QlikView

Best Practices

QlikView Optimisation

Version: 1
Date: 20/03/2007
Author(s): STB/ABY

"A best practice is a technique or methodology that, through experience and research, has proven to reliably lead to a desired result."

Contents

Managing Your Data Sources

QlikView's ability to sit on top of multiple data sources give developers the challenge of ensuring that the data model and data that QlikView builds is done so in an efficient manner that generates the highest possible performance legel wihtin the analysis of the data.

Performance and Efficiency

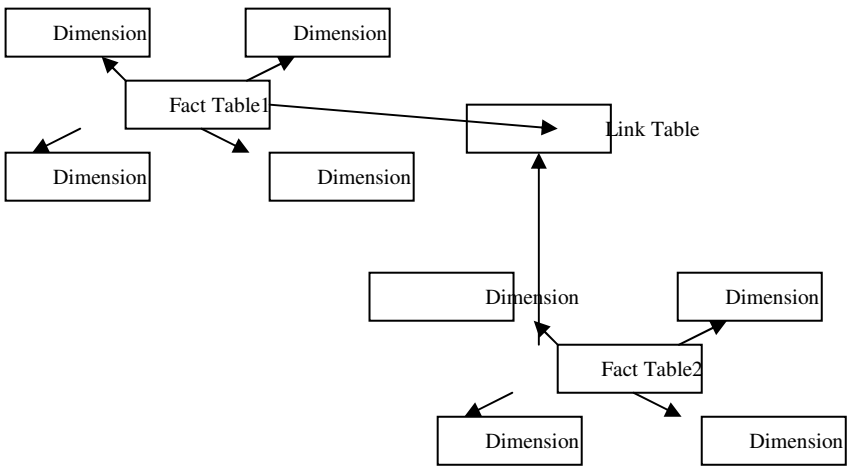
Although QlikView can analyse millions of rows of data (your hardware permitting) we have to ensure that we have developed the script in such a way that QlikView can perform as you would expect it to.

Data Model Scenarios and Solutions

QlikView can and will analyse multiple data sources and models but works most efficiently with a Star Schema or Snowflake data model. Some example are below to help you understand how to handle various data scenarios and how best to implement them within QlikView Script.

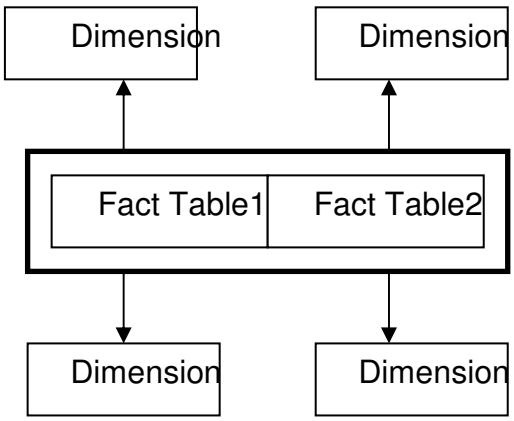
Multiple Star Schemas

Many organisations have various disparate data models that have 'loose' links between them but will want to analyse these sources all together. The data model would look something like the example below:



This example would work best within QlikView if we look to create one single fact table that spans your schemas and then has multiple dimension tables hanging off of it. The example below shows (by using the CONCATENATE function) how QlikView would work best with the data:

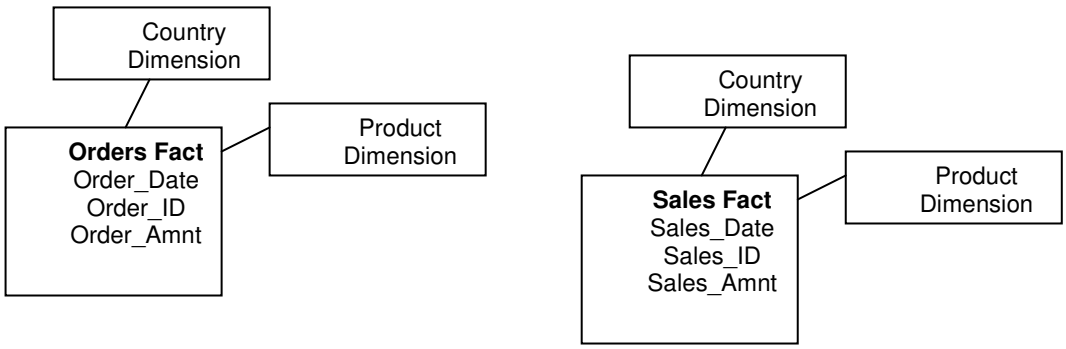
```
load * from file1.csv;
CONCATENATE
load * from file2.csv;
```



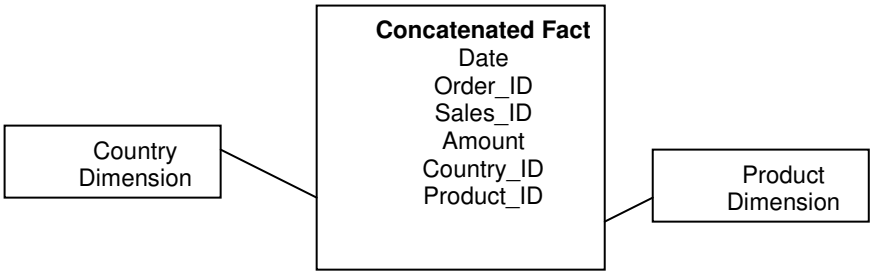
If two tables that are to be concatenated have different sets of fields, concatenation of two tables can still be forced with the concatenate prefix. This statement forces concatenation with an existing named table or the latest previously created logical table.

Functional example of Concatenate:

Before



After



Script Example:

```
Load OrdersFact
        Order_Date as Date
        Order_ID
        Order_Amount as Amount
        Country_ID
        Product_ID
        'Order' as TransactionType
CONCATENATE
Load SalesFact
        Sales_Date as Date
        Sales_ID
        Sales_Amount as Amount
        Country_ID
        Product_ID
        'Sale' as TransactionType
```

Placing the 'Sale' and 'Order' text types in the script will provide you with a column to determine the transaction type.

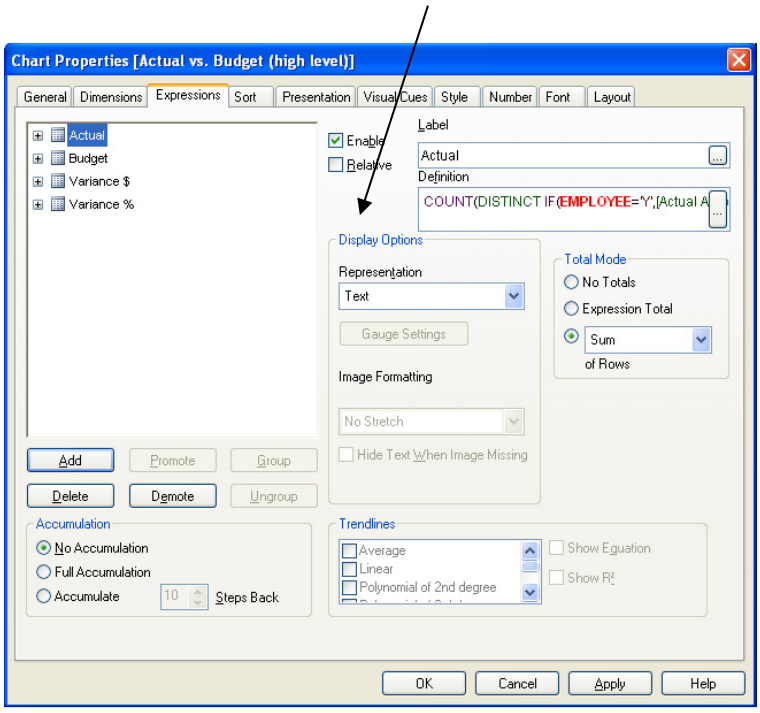
Complex Dimensions and Expressions

Many dimensions and expression that are to be placed in charts or tables require some degree of complex scripting such as IF THEN ELSE statements or WHERE [FIELD1] IS NULL.

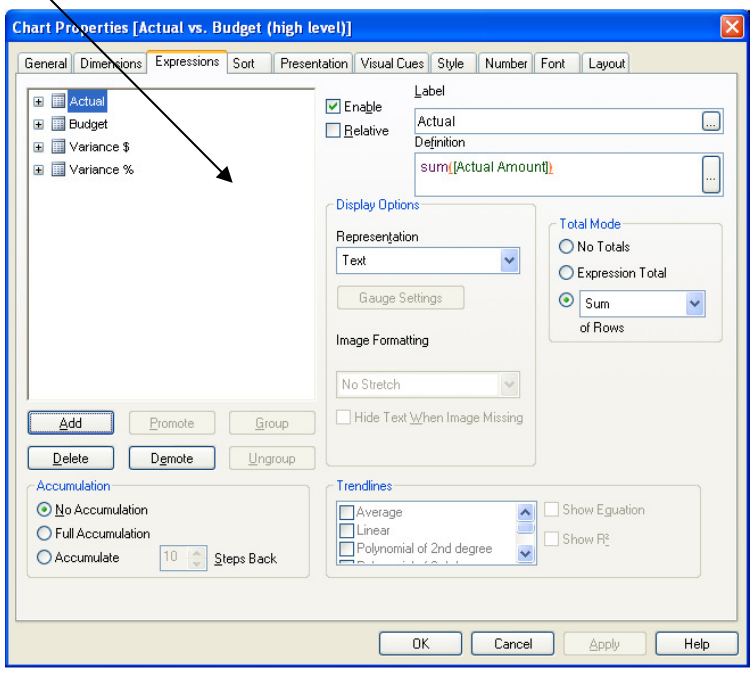
Many developers build there initial data model using the processes above but then stop amending the script and work solely within the dashboard/GUI. When designing/writing your script you should already be aware of some of the measures that you are looking to create in the end.

Where ever possible you should look to place all complex formulas and statements within the script of the application and not in the actual dashboard/application objects.

Example…..
If you are to use complex expressions within a dimension or calculation then you should move as much of this as possible in to the scripting of the QVW. In the example below you can see that the developer has added an IF statement to the Definition of an expression.



Where ever possible you should keep the expressions and dimensions as simple as you can and move the complex expression in to the script.



Resource intensive Expressions

The recommendations below should not be seen as universally beneficial. Use them when they improve the general state of the application or when they make that little bit of difference that makes or breaks.

Cases:

1. **Count** (Distinct '*FieldName*')
2. **If** (Condition(*Text*),…..)
3. **Sum** (If (Condition, '*FieldName*'…))
4. **If** (Condition, Sum('*FieldName*')..)
5. **If (** Condition1, Sum('*FieldName*'),
   If (Condition2, Sum('*FieldName*')……..
6. **Sort** text
7. Dynamic captions and text objects
8. Macro triggers ("on change")

Case1. Replace the *count()* with *sum()* and the distinct qualifier by assigning the value '1' to each distinct occurrence as it is read in the script.

Case2. Map Text to numeric e.g. by using *autonumber* and/or do the test in the script.

Case3. Here the aggregation is independent of the table dimensions and the result is distributed over the dimensions of the table. The problem can be treated either by doing the test in the script and aggregating in the table or by doing the whole operation in the script. There are numerous techniques for this e.g. *interval match, group by, peek, if….then…else.*

Case4. Included here to emphasize the difference to Case3. This aggregation is completely contextual.

Case5. The logic of nested *If..then else..* is conceptually easy but can often become troublesome to administer. We have seen cases with hundreds of nesting levels. This will be memory and CPU intensive. Often the "Conditions" can be replaced by transforming them. A typical example is aggregating quantity*price where price is variable. This can be handled by "extended interval match". If two conditions, e.g. " *A* **AND** *B* " are to be satisfied the test might be replaced by a condition "*C*" .

Case6. QlikView automatically evaluates if a *Field* is to be treated as *numeric, text* or *general*. Fields evaluated as *text* will be sorted as *text* which is the slowest sort operation. This can be replaced manually to sort by *load order.*

Case7. Expressions can be entered almost anywhere that you can enter text. The evaluation of an expression is however dependent on its environment. Expressions in charts and straight- and pivot- tables that are defined in the expressions dialog are embedded and only calculated when the object is active. For instance they are not

calculated when the object is minimized. On the other hand if the object title is calculated this calculation is performed every time any change occurs. We also have numerous ways of defining show conditions, calculation conditions etc. These tests will also be performed at all times. Some expressions are more expensive than others and of course become more expensive the more frequently they have to be evaluated. The introduction of asynchronous calculation has shifted the behaviour and these effects may have become more noticeable in your applications. The time functions e.g. Now(), Today() will be evaluated whenever a recalculation has to be done. Especially the Now() function can become quite costly since it causes a recalculation of the application every second. For example

If (ReloadTime()+3>Now(), 'Old Data', 'New Data')

Here one might consider…
If (ReloadTime()+3>Today(), 'Old Data', 'New Data')
As a simple test, put the expressions into textboxes. Then try sizing the textbox with Now() in it.

Case8. Macros can be set to be triggered by almost any event taking place in the application. Beware of cascading or recursive events, where one event triggers the next, which in turn …..

Adding Aggregatable Columns in to Your Script

It is sometime good to add manual columns in to your script to give you application the ability to sum up over these values. This will mean you can then place the complex statement (IF) in to your script and have a simple sum in your dashboard object.

To do this you should simply place an IF THEN ELSE statement in to your script that substitutes a database column or a 1 or 0 as a value to enable a summing/count to take place.

Examples:
IF(ACTIVE='Y',1,0)
IF(ACTIVE='Y',sales_amount,0)

Forcing through a 0 then takes our all of the unnecessary / unwanted values when you apply your SUM.

Hands On

The following is a list of examples of applied methods for the handling of the problems above. They are meant to illustrate the problem and to point at useful QlikView functionality. It is not possible to give a general recommendation as to which method is best, but the order of the examples is an indication.

Case1. Count(Distinct 'FieldName').

The distinct qualification, especially if text strings are read, is costly. A useful technique is to assign the value '1' to each new value as the field is read:

```
Load
        Alfa,
        if (peek('Alfa')=Alfa,0,1) as Flag1,
        Num
 resident table_1
 order by Alfa Asc;
```

Here the "peek" compares the value of Alfa being read with that previously read. If the values are the same "Flag" is set to 0, if they are different "Flag" is set to 1. The number of distinct values will then be = sum(Flag). Please note that the list has to be ordered and that when using "order by" in a load resident QlikView orders the list before starting to read.

Another method:

```
Load distinct
        Alfa,
        Alfa as AlfaDist
resident table_1;
```

Now Count(Distinct Alfa) can be replaced by a simple count: Count(AlfaDist).
Notice that Alfa is read twice, once with the original name to link to the original table, once with a new name to allow Count(). (Linking fields not allowed in Count()). All other fields must also be left out as they would degrade the distinct clause.

A third method is to give each distinct value of "Alfa" a numeric value:

```
table_2:
Load
        Alfa,
        Autonumber(Alfa) as AlfaNum,
```

```
        Num
resident table_1;
```

Count(Distinct AlfaNum) is a cheaper operation than Count(Distinct Alfa) since the comparison is of numeric values. An even cheaper method is to find the last (or largest) result of the autonumber function.

```
        set AlfaDistinctCount = peek( 'AlfaNum', -1, 'table_2' );
```

in the script or as expression:

```
        max( AlfaNum)
```

in a layout object.

Case2. If (Condition(Text),…..)

The testing of text strings is slower than numeric testing. Consider the expression

```
        If (Alfa= 'ABC', 'ABC', left (Alfa, 2))
```

The test could be done directly in the script without loosing any flexibility

```
Load
        *,
        If (Alfa = 'ABC', 1, 0) as Flag
resident table_1 ;
```

The expression becomes

```
        If ( Flag = 1,'ABC', left (Alfa, 2))
```

and the test is much simpler.

Case3. Sum(If (Condition, 'FieldName'…))

This case involves two steps. The testing of "Condition" and the aggregation of the result. Taking the previous example and adding the aggregation

```
                Sum ( If (Alfa= 'ABC',  Num*1.25 , Num) )
Load
        *,
        If (Alfa = 'ABC', 1, 0) as Flag
resident table_1 ;
```

The expression becomes

```
        Sum ( If ( Flag = 1, Num* 1.25 , Num ) )
```

The aggregation can also be done directly in the script as follows:

```
table_2:
Load
        *,
        If (Alfa = 'ABC', 1, 0) as Flag
resident table_1 ;

table_3:
Load
        Alfa,
        If ( Flag = 1, Num* 1.25  , Num ) as NewNum
resident table_2 ;

table_4:
Load
        Alfa,
        Sum( NewNum ) as SumNum
resident table_3
 group by Alfa ;
```

Note that the aggregation is done over Alfa as this is the dimension in the test.

Case5 Nested If..then else..

Often the “Conditions” can be replaced by transforming them. A typical example is aggregating quantity*price where price is variable. This can be handled by “extended interval match”.

Example:

```
sum((GAC12_STD_COST * GAC15_EXCHANGE_RATE) * GIV24_DISP_QTY)

Replaces

Sum(
        If((GAC12_EFCT_DT<= GIV23_REJ_DT and
                GAC12_EXPIRE_DT>GIV23_REJ_DT) and
                (GAC15_EFCT_DT<= GIV23_REJ_DT and GAC15_EXPIRE_DT>GIV23_REJ_DT),
                GAC12_STD_COST * GAC15_EXCHANGE_RATE) * GIV24_DISP_QTY,
                Null()))
and

 Sum(
        If(GAC12_EFCT_DT<= GIV23_REJ_DT,
                If(GAC12_EXPIRE_DT>GIV23_REJ_DT,
                        If(GAC15_EFCT_DT<= GIV23_REJ_DT,
                                If(GAC15_EXPIRE_DT>GIV23_REJ_DT,
                (GAC12_STD_COST * GAC15_EXCHANGE_RATE) * GIV24_DISP_QTY,
        Null())))))
```

by reading the fields *GAC12_STD_COST* and *GAC15_EXCHANGE_RATE* as ***slowly changing dimensions***. (Please refer to Reference Manual).