCT_NAME in ('Aloe Relief','Husky Rope 50')) and
        (SALES_STAFF.LAST_NAME = 'Wiesinger') and
        (SALES_STAFF.SALES_STAFF_CODE = SALES_TARGET.SALES_STAFF_CODE) and
        (PRODUCTS.PRODUCT_NUMBER = SALES_TARGET.PRODUCT_NUMBER)
    group by
        PRODUCTS.PRODUCT_NAME,
        SALES_STAFF.LAST_NAME
    ) D3
    on ((D2.PRODUCT_NAME = D3.PRODUCT_NAME) and (D2.LAST_NAME = D3.LAST_NAME))
```

There is now a coalesce statement on each shared table. This is also known as a conformed dimension.

There is now a correct group by on the Orders query.

There is still a correct group by on the Sales Target query.