Probably, the most common situation that requires more complex data modeling is where we have more than one fact table—a table containing a number that we might use in a calculation—all of which have several common key fields linking to other dimension tables.

There are several ways to deal with this situation; by far the easiest way is to simply concatenate the fact tables together to form one large fact table. The result is often a typical star or snowflake schema.

Getting ready

Load the following script:

Store:

Load \* Inline [

StoreID, StoreName

1, Store A

2, Store B

];

Calendar:

Load MonthID As DateID, Month Inline [

MonthID, Month

1, Jan

2, Feb

];

Product:

Load \* Inline [

ProductID, Product

1, Product A

2, Product B

];

Sales:

LOAD \* INLINE [

DateID, StoreID, ProductID, SaleQty, SaleValue

1, 1, 1, 2, 23

1, 1, 2, 4, 24

www.it-ebooks.info

Chapter 6

155

2, 1, 1, 4, 33

2, 1, 2, 3, 28

1, 2, 1, 2, 21

1, 2, 2, 4, 30

2, 2, 1, 3, 25

];

Waste:

LOAD \* INLINE [

DateID, StoreID, ProductID, WasteQty, WasteValue

1, 1, 1, 1, 10

2, 1, 2, 1, 9

1, 2, 2, 2, 17

2, 2, 2, 1, 8

];

Note that the data schema contains a synthetic key.



How to do it…

1. Edit the script and modify the Waste table load as follows:

//Waste:

Concatenate (Sales)

LOAD \* INLINE [

DateID, StoreID, ProductID, WasteQty, WasteValue

www.it-ebooks.info

*Data Modeling*

1, 1, 1, 1, 10

2, 1, 2, 1, 9

1, 2, 2, 2, 17

2, 2, 2, 1, 8

];

1. Reload the script.



Note that the synthetic key has been removed



Note that the Sales table contains data for both sales and waste quantities and values, but they are on different lines.

How it works...

This works very well because of the way QlikView associations work and the fact that QlikView ignores null values for the purposes of calculations. So, even though we have nulls on several rows, these will just be ignored and the expressions will calculate correctly for the associated keys.

If I select a particular store, the sum of SalesValue for the store will still be calculated correctly as will the sum of WasteValue, and both of those expressions can be juxtaposed in a chart.

It is worth noting that you can achieve rows of data with different granularity (for example, rows of sales with product and store but rows of budget that only have product and no store) with this method, and you should think carefully about what happens in this case. It is possible to use set expressions to allow different values to calculate under different selections, but it is not always valid to do so.

There's more...

Another approach might be to try and join the two tables using the common key values. This would only work if each combination of keys was unique in both tables (which it might be if you are using preaggregated data). If it isn't, you would end up with Cartesian joins of and duplicate, triplicate, etc. values. If there were missing values in either table, that would also cause issues.

One way to watch out for this is to count the number of rows in each table and count the number of rows in the resulting table. If there is a significant increase in rows, you might have a Cartesian issue.

See also

fThe *Creating a Key/Link table in QlikView* recipe

Creating a Key/Link table in QlikView

Where there are fact tables that mostly share a set of keys, concatenation should always be considered first. However, where there are many more keys in one fact table than another, and the additional keys are not relevant to the second table, it may be a better approach to create a key table to link the common keys.

In this recipe, we are going to create a budget for store and product but not put any date on this.

Getting ready

Load the following script:

Store:

Load \* Inline [StoreID, StoreName

1, Store A

2, Store B

];

Calendar:

Load MonthID As DateID, Month Inline [

MonthID, Month

1, Jan

2, Feb

];

Product:

Load \* Inline [

ProductID, Product

1, Product A

2, Product B

];

Sales:

LOAD \* INLINE [

DateID, StoreID, ProductID, SaleQty, SaleValue

1, 1, 1, 2, 23

1, 1, 2, 4, 24

2, 1, 1, 4, 33

2, 1, 2, 3, 28

1, 2, 1, 2, 21

1, 2, 2, 4, 30

2, 2, 1, 3, 25

];

Budget:

LOAD \* INLINE [

StoreID, ProductID, BudgetQty, BudgetValue

1, 1, 5, 50

1, 2, 6, 47

2, 1, 5, 41

2, 2, 4, 27

];

Note that the data schema contains a synthetic key.



1. Create a key field for the Sales and Budget tables that contains the common key fields.

Sales:

LOAD

AutoNumberHash256(StoreID, ProductID) As SalesBudgetID,

\*

INLINE [

DateID, StoreID, ProductID, SaleQty, SaleValue

1, 1, 1, 2, 23

1, 1, 2, 4, 24

2, 1, 1, 4, 33

2, 1, 2, 3, 28

1, 2, 1, 2, 21

1, 2, 2, 4, 30

2, 2, 1, 3, 25

];

Budget:

LOAD

AutoNumberHash256(StoreID, ProductID) As SalesBudgetID,

\*

INLINE [

StoreID, ProductID, BudgetQty, BudgetValue

1, 1, 5, 50

1, 2, 6, 47

2, 1, 5, 41

2, 2, 4, 27

];

1. Start to build the Key table from the data in the Sales table.

Key:

Load Distinct

SalesBudgetID,

StoreID,

ProductID

Resident

Sales;

1. Join all the matching values from the Budget table.

Join (Key)

Load Distinct

SalesBudgetID,

StoreID,

ProductID

Resident

Budget;

1. Finally, we drop the fields in the Sales and Budget tables that are now in the Key table.

// These fields are no longer needed in the fact tables

Drop Fields StoreID, ProductID From Sales;

Drop Fields StoreID, ProductID From Budget;

1. Reload the script.



Note that the synthetic key is gone.

How it works...

The Join load here, which is a full outer join, combines all the possible values from both tables into the Key table. Any selection on the product or store will still associate correctly to the fact values. The Distinct clause in Load ensures that only unique values are added to the Key table.

AutoNumberHash265 is a function that accepts a number of values and will always return the same integer if the same values are passed again. It is very useful for converting several key values into one integer key. You should note that it only works to return the same integer when used within the same load script. It would not be valid to use in different load scripts that might store the data in QVDs.

There's more...

In this case, we created the same key in both the tables. This is not always the case; as long as the key can associate correctly to the fact table, any key will do. A primary key from the fact table is often used.

See also

fThe *Concatenation of fact tables to avoid loops and synthetic keys* recipe