

Kimball Revisited – Context is our guide

Taking sustainability and agility in data modelling to the next level

Authors: Ronald Damhof & Frederik Hofstra
December 2011

Author bio

Ronald Damhof (Ronald.Damhof@prudenza.nl)

Independent consultant (www.prudenza.nl) for companies in Europe on strategy and execution regarding (data) quality, data governance, data warehousing and decision support. Focused on tangible results by coaching (agile) teams, architects and management.

Frederik Hofstra (Frederik.Hofstra@ordina.nl)

Business Intelligence & Data Warehousing consultant for a Dutch consultancy company, with a broad focus on architecture, infrastructure, data quality, and agile data warehousing.

Abstract

This paper is all about delivering quality data products. However, quality is by definition a relative construct. Its requirements can only be fixed by the time the user needs the data. Context - as we use the word in this paper - is a function of the *person involved* at a specific moment in *time* with the *quality attributes* he desires. And these contexts change all the time. Accommodating these constant changes is partly covered by continuous innovation in technology. However, the use of innovative technologies does not free us from proper software design. This paper addresses a software design perspective to offer the required agility.

Background

Both authors are involved in a data warehouse project with a university in the Netherlands. This university implemented a new SAP module; Student LifeCycle Management. A comprehensive module covering all processes from enrolling students, to credit scoring, to progress reports, to eventually graduating. This module is also the main source for master data like the academic structure of programmes and modules, as well as the organisational structure (faculties, divisions, subdivisions, etc.) and the student information.

The strategy of the university is to deliver a service to their organisation that is capable of delivering data in various 'truths' to people or information systems and with an array of functionality (e.g.: reporting, analysis, etc.).

The solution architecture consists of two storage layers. The first one is based on Data Vault¹ principles and heuristics and the second one is a dataset layer. The latter is the object of discussion in this paper. This solution architecture is currently available in production mode and is - as such - a proven solution.

For the sake of explanation and understandability, the examples in this paper are based on the classic processes of sales and purchasing and are somewhat simplified.

Introduction

Lets face facts, our job in data exploitation is ultimately to deliver quality products that add value to the organisation. A no-brainer one could say. But, what products are requested? Or even better, what *quality* does the buyer of our products require? The first question might be answered by saying; "they want data". However, the second question cannot easily be answered. "Quality is value to some person" [1] implies the relativity of quality. What quality is to some person may not be sufficient quality to another. Subsequently, each person behind the statement of quality is the only person with the answer to the question what constitutes "quality."

¹Data Vault Business objectives for next generation data warehousing- T.Breur/R.D.Damhof - 2011 Infomagazine BI community, The next generation EDW published in three papers (2008,2009) by R.D.Damhof

There is a catch in the data- or even the software `world'² though; even if we embark on a monumental requirements gathering process where we interview all people involved and write down all quality attributes, we would still not have a complete requirements specification that covers the quality demands of our business users (simply because they themselves do not fully comprehend them³). But quite apart from that; such an elaborate process simply isn't attainable.

So, is this a dead end street? No. Embrace that quality is relative and requirements can only be fixed by the time the user needs the data. Context - as we use the word in this paper - is a function of the person involved at a specific moment in time with the quality attributes he desires. And these contexts change all the time.

With dimensional modelling from Dr. Kimball [2], the industry made great strides in accommodating these uncertainties. However, dimensional modelling still results in solidifying a context, for a particular point in time, for a particular group of users and offering a particular set of quality attributes. For dimensional modelling, a change in context typically leads to new (sometimes, slightly altered) data products that need to be designed, built, tested, deployed and maintained.

There is a huge need for more agility of data to accommodate these continuously changing contexts. New technology is of course beneficial; in-memory business intelligence tooling, massive parallel data processing capabilities, or data virtualization are all viable enabling technologies. But they are no silver bullets for offering this agility. Use of new technologies does not free us from proper software design. This paper addresses a software design perspective to offer the required agility.

² As opposed to freezing specification/requirements of a manufactured (physical) product

³ Which is especially true for datasets where the future use of the data is - by definition - not fully known

Identifying the problems

Figure 1 represents a conceptual model of a purchasing & sales system, showing the most relevant entities and their relationships.

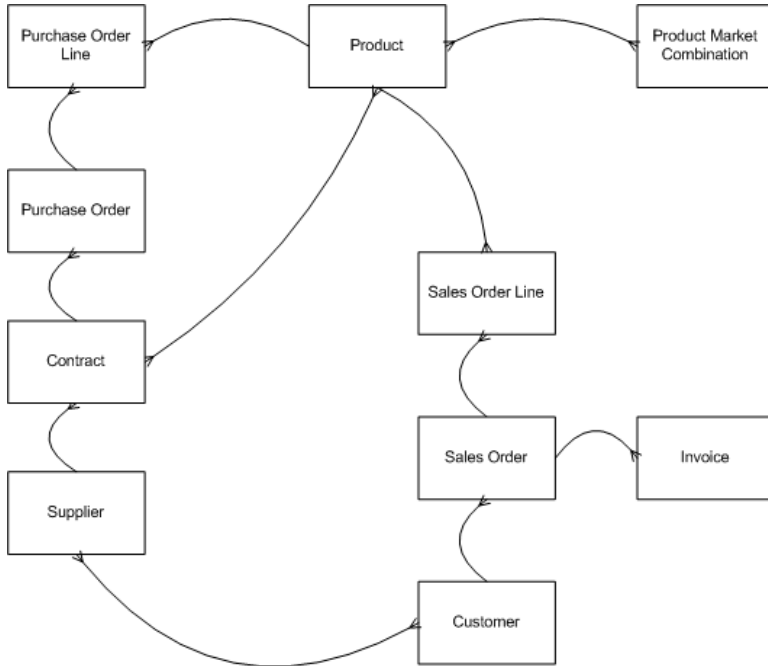


Figure 1: Conceptual model purchasing & sales system

As an example for further examination, let's look at the Purchasing part of the model above and construct a dimensional model according to Dr. Kimball (Figure 2).

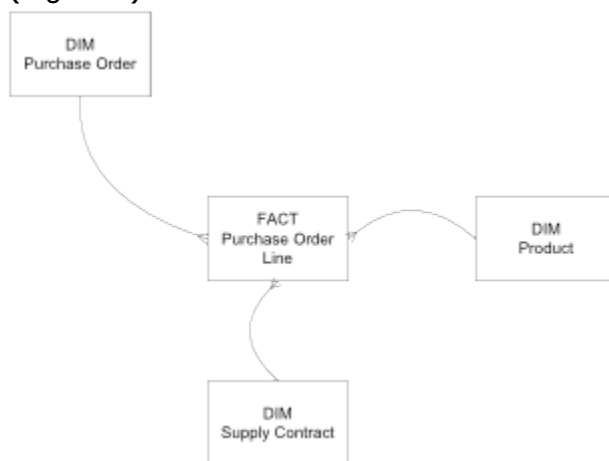


Figure 2: Dimensional model of the purchasing part of the conceptual model

A few problems with this model can be identified and prohibit the needed agility:

1. There is a many-to-many relationship between Product and Product Market Combination. If you need to include the Product Market Combination, it is required to bridge this many-to-many relationship. The same holds for the many-to-many relationship between Product and Supplier (see the conceptual model on the previous page: there are two relationships between product and supplier: one via the Purchase Order Lines, and one directly)
2. Even after solving all relationship problems, this model is still rather inflexible. Dr. Kimball allows for dimension history using type # SCDs⁴, allowing users to see the current and/or historically correct dimension state. Now, suppose the company needs to report all last year's purchase orders with Supplier information as of the 31st of December. That is not necessarily the current or historically correct state of the dimension.
3. Hierarchies exist in many different varieties, and tend to change over time. Embedding hierarchies in dimensions limits flexibility (we need to identify parent and child levels when designing the dimension) and restricts the available history to the SCD-structure of the dimension (see problem #2).
4. When communicating with business users, another problem pops up: Kimball divides the model into Facts and Dimensions. The Order Amount is a fact, not a dimension. The Supplier VAT Registration Number is a dimension, not a fact. So...the VAT Registration Number is *not* a fact? It is registered in an administrative system, so it *must* be a fact. Everything in a well-built data warehouse is a fact; otherwise the data could not have entered the data warehouse. The distinction between facts and dimensions is - from a business user's perspective - a superfluous one and should be avoided.
5. The concept of factless facts makes communications with business users even more problematic. Let's try to explain this concept: With factless facts, we store something into a fact table, which is not a fact in the fact-centric modeling technique used, and call it a factless fact - still with me? You do not understand? How do you think a business user feels?

⁴ Slowly Changing Dimensions

Lessons learned:

- In practice, many relationships are many-to-many, and they invariably require a solution;
- There are more potentially useful dimension states than the current and historically correct state;
- In communicating with business users, the terminology with regards to facts and dimensions is confusing.

But there are some more snags. Lets take a closer look at Customer and Supplier. Suppose there is a need to report or analyze only on purchase orders for Suppliers that are Customers as well. In the classic star model, we would have modelled both Customer and Supplier as separate dimensions. Then, how can we report on orders of Suppliers, which are Customers? We would need some kind of link between the Suppliers and Customers.

Looking at the same requirement differently, we want to be able to report on all purchase orders for our Customers. In other words: the Supplier is a link between the Customer and our purchase order. And it might even get more difficult when we want to restrict our question to specific time-frames (Suppliers which were customers *in the past year*) or product lines (Suppliers which were customers of our premium products). Products, Suppliers and Customers, all modelled as dimensions in the classic star model now seem to play the role of "link to other entities", depending on the context.

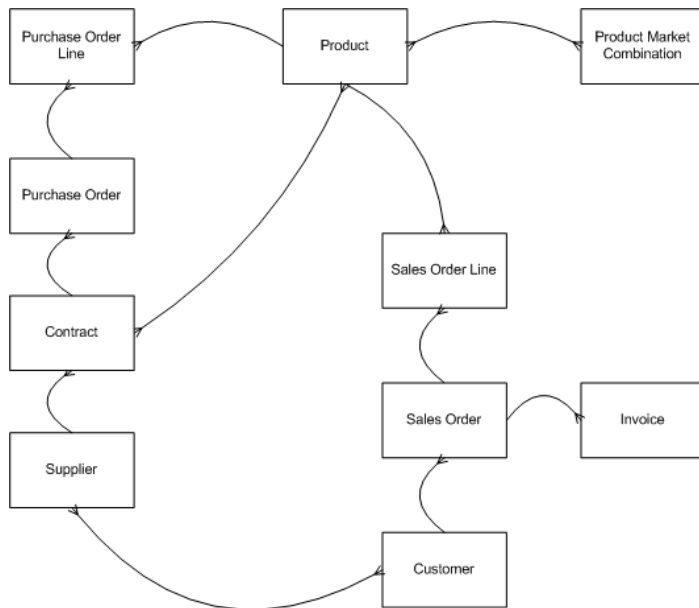
Lessons learned: everything is linked and nothing stands on itself. Dimensions only appear at the outer edges of our business model. Dimensions are the points where we choose to look no further. If we were to capture the whole world into our model, hardly any dimensions would remain. Ultimately, everything is a linkage of other things, until we reach the tiniest known particle.

So, we have seen that facts are a confusing concept, and that dimensions are a self-imposed limitation. Also, we have seen that everything is linked, in time-dependent, context-dependent ways. Many-to-many relationships are the rule, not an exception.

Addressing the problems

Confronted with these problems and limitations, the quest for a better solution begins. Since entities are related to each other in various ways, we need a model to discuss the relevant relationships for each context. Again, the starting point is the overall model, which shows all entities and relationships:

Back again to our conceptual model. This model shows how all entities are related, without giving any judgement/classification of the role of the entities



or their relationships. To express that this model gives an overview of how all entities are related to each other, we will call this model the Interspective Model (Figure 3).

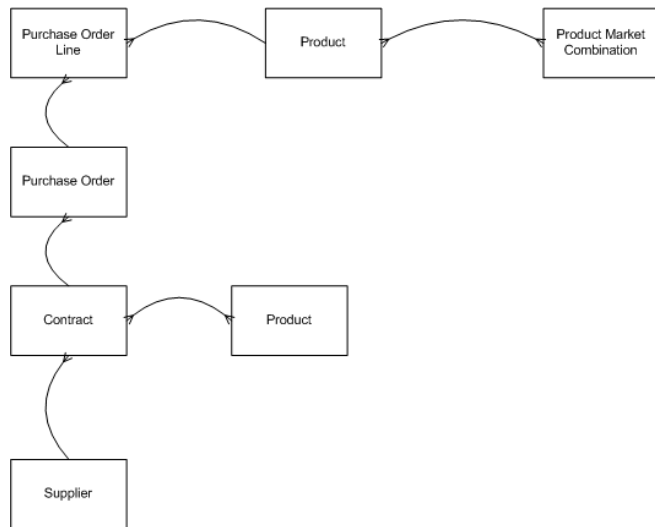
Let's take a closer look at the relationships. How do we determine which weight to give to the "many" side of the relationship. The answer: "Well, it depends...on the context."

Figure 3: The interspective model

So it seems that a detailed model of linked entities can be built only within a specific business context. In other words, we should be able to decompose the complex Interspective model into contexts.

Starting with the Interspective model, we take all entities, which are relevant for a specific context, including all relevant relationships, and use those as a starting point for discussions with business experts.

For example, let's take the sales context. The story of this context is that the business wants to see all sales orders, with information about customers, and the products sold. The relevant entities and relationships in this context are:



Now that we have limited ourselves to one context (note: now we *choose* to look no further, dimensions may *appear* to be present), the model becomes much easier to read (Figure 4).

Figure 4: The sales context of the interspective model

This model is simple enough to be used in discussions with business experts. During discussions with business experts, now or at any moment in the future, the model might grow or change. That is no problem, in fact it is what the model was designed for:

The model is not a *product*, just a guide in the design *process*.

We can now try to identify a central entity for the context. For anyone familiar with the Kimball methodology: this entity *might* look very much like a classic “Fact” table. In our example, the most natural entity is the sales order line, which gives the lowest level of detail of purchase orders. From now on, we will call this the *Context Entity*.

Once we have presented the initial model to the business users, and possibly made some initial additions or changes, the next step is to define at least one business rule for each of the relations: how do the entities relate in this specific context.

For every relationship, there is always at least one business rule, which must be specified explicitly. If the business experts cannot specify the business rules, specifying “we don’t know, yet”, is good enough for the first session, as long as we explicitly specify this conclusion and start a process to solve the unknowns. Knowing what we don’t (yet) know is valuable knowledge as well. And we are designing for agility, so changing or growing knowledge shouldn’t be an issue.

To avoid duplicates at query time, there should only be many-to-one relationships starting from the Context Entity. Going from sales order line, via the relationships to the far ends of the model, we must take care of any one-to-many or many-to-many relationships found. This situation is present at multiple locations in the model:

- Many products to many product-market combinations;
- One sales order to many invoices.

For each of these relationships, we need to know how the business users want to handle them. So, we need to discuss the model with a business expert, and ask specifically about the problematic relationships: “when there are more...what do you want to see?”

In our example, for the relationship between sales order and invoice, the business expert indicates that multiple invoices might be sent for an order, for instance in case of corrections or late payment. The business only wants to see the most recent version of the history for each invoice.

For the relationship between products and product –market combinations (PMCs) it turns out, that the PMCs are related to the customers: the customers are in a specific market segment, and therefore the PMC to which a specific product in a specific sales order should be linked, can be determined by looking at the customer.

So, while discussing this with the business expert, we adjust the model:

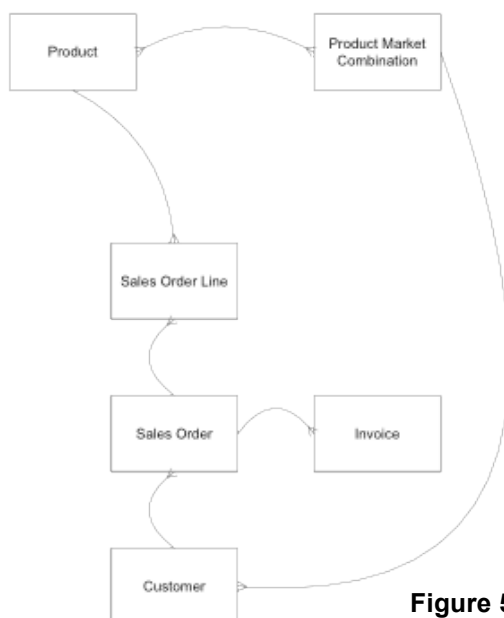


Figure 5: Adjusting the model as we speak

The business wants to be able to see the sales PMC on the time of ordering, as of now (the most recent version of the historical states), and as of the 1st of January of the current year.

Another challenge becomes visible: tracking history. Attributes of dimensions tend to change over time. In the Kimball model, this is solved by introducing type # SCDs.

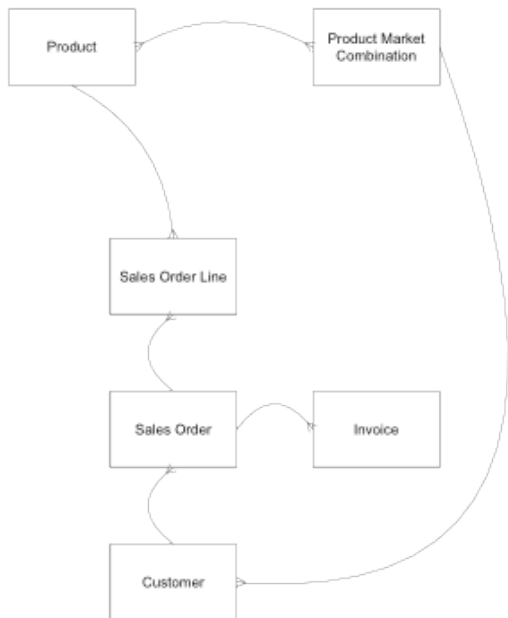
These SCDs however can only handle two historical states:

- Current state;
- Historically correct state at the time when the fact happened.

In real business situations, more versions of history might be needed, e.g. the state as of the 1st of January. In other words, if the data model should be able to handle various versions of the history flexibly, we cannot limit ourselves to the classic SCDs. To be flexible, we need to register *all historical states*, indicate the validity with a start date and end date, and be able to select *any* point in time for queries.

Technical implementation

Take another look at our last conceptual model:



Although it might represent the business very well, something, which looks like a classic star scheme, is much easier for querying purposes.

So, ideally we're trying to implement a model, which can represent both the business model (see model above) and a query-optimized model like depicted in

Figure 6

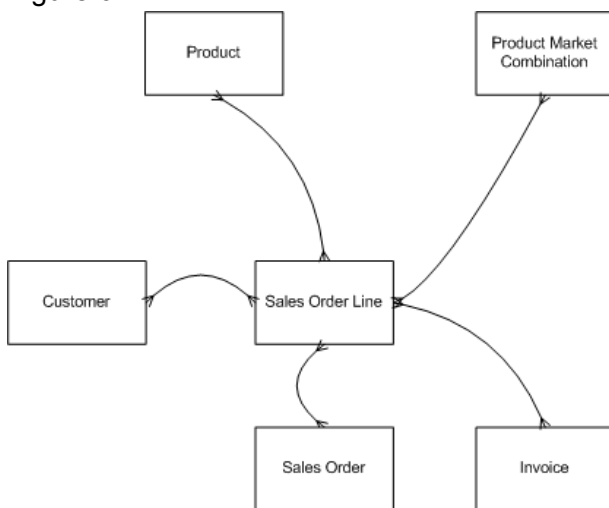


Figure 6: Query optimized model

What's the difference between the two models? In the query-model, we have directly linked all entities to the context entity: the context determines the meaning of everything else. But the query-optimized model is not a model of fact tables and dimension tables. The query-optimized model was derived

from the business perspective model, which in turn was derived from the Interspective Model.

It might be that entities, which appear to be a dimension in this context, play a different role in another context. The same goes for the Context Entity. The only thing we know is that everything is a linkage of other things, and that we should model for flexibility. For the implementation of these linkages, we use so-called bridges, loosely inspired by the bridge tables described by Dr. Kimball.

When implementing these bridges, we only specify how they are related to other bridges, and what their identifying and descriptive properties are. We do not tell which role a bridge plays. The exact role of a bridge is determined by placing the bridge in a specific context.

Therefore, we start off by implementing every entity as a bridge, even the entity we marked earlier on as Context Entity.

So, can we now start to design tables? No....and here is why:

A real-world bridge itself is an ultimate example of the concept of bridges: when looking closer at a bridge, you will see that is actually a relationship of elements (bars, chains, bricks,...) connecting to each other (that is: the bridge itself is a set of bridges). Each of these elements, upon further inspection, will turn out to be a relationship of elements as well....and so on...until we reach the tiniest known particle.

That is a good lesson for our modelling: be prepared that a bridge might actually turn out to consist of bridges itself. Which level of detail is relevant is determined by context and perspective.

That is why we call the implemented bridges "bridge views". And that is how you should consider implementing them: only materialize them when you need to, for example because the size of your data demands this, the scale of your data vault, or the complexity of your business rules..

In other words: we initially model all entities as bridge views.

Every bridge view has at least columns for:

- Start date of functional validity of the record;
- End date of functional validity of the record;
- ID, likely to be sourced from a HUB_ID in the Data Vault;
- Business identifier.

Likely, there are IDs of other bridges within our context as well. For some of the entities, we might already know of relationships to other entities that are not yet part of our context. Only if we are absolutely sure about the functional meaning and cardinality of the relationship, can we implement them in our bridges. We can always easily add them later. Adding them now without needing them creates risk, not value.

Having created all bridge views, we now need to combine them to be able to query them within a specific context. For this, we create a context view. This context view contains at least:

- a start date + end date *or* a "valid on" date;
- IDs of all the bridges in our context model.

Agility in action

Figure 7 depicts the resulting set of views:

- Bridge views, with start date and end date;
- Context views, with at least one date stamp (e.g.: "valid on").

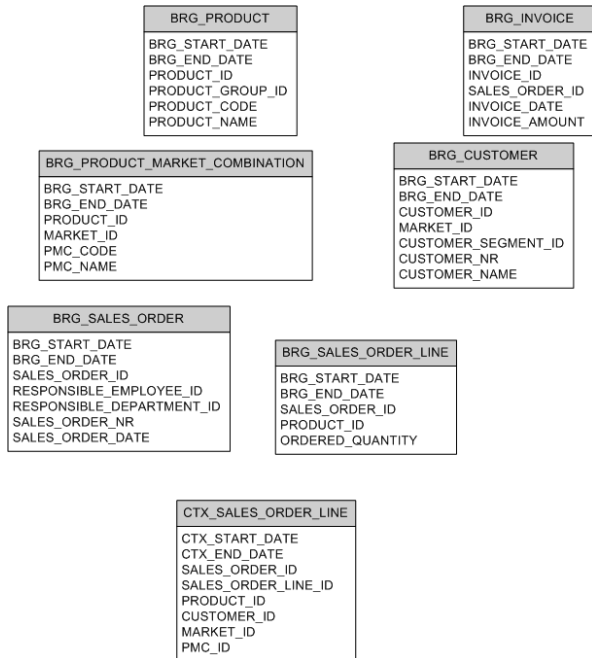


Figure 7: Bridge - and Context views

With these views, we can build the data model for queries, based on the contextual perspective model (Figure 8).

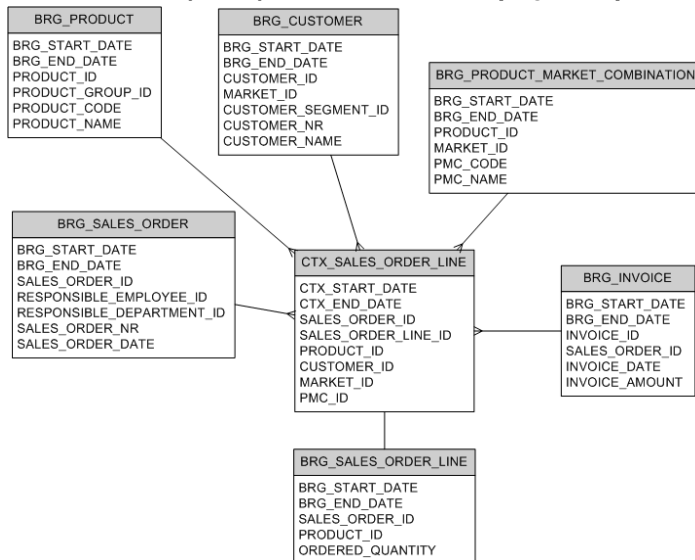


Figure 8: Query optimized model, based on contextual perspective

As can be seen from the model above, the Context View is the centre of the model. This view links one-on-one to the Sales Order Line Bridge View.

All other bridge views are joined directly to the Context View, based on the bridge view ID, which is present in both the Context View and the Bridge View. However, this is not enough: since the bridges contain all historical states, we also need to specify on which historical state we want to join.

Overall, there are three types of historical states to choose from (and one anomaly – Lost Time):

1) *Point in time*

In the Sales Order Line context view, there are technical IDs referring to the related bridge views. We can now create joins between the context view and the bridge views using these IDs. However, since bridge views track all history, and the context determines the perspective on this history, we must also specify the timeline for each join.

For example, the join between CTX_SALES_ORDER_LINE and BRG_CUSTOMER might be:

```
CTX_SALES_ORDER_LINE.CUSTOMER_ID=BRG_CUSTOMER.CUSTOMER_ID  
AND  
CTX_SALES_ORDER_LINE BEGIN_DATE >= BRG_CUSTOMER.END_DATE and  
CTX_SALES_ORDER_LINE END_DATE <=BRG_CUSTOMER.BEGIN_DATE
```

2) *Current time*

This join immediately shows the flexibility of the model. Suppose, we also want a “current” view of the customer. We can then add an alias of the BRG_CUSTOMER to the model, and join it to the Context View:

```
CTX_SALES_ORDER_LINE.CUSTOMER_ID = CURRENT_CUSTOMER.CUSTOMER_ID  
AND  
CURRENT_CUSTOMER.BEGIN_DATE < sysdate  
AND CURRENT_CUSTOMER.END_DATE > sysdate
```

3) *Perspective-specific time*

So far, nothing new. We can do this with SCDs as well. But what if we learn that there is an issue with the customer status. Customers might be

“Bronze”, “Silver” or “Gold”. Placing an order might increase the customer status, which becomes effective on the order date...

...and the requirement is to show what the customer status was *just before* the order was placed.

Now, if we use the joins specified earlier, we will see the potentially already updated customer status. So, we need the customer status on the day *before* the order date.

With classic SCDs, we would now have to ask an ETL⁵ expert to create another copy of the customer status, based on the date of ordering. But he cannot do that, since Customer is not dependent on Order. It is the other way around.

Since the perspective model does not enforce time-lines or one-way dependencies, we can again re-use the BRG_CUSTOMER by creating an alias, and join it with the Context View:

```
CTX_SALES_ORDER_LINE.CUSTOMER_ID = PREVIOUS_CUSTOMER.CUSTOMER_ID  
AND  
CTX_SALES_ORDER_LINE.VALID_ON_DATE - 1 between  
PREVIOUS_CUSTOMER.BEGIN_DATE and PREVIOUS_CUSTOMER.END_DATE
```

Remember that everything is just a link to other things? Customer is related to invoice as well, so we can add a CUSTOMER_ID in the BRG_INVOICE. We can easily join the BRG_CUSTOMER to the BRG_INVOICE, allowing us to see another timeline: the Customer status on the invoice date. Now, we are glad that we didn't *choose* upfront to look no further, and created a flexible bridge instead of a dimension.

⁵ Extraction, Transformation, Load

4) *Lost in Time*

The flexible timeline of the contextual perspective model might lead to problems when data is present, but not at the right moment (e.g.: the customer record is valid from 1-1-2009, but the first order was seen on 12-12-2008).

For the timeline problem, we can adjust the timeliness of the bridges, by making the first occurrence of a customer valid from a date in early history (e.g. 1-1-0000), and making the last occurrence of a customer valid until a date in the far future (e.g. 31-12-9999). We add these dates as separate timeline to the BRG_CUSTOMER. This allows us to see the functionally true start date and end date of the customer as well. That, in turn, makes it easy to create data quality reports:

- Show me orders placed before the customer was known, including lead (how many days between order date and first occurrence of customer);
- Show me orders for products, which were no longer in our portfolio on the date of ordering, including the lag (how many days between product-end and order-date).

Closing note

The bottom line: when using the approach described above, without any ETL, without creating any additional copies of the data, we can enhance the perspective, resulting in very short time-to-market for new user requests. Also, for data analysts, the bridge views provide a method of querying data in a flexible way. This frees them from having to use the hundreds of tables in the data vault, and/or frees data vault experts from creating all kinds of specific data sets for the analysts. Since we re-use the IDs already present in the data vault, it is even possible to combine data from the bridges/contexts with data from the data vault in a single query.

The above approach to modelling datasets is a prime candidate to flourish even more when used with technology like data virtualization, in-memory processing and massive parallel processing capabilities.

However, technology does not take precedence over robust software design. The goal is straightforward: addressing the need to answer the questions of

tomorrow in a way that is cost effective, predictable, highly standardized and with a short time-to-market. If that isn't quality.....

- [1.] Quality Software Management, volume 1, Gerald M. Weinberg, 1991
- [2.] The complete guide to dimensional modeling (2nd edition), R.Kimball, M.Ross, 2002