Your data is changing, there must be an error?

When discussing data topics with business consumers of the data I always lead with explaining that, to me, data is just 'stuff'. Data usually doesn't really make much sense to me at first. Data are the recorded events generated by business processes. If we don't properly understand these business processes, this context, it can be very hard to understand what meaning or interpretation we can draw from these raw events. You only know what you know.

My guiding principle is to always enable frequent redesign of the structures we use to prepare data for consumption, such as a Data Warehouse. The more we learn, the better our interpretation of the raw events will become. Therefore, having the ability to easily change data models and corresponding load processes is important. It allows us to embed our progressive understanding of the subject matter domain.

This principle of redesign (refactoring) of the data models requires the loading processes to be completely deterministic. A deterministic process means that given the same inputs, the output of the process or calculation will always be the same.

In a fully metadata driven solution, and by using deterministic patterns, it is possible to completely drop or truncate the full Data Warehouse – the Presentation Layer (Data Marts) but also the core Data Warehouse – and rebuild these to look exactly as they were before.

Of course, to enable this you need an underlying Persistent Staging Area (PSA) that captures and stores the raw events as and when they were made available.

This is the mindset of Virtual Data Warehousing. If it is possible to fully generate all the loading processes (i.e. Extract-Transform-Load, or ETL) *and* use fully deterministic patterns then it is in principle possible to generate database views that simulate the entire Data Warehouse as a giant schema-on-read.

This does not mean you *have* to use views (performance!), but the fact that *can* you do this means you have options to choose which areas of the solution to materialise (persist) so you can achieve an optimal mix of performance and flexibility. More importantly, using views proves having the ability to refactor the solution over time. If you can simulate your Data Warehouse using views you know your patterns are truly deterministic.

In this paper I will apply this concept to some of the more complex topics of Data Warehousing: bi-temporality and backdated adjustments. This comes into play in the example below.

Imagine it is November and you are showing the sales figures from June earlier that year. The sales amount over June is $120k. A month later, in December, you again sit down and open the sales figures report to look at the June figures again. This time, however, they are $125k. How can this happen, the consumer asks?

In Data Warehousing many things come down to (establishing) trust in the information that is provided. A common response is that 'the data is wrong'. How can you otherwise explain that figures from the past change?

One way of establishing trust is ensuring that deterministic patterns are implemented. These can be used to explain why data behaves the way it does to consumers. This shows that, whatever you do, the output always follows the rules. Demonstrating this is a great way to build some confidence around the data management.

Considering the above, a deterministic process that can show how backdated adjustments (late-arriving data) are being handled can be a way to start explaining why the June figures are different (but still correct) depending on *when* you look at them.

With this understanding you can start discussing what decisions can be made on how these changes in figures can be presented.

Many thanks to various professionals for discussing, providing input and reviewing this paper and examples. Especially Scott Diprose for coming up with the Outer Apply / Union code and Dirk Lerner and Christian Haedrich for reviewing.

Please make note of Dirk Lerner's training materials on bi-temporal matters, they are worthwhile if you want to learn more on this topic. Information on this is available here.

Merging time-variant data sets

The intent of this paper is to explain the mechanics of a deterministic process to display (the impact of) backdated adjustments. It is building on the foundation provided in the paper 'A pattern for Data Mart delivery', which outlines the principles around merging the data in multiple time-variant (historised) tables.

'A pattern for Data Mart delivery' also introduced the concept of using start- and end-date / time parameters to enable load windows – the Table Valued Function example. This is an elegant way to 'time travel' as mentioned later in this paper.

Using the load window, you can present the Data Mart both as a database view (full-length load window) or use this to facilitate incremental updates to a physical Data Mart table (using the load window parameters in ETL).

Either way, the view that represents the Data Mart would display the results *exactly the same* as if would have been loaded incrementally over time – a deterministic process.

This ability to present results in a single pass (view) as if they were loaded incrementally is the key requirement for the approach outlined in this paper. We will apply this concept to backdated adjustments to prove that we can implement a deterministic approach that displays the correct history including backdated adjustments – as if they were loaded incrementally over time.

Only when this requirement is met we can implement a 'Data Warehouse Time Machine', travel back in time and explain consumers what happened, when it happened and why, and what to do about it.

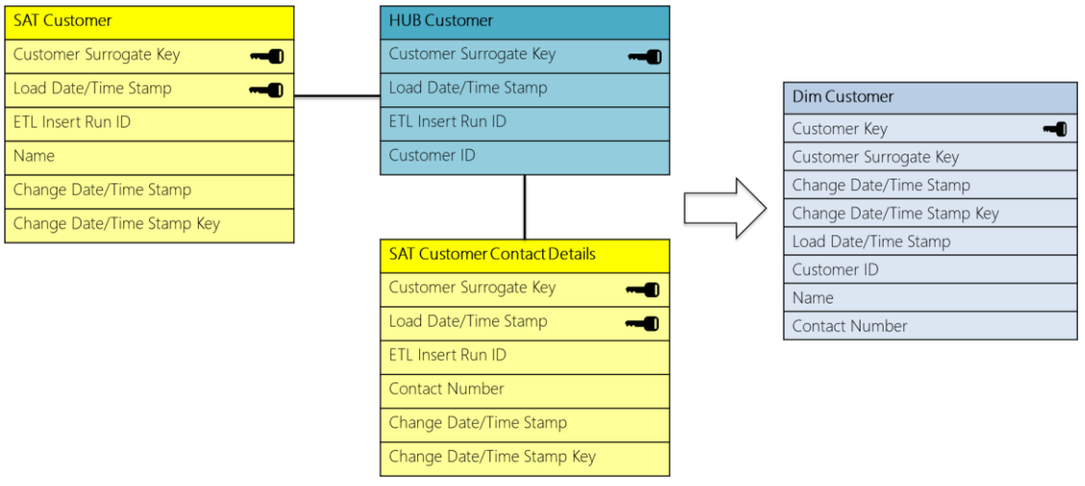
Combining customer details

The techniques related to merging time-variant data sets, as outlined in the 'A pattern for Data Mart delivery' paper, are not limited to any particular modelling technique and can be used for combining many different types of historised data sets.

To explain the impacts of backdated adjustments I have prepared an example that shows what happens when two Data Vault Satellite (time-variant) entities are combined into a single Dimension object – a common use case.

This example contains a Customer Satellite that contains the 'Name' of the Customer. The second Satellite contains additional contact details such as the 'Number'. Both tables, as well as their Core Business Concept ('Hub') table are merged to represent a single Dimension table called Dim Customer. For clarity, please assume that both Satellites are loaded from different sources (i.e. not from the same Staging Layer table).

For brevity, I have omitted the Hub Customer from the examples. For the same reason, I added the Business Key directly in the Satellite.



There are two time attributes of note in both the Satellite tables, these are the:

- Load Date/Time Stamp (LDTS), the recorded arrival time in the Data Warehouse environment. To facilitate deterministic processing, this value is set in the Staging Layer and is inherited to the Satellites.
- Change Date/Time Stamp (CDTS), the agreed (conformed, standardised) timeline that represents the business, or functional, effective date/time. In practical terms this value is populated from a business effective date attribute provided by the source application.

The CDTS represents the timeline that business users / consumers are familiar with. These are the effective dates in the operational system.

A useful way to think about this is to consider what users of operational systems see on their screens when they do their work from day to day. If they open a screen in their application to query customer contact details and see the most recent (effective) customer contact number in the system, we must be able to present this as such in the Data Warehouse as well.

This is relevant, because this is not necessarily the information that was *received* last by the Data Warehouse (the LDTS axis).

Equally important: the CDTS timeline contains the date range that is used for joining tables 'in time' as they represent the reality that consumers work with in their systems.

Incidentally, this is a major reason why the LDTS is not suitable as a timeline for information delivery – something that is explained in detail in the 'A pattern for Data Mart delivery' paper.

We really should not join Data Warehouse tables using the LDTS effective and expiry dates.

Also note the 'Change Date / Time Stamp Key' attribute. This is a uniquefication of the CDTS used for the joining in the pattern. The CDTS can be constructed as a concatenation of the CDTS, the ETL Execution Instance ID and the Row ID within that unique ETL execution.

Or, it can be constructed as the concatenation of the CDTS, LDTS and Row ID. Either way, it is always numeric, unique and in sequence.

While the CDTS value represents the timeline, the 'Key' equivalent is used for the actual joining to other tables to retrieve the correct information at that point in time.

Having a dependable unique value helps to reduce duplicates in the result caused by data quality issues. We are using 'source' values, so are exposed to the quality of data we receive.

The CDTS can also be used as join key for bi-temporal evaluation. More on this later.

The data arrives

We can now start looking at the changes that are incrementally presented to the Data Warehouse, and – over time – populate the two Satellite tables and subsequently the corresponding Dimension table.

In this example, the first change that is presented to the Data Warehouse populates the Customer Satellite. We now know that the Customer only known as Customer ID 'CUST_4' is in fact called 'Jonathan'.

This information was received by the Data Warehouse at 2010-04-01 (LDTS axis) and is effective dated in the operational systems as of 2010-02-01 (CDTS axis).

This results in the creation of one record in the Dimension table. There are no records in the other Satellite (Customer Contact Details), which contains the information on the 'Number'. Therefore, the value for the 'Number' is unknown (NULL) at this point in time.

In the example below, the number (1) on the left of the LDTS displays the order of arrival (the LDTS axis), whereas the number to the right of the Name (also 1 currently) shows the order of effective dating following the CDTS axis.

Sat Customer

	LDTS	CDTS	ID	Name	
1	2010-04-01	2010-02-01	CUST_4	Jonathan	1

Dim Customer

Arrival #	ID	Name	Number	LDTS	CDTS
1	CUST_4	Jonathan	NULL	2010-04-01	2010-02-01

The second change that arrives in the Data Warehouse contains information about the Number. This change populates a row in the Customer Contact Detail Satellite, and results in the creation of a second row in the Dimension table.

This information arrived on 2011-01-01 and is effective dated from the same date onwards. In the Dimension.

Now we know that, as per 2011-01-01, Jonathan's number is '123'.

Sat Customer

	LDTS	CDTS	ID	Name	
1	2010-04-01	2010-02-01	CUST_4	Jonathan	1

Dim Customer

Arrival #	ID	Name	Number	LDTS	CDTS
1	CUST_4	Jonathan	NULL	2010-04-01	2010-02-01
2	CUST_4	Jonathan	123	2011-01-01	2011-01-01

Sat Customer Contact Details

	LDTS	CDTS	ID	Number	
2	2011-01-01	2011-01-01	CUST_4	123	2

The third row that is captured in the Data Warehouse leads to a change in the 'Name' content. We received this information at 2012-02-02, but it is effective dated as per 2012-01-01. The name of customer CUST_4 is now 'John', where it used to be 'Jonathan'.

This is reflected in the Dimension table as a new row that highlights the name change from 'Jonathan' to 'John'. The 'Number' is still effective as '123' so this value inherited into the new Dimension row.

Sat Customer

	LDTS	CDTS	ID	Name	
1	2010-04-01	2010-02-01	CUST_4	Jonathan	1
3	2012-02-02	2012-01-01	CUST_4	John	3

Dim Customer

Arrival #	ID	Name	Number	LDTS	CDTS
1	CUST_4	Jonathan	NULL	2010-04-01	2010-02-01
2	CUST_4	Jonathan	123	2011-01-01	2011-01-01
3	CUST_4	John	123	2012-02-02	2012-01-01

Sat Customer Contact Details

	LDTS	CDTS	ID	Number	
2	2011-01-01	2011-01-01	CUST_4	123	2

The fourth change to the data is again a change in the 'Name' value, so again leads to the creation of a new record in the Customer Satellite. This time, 'John' is now called 'Jon'. This information was received at 2013-01-01 and is effective as of 2013-01-01.

The 'Number' value is still in effect so is again inherited into the new Dimension row.

Sat Customer

	LDTS	CDTS	ID	Name	
1	2010-04-01	2010-02-01	CUST_4	Jonathan	1
3	2012-02-02	2012-01-01	CUST_4	John	3
4	2013-01-01	2013-01-01	CUST_4	Jon	4

Dim Customer

Arrival #	ID	Name	Number	LDTS	CDTS
1	CUST_4	Jonathan	NULL	2010-04-01	2010-02-01
2	CUST_4	Jonathan	123	2011-01-01	2011-01-01
3	CUST_4	John	123	2012-02-02	2012-01-01
4	CUST_4	Jon	123	2013-01-01	2013-01-01

Sat Customer Contact Details

	LDTS	CDTS	ID	Number	
2	2011-01-01	2011-01-01	CUST_4	123	2

The fifth change as presented to the Data Warehouse is received at 2014-01-01. CUST_4 has been updated to a new 'Number' of '999' which is effective from 2014-01-01 onwards, and therefore supersedes the 'old' value of '123'.

A new row is created in the Dimension table to reflect this. At 2014-01-01 the 'Name' value is 'Jon', so this is inherited into the Dimension row.

Sat Customer

	LDTS	CDTS	ID	Name	
1	2010-04-01	2010-02-01	CUST_4	Jonathan	1
3	2012-02-02	2012-01-01	CUST_4	John	3
4	2013-01-01	2013-01-01	CUST_4	Jon	4

Sat Customer Contact Details

	LDTS	CDTS	ID	Number	
2	2011-01-01	2011-01-01	CUST_4	123	2
5	2014-01-01	2014-01-01	CUST_4	999	5

Dim Customer

Arrival #	ID	Name	Number	LDTS	CDTS
1	CUST_4	Jonathan	NULL	2010-04-01	2010-02-01
2	CUST_4	Jonathan	123	2011-01-01	2011-01-01
3	CUST_4	John	123	2012-02-02	2012-01-01
4	CUST_4	Jon	123	2013-01-01	2013-01-01
5	CUST_4	Jon	999	2014-01-01	2014-01-01

When the sixth row arrives, something interesting happens.

This row arrives at 2014-02-01, so still in sequence along the LDTS axis - guaranteed by the LDTS configuration of course. However, the CDTS is set at 2011-12-01 which means that this row is in fact a backdated adjustment.

The CDTS order places it 3rd in the order of arrival along the CDTS axis when considering all known changes from both in scope tables. It is dated *after* 2011-01-01 but *before* 2012-01-01 in the CDTS timeline.

Sat Customer

	LDTS	CDTS	ID	Name	
1	2010-04-01	2010-02-01	CUST_4	Jonathan	1
3	2012-02-02	2012-01-01	CUST_4	John	3
4	2013-01-01	2013-01-01	CUST_4	Jon	4

Sat Customer Contact Details

	LDTS	CDTS	ID	Number	
2	2011-01-01	2011-01-01	CUST_4	123	2
5	2014-01-01	2014-01-01	CUST_4	999	5
6	2014-02-01	2011-12-01	CUST_4	345	3

Dim Customer

Arrival #	ID	Name	Number	LDTS	CDTS
1	CUST_4	Jonathan	NULL	2010-04-01	2010-02-01
2	CUST_4	Jonathan	123	2011-01-01	2011-01-01
3	CUST_4	John	123	2012-02-02	2012-01-01
4	CUST_4	Jon	123	2013-01-01	2013-01-01
5	CUST_4	Jon	999	2014-01-01	2014-01-01
6	CUST_4	Jonathan	345	2014-02-01	2011-12-01

The order of arrival according to the CDTS timeline needs to be updated with this new information.

Sat Customer

	LDTS	CDTS	ID	Name	
1	2010-04-01	2010-02-01	CUST_4	Jonathan	1
3	2012-02-02	2012-01-01	CUST_4	John	~~3~~ 4
4	2013-01-01	2013-01-01	CUST_4	Jon	~~4~~ 5

Sat Customer Contact Details

	LDTS	CDTS	ID	Number	
2	2011-01-01	2011-01-01	CUST_4	123	2
5	2014-01-01	2014-01-01	CUST_4	999	~~5~~ 6
6	2014-02-01	2011-12-01	CUST_4	345	3

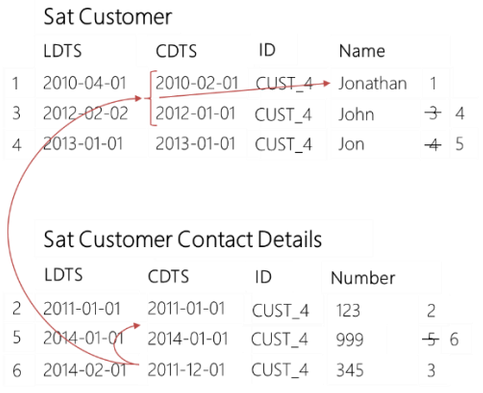
Dim Customer

Arrival #	ID	Name	Number	LDTS	CDTS
1	CUST_4	Jonathan	NULL	2010-04-01	2010-02-01
2	CUST_4	Jonathan	123	2011-01-01	2011-01-01
3	CUST_4	John	123	2012-02-02	2012-01-01
4	CUST_4	Jon	123	2013-01-01	2013-01-01
5	CUST_4	Jon	999	2014-01-01	2014-01-01
6	CUST_4	Jonathan	345	2014-02-01	2011-12-01

A new record will be created in the Dimension table using the 'Number' value from the new row in Customer Contact Details, but it will retrieve the value for the Name that was active *at the point-in-time of the back-dated adjustment.*

If you look at it from this perspective, the name change from Jonathan to John *hasn't happened yet*.

With the wisdom of the present we know what this *will* occur in the future. But even though we know the name change will happen with the information available, *at that (functional) point in time* (2011-12-01) the name was still set as Jonathan.

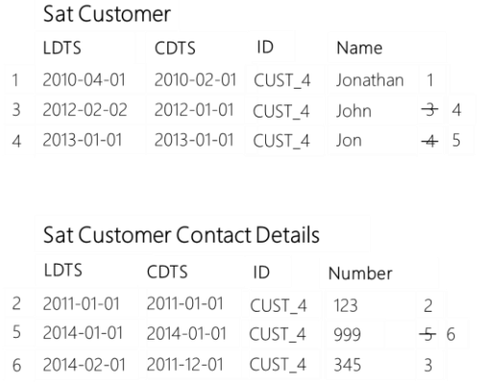


Dim Customer

Arrival #	ID	Name	Number	LDTS	CDTS
1	CUST_4	Jonathan	NULL	2010-04-01	2010-02-01
2	CUST_4	Jonathan	123	2011-01-01	2011-01-01
3	CUST_4	John	123	2012-02-02	2012-01-01
4	CUST_4	Jon	123	2013-01-01	2013-01-01
5	CUST_4	Jon	999	2014-01-01	2014-01-01
6	CUST_4	Jonathan	345	2014-02-01	2011-12-01

This is not all that needs to happen, and this is where the complexity of backdated adjustments really comes into effect.

At this stage, two additional rows are inserted into the Dimension table to reflect *what would have happened* if we would have received the backdated adjustment in the regular sequence (normal order along the CDTS axis).



Dim Customer

Arrival #	ID	Name	Number	LDTS	CDTS
1	CUST_4	Jonathan	NULL	2010-04-01	2010-02-01
2	CUST_4	Jonathan	123	2011-01-01	2011-01-01
3	CUST_4	John	123	2012-02-02	2012-01-01
4	CUST_4	Jon	123	2013-01-01	2013-01-01
5	CUST_4	Jon	999	2014-01-01	2014-01-01
6	CUST_4	Jonathan	345	2014-02-01	2011-12-01
6	CUST_4	John	345	2014-02-01	2012-01-01
6	CUST_4	Jon	345	2014-02-01	2013-01-01

Where are these extra rows coming from?

These are the rows that *would have been created* if this change was received at the point-in-time it is effective for (2011-12-01).

In other words; if we would have received this row at 2011-12-01, instead of 2014-02-01 as we have now, the normal change detection would have led to the creation of these two rows in the future.

We would not have known this back then, but we do know this was going to happen with the information we have received up to and including the present.

We are simply simulating what would have happened.

Let's try that out.

Receiving the data in the correct order

What would have happened if we would have received the data in the correct order?

If we simulate loading all the records in the 'correct' order of arrival, where the CDTS and LDTS follow the same sequence the result would look like this:

Sat Customer

	LDTS	CDTS	ID	Name	
1	2010-04-01	2010-02-01	CUST_4	Jonathan	1
4	2012-02-02	2012-01-01	CUST_4	John	4
5	2013-01-01	2013-01-01	CUST_4	Jon	5

Sat Customer Contact Details

	LDTS	CDTS	ID	Number	
2	2011-01-01	2011-01-01	CUST_4	123	2
3	2011-12-01	2011-12-01	CUST_4	345	3
6	2014-01-01	2014-01-01	CUST_4	999	6

Dim Customer

Arrival #	ID	Name	Number	LDTS	CDTS
1	CUST_4	Jonathan	NULL	2010-04-01	2010-02-01
2	CUST_4	Jonathan	123	2011-01-01	2011-01-01
3	CUST_4	Jonathan	345	2011-12-01	2011-12-01
4	CUST_4	John	345	2012-02-02	2012-01-01
5	CUST_4	Jon	345	2013-01-01	2013-01-01
6	CUST_4	Jon	999	2014-01-01	2014-01-01

The highlighted rows show the information that, in the previous example, we artificially created as a result of the backdated adjustment. In the normal order of processing they would have been detected and created as such in the Dimension table. These rows are created because of the changes occurring after 2011-12-01. (the row that in the main example represents the backdated adjustment).

But wait, this looks like there are *less* rows compared to the backdated adjustment example?

The image below shows that if we would have loaded the rows in the 'correct' CDTS order, we would have 6 rows. But using the same example with the backdated adjustment results in 8 rows so far.

Back-dated

Arrival #	ID	Name	Number	LDTS	CDTS
1	CUST_4	Jonathan	NULL	2010-04-01	2010-02-01
2	CUST_4	Jonathan	123	2011-01-01	2011-01-01
3	CUST_4	John	123	2012-02-02	2012-01-01
4	CUST_4	Jon	123	2013-01-01	2013-01-01
5	CUST_4	Jon	999	2014-01-01	2014-01-01
6	CUST_4	Jonathan	345	2014-02-01	2011-12-01
6	CUST_4	John	345	2014-02-01	2012-01-01
6	CUST_4	Jon	345	2014-02-01	2013-01-01

Sequential load

Arrival #	ID	Name	Number	LDTS	CDTS
1	CUST_4	Jonathan	NULL	2010-04-01	2010-02-01
2	CUST_4	Jonathan	123	2011-01-01	2011-01-01
3	CUST_4	Jonathan	345	2011-12-01	2011-12-01
4	CUST_4	John	345	2012-02-02	2012-01-01
5	CUST_4	Jon	345	2013-01-01	2013-01-01
6	CUST_4	Jon	999	2014-01-01	2014-01-01

Why are there more rows in the back-dated loading example compared to when the changes would have been received in sequence?

This is because, due to the backdated adjustment *arriving later* (i.e. the LDTS axis), these rows were created as changes because there was no way for them to know that in the (relative) future (e.g. at 2014-02-01) there would be a change which would have impacted their values.

In an incremental loading approach this is what happens. Explaining this to consumers is where time-travel comes in. More on this later.

The example completed

At 2020-01-01 the last change for this example arrives at the Data Warehouse. This change is effective dated at the same date/time, so follows the regular order of arrival considering both time perspectives.

The 'Number' value has been updated to '111', which leads to a regular new row in the Dimension table. The 'Name' value at that point in time for the functional timeline (CDTS) is still 'Jon', so this value is inherited in the Dimension record.

This row has been added to the example to make sure that, when we translate this concept into a view, we can limit the creation of additional rows caused by backdated adjustments up to the point that there is a regular change again. You don't want the additionally created rows going too far into the future.

Sat Customer

	LDTS	CDTS	ID	Name	
1	2010-04-01	2010-02-01	CUST_4	Jonathan	1
3	2012-02-02	2012-01-01	CUST_4	John	4
4	2013-01-01	2013-01-01	CUST_4	Jon	5

Sat Customer Contact Details

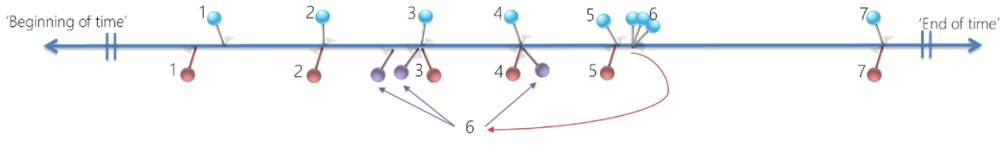
	LDTS	CDTS	ID	Number	
2	2011-01-01	2011-01-01	CUST_4	123	2
5	2014-01-01	2014-01-01	CUST_4	999	6
6	2014-02-01	2011-12-01	CUST_4	345	3
7	2020-01-01	2020-01-01	CUST_4	111	7

Dim Customer

Arrival #	ID	Name	Number	LDTS	CDTS
1	CUST_4	Jonathan	NULL	2010-04-01	2010-02-01
2	CUST_4	Jonathan	123	2011-01-01	2011-01-01
3	CUST_4	John	123	2012-02-02	2012-01-01
4	CUST_4	Jon	123	2013-01-01	2013-01-01
5	CUST_4	Jon	999	2014-01-01	2014-01-01
6	CUST_4	Jonathan	345	2014-02-01	2011-12-01
6	CUST_4	John	345	2014-02-01	2012-01-01
6	CUST_4	Jon	345	2014-02-01	2013-01-01
7	CUST_4	Jon	111	2020-01-01	2020-01-01

The resulting Dimension displayed in the image below (including timeline showing both time perspectives) provides for an interesting observation.

Arrival #	ID	Name	Number	LDTS	CDTS
1	CUST_4	Jonathan	NULL	2010-04-01	2010-02-01
2	CUST_4	Jonathan	123	2011-01-01	2011-01-01
3	CUST_4	John	123	2012-02-02	2012-01-01
4	CUST_4	Jon	123	2013-01-01	2013-01-01
5	CUST_4	Jon	999	2014-01-01	2014-01-01
6	CUST_4	Jonathan	345	2014-02-01	2011-12-01
6	CUST_4	John	345	2014-02-01	2012-01-01
6	CUST_4	Jon	345	2014-02-01	2013-01-01
7	CUST_4	Jon	111	2020-01-01	2020-01-01



Due to the backdated adjustment, at 'arrival point' 3 and 4 there are multiple values 'active' at the same CDTS point in time. For example, at CDTS 2012-01-01 the value for 'Number' is *both* '123' and '345'.

Both values are correct. Which one is should you use?

The bi-temporal time machine

The provided example shows what happens to the data when backdated adjustments appear, and subsequently why values in the past are updated – or at least appear to be – looking at it from the present.

The example uses the attributes 'Name' and 'Number' to explain the concept, but the mechanism applies equally to any other attribute including metrics.

A mechanism that can query data in a way that represents this example provides a way to explain how data behaves to consumers – why the June sales figures are different today compared to last month for example.

Having this pattern implemented in a deterministic way makes it possible to travel back in time along the LDTS axis to 'rewind' the records that were created by back-dated CDTS information. This is also where the parameters of the load window come into play, as an implementation of this time travel concept. Time travel (filtering) is done along the LDTS axis, the order of arrival.

If, for example, at present (now) we *show* the data as reported at 2013-06-06 (LDTS timeline!) we will not yet have received the backdated adjustment and will (deterministically) display the data as it was present in the Data Warehouse at that point in time. This way we can (re)produce the June figures, and prove that the information represented was accurate.

We can then fast-forward (again along the LDTS axis) to the point in time when adjustments were made – to 2014-02-01 – and show the effect this has on the figures by virtue of the additionally created rows.

Which records to use?

The answer on which records to use lies in this bi-temporal context. Do you want to display information as per the latest date of arrival? In that case you would use the 'Number' value '345' and not '123' at CDTS 2012-01-01 and 2013-01-01. This is what happens in a bi-temporal result.

The provided example essentially created a bi-temporal Dimension, which can be complex to understand and is known to create confusion for consumers.

If you are looking to deliver a bi-temporal dimension as the final result, consider using better understandable terms such as:

- Known_from (LDTS axis)
- Known_to (LDTS axis)
- Valid_from (CDTS axis)
- Valid_to (CDTS axis)

True to the Virtual Data Warehouse way of thinking, we have created a single Dimension view that incorporates deterministic mechanisms for displaying the full correct history, presenting the data the same if it was to be incrementally loaded over time. The load window mechanism allows for the time travel part.

Try it out! The sample code is available on the website (requires SQL Server).

This result, the bi-temporal Dimension represented either as a view or as physical object loaded by the view or similar ETL process, should be seen as a helper / interim construct to facilitate further delivery.

It provides an accurate representation of what has happened and allows for a next layer of interpretation to make it suitable for further consumption – following the agreed business rule for its interpretation.

Based on this capability, next steps for information delivery may include traditional Slowly Changing Dimension approaches (e.g. Type 1-3 or even up to 7).

How does the view work?

As you can imagine, the logic to represent this mechanism in a deterministic way is suitably complicated. However, it is also a pattern with a clear structure and as such can be generated from metadata.

Because the logic largely follows the same approach as described in the 'A pattern for Data Mart delivery' paper I will only cover the differences that concern the handling of back-dated adjustment.

The full examples can be downloaded from the weblog. This includes the sample code and functions to run these yourselves, assuming you have a SQL Server instance handy.

The difference compared to the logic as outlined in 'A pattern for Data Mart delivery' lies in the sections where the context is joined back to the date range containing all change points-in-time.

Comments are added in the query on the next page to clarify the function of the added sections. But essentially, there is a union added in the join section (Outer Apply) which will artificially add the additionally created rows caused by the backdated adjustment – within the range that they are required for.

```
OUTER APPLY
(
  SELECT sq.* -- Regular row creation using All Changes date range
  FROM
  (
    SELECT
      src.*,
      'false' as is_generated_row,
      LEAD(src.CHANGE_KEY, 1, 99999999999999999999999999999999999999)
      OVER (PARTITION BY src.CUSTOMER_SK ORDER BY src.CHANGE_KEY ASC) AS CHANGE_KEY_EXPIRY
    FROM dbo.SAT_CUSTOMER src
    WHERE src.CUSTOMER_SK = All_Changes.CUSTOMER_SK
      AND src.LOAD_DATETIME <= All_Changes.LOAD_DATETIME
  ) sq
  WHERE sq.CHANGE_KEY <= All_Changes.CHANGE_KEY
    AND sq.CHANGE_KEY_EXPIRY > All_Changes.CHANGE_KEY

  UNION ALL
  -- Late arriving rows which are created additionally.
  -- This is to cover the changes going forward from a back-dated adjustment
  SELECT sq.*
  FROM
  (
    SELECT
      src.*,
      'true' AS is_generated_row,
      NULL AS CHANGE_KEY_EXPIRY
    FROM dbo.SAT_CUSTOMER src
    WHERE src.CUSTOMER_SK = All_Changes.CUSTOMER_SK
      AND All_Changes.row_src != 'SAT_CUSTOMER' /* Should not be a change from this table */
      AND src.CHANGE_DATETIME < All_Changes.CHANGE_DATETIME_PREV /* Is late arriving */
      /* and within affected range */
      AND src.CHANGE_DATETIME > All_Changes.CHANGE_DATETIME
      AND src.CHANGE_DATETIME < All_Changes.CHANGE_DATETIME_EXPIRY
  ) sq
) SAT_CUSTOMER
```

Final thoughts

One of the first questions that comes up is 'does it perform'?

The answer (beyond 'it depends') is usually 'not really. Load windows alleviate the performance to a certain extent, but an Outer Apply will always be a resource-expensive operation. Also, Outer Apply is not supported by all databases although standard SQL provides a similar feature called a Lateral Derived Table (e.g. Cross Join Lateral).

As with many things, the implementation can be done in many ways – including breaking up the steps into more procedural code.

For now, I hope that this example and corresponding code explains the nature of late-arriving data and demonstrates that it is possible to represent this correctly in a view.

Next, I will investigate how I can incorporate this into the 'twine' concept as published by Lars Rönnbäck. This is a very interesting approach that is likely to fit well in what I am trying to achieve and hopefully achieves better performance.

Comments and feedback are always welcome!