

## ***A Sense of Perspective:***

### ***How to use a Star Schema Data Warehouse to see any historical view you want***

Steve Morton  
Applied System Knowledge Ltd

#### Introduction

Star Schema data structures have been with us for nearly a decade, since Ralph Kimball's pioneering work in the 1990's. But although they are widely implemented, few really use the true power of "slowly changing dimensions" (SCDs) to reflect any viewpoint in time. And did you know there are three different ways to structure SCDs, each better for a different type of history view?

First there are the obvious queries - describing fact values using attributes of the date they were transacted, or describing them using today's attributes.

But how would you query for lifetime value of all customers matched by some other dimension query? Some stars can support this in a single query - others would take two or even three stages of processing. Yet each is using SCDs! And what about elapsed-time queries? Such as 'business for all customers who spent less than three months on a price plan'? Which is the best form of SCD to support that?

This paper describes how to design, maintain and query historical Star Schema, including what to consider when choosing the best type of SCD for your most frequent queries.

#### Time in the Real World

In the real world, business events and transactions happen through interactions involving people (customers, suppliers, staff and so on) - but in computer systems we can only consider what is recorded in the system, and when. So Time in the computer is only an approximation to Time in the real world. In the Data Warehouse, we can only reflect what has been recorded - but this in itself is a challenge to get right.

Real life is always more "messy" than we would like. It would be marvellous if every customer told us immediately they did something, like move house, get married, change bank account, change employer... Often we learn about events some time after they have happened or occasionally *before* they happen (such as change of address from next week). When this is important to the business, data will be recorded with an 'effective date' - and we need to use the same techniques in the Data Warehouse.

#### Time Among the Stars

One of the most effective ways of recording Time is in the context of a Star Schema. Techniques can be applied that allow any variation of Time perspective to be supported.

When designing and implementing Star Schema data stores, we must always consider how we need to deal with Time - both in how we intend to maintain Time-based data, and how we intend to query it.

Apply the KISS principle (Keep It Simple, Stupid...) - the more sophisticated the treatment of history, the more difficult is the maintenance and the more complex the star query becomes.

Decide:

- What history views will need to be queried?
- How rapidly do attributes (fields) change?
- How will you identify changes in the incoming data?

## What Version of History Do You Want?

This is a seemingly simple question - yet sometimes complex to answer. The basic decisions have to do with Dimensions and how they will be used.

One perspective that is always required in my experience is that of *information as at the time it happened* - reporting Facts using all the Dimension attributes that were current at the time each transaction took place. This can be complicated by the impact of *late-arriving data* - a topic that I will consider later in the paper.

Another common perspective is *using today's (current) attributes for one or more Dimensions* - such as, sales performance grouped on current team membership after a reorganisation; or product-line grouping. A refinement to this can be *using a proposed (future) organisation* - taking a version of the Dimension and re-organising it, then using this to analyse and report, often expressed as a type of "what if?" analysis by business users of the data.

A variation on that perspective is *as at some past date* - usually corresponding to a key point in the business calendar, such as start of financial year, or start of calendar year.

When one considers Fact data, some other choices emerge.

Some Fact values are easy to consider, since they happen at a moment in time. For example *value of a sold item* in retail is simple - a single date and time for the event records it exactly. This can be complicated by how to count *product returns and refunds*, but these also happen at a moment in time, so are relatively simple. In each case, it is easy to identify the Dimension attributes to apply to the Fact.

Other type of Facts can be more difficult. Consider *payments that cover a time-span* - for example, annual subscriptions, or an insurance premium. These are effective over a period of time, during which other information may change; a subscriber might move house, change bank account details etc. - and yet the annual payment took place at a single moment.

Sometimes we have to report these spanned Facts *as at the time of payment*; this is relatively simple.

In other cases, we may need to consider the payment to be spread over the whole interval. This is common with insurance premium, where the business considers *earned premium*, often on a daily basis over the term of the contract. With this perspective, the amount earned each week could be reported with Dimension attributes *as at the time of payment* or attributes *as at the time of counting earned value*. This might even need to consider the daily changes - for example, with a mid-week change, do we need to use each day's attributes for summarising, or just those at the start (or end) of the week?

These Fact perspectives can become even more complex when we have to consider *refunds and additional payments mid-term*. For example, if a customer cancels insurance after a few months and is given a credit or repayment for the "unused" part of the year, how will this be counted?

Be assured - the more variations the business needs to see, the more complex the data management and query will become! Reality check: try to identify the cost effects of supporting a wide range of history needs, so these can be balanced against the benefits; sometimes the business asks for the moon, but only has the budget for a balloon!

## Simple Basics - Facts and Dimensions, including Time

The basic principle of the Star Schema is that of recording additive, numerical information (Facts - such as transaction value) in one table, identifying the rows by a combination of keys that relate to Dimension tables that will contain all descriptive information corresponding to the Dimension key value. (For a full discussion of Star Schema principles, see references at the end of this paper).

Almost always, one of those Dimensions is called 'Time' (or something similar). This allows identification of when the Fact event occurred - for example, the date of a sale.

Star Schema's actually vary in type between simple aggregate summaries, that use Time only as a reporting Dimension, to highly sophisticated history of 'atomic' detail Facts with every Dimension change being recorded.

The type you choose for your Star will depend upon what kind of history you need to keep and query.

### Slow Change - Kimball Type 2 and 3

Kimball originally identified three types of change recording in Dimensions:

- Type 1 - replace the attributes with the latest value (i.e. no history)
- Type 2 - full tracking of all changes, a new "edition" of the Dimension row for each change
- Type 3 - current and previous value for specific field(s) only (limited history)

Types 1 and 3 are "cheap" in terms of disk-space, but only type 2 gives full history recording.

However, there is not one single way of defining Type 2 management. The essential factor is writing an updated version of each Dimension row as it changes, and identifying when that 'version of the truth' applies. But the detail of how this is recorded can vary somewhat, leading to particular performance characteristics.

### How many Dates do you want? - Business Dates versus Warehouse Dates

It is important to distinguish between *when an event happens* and *when the data warehouse records the event*. The latter is, largely, a technical date - often calculated as the load-date or extraction-date of new records being stored. We typically time-stamp our incoming data automatically for two main reasons:

- So that we can associate Fact data with the Dimension values that are true at the time of loading
- To support removal of unwanted data - either retiring data 'aged' beyond its retention period, or (sometimes) deleting data loaded in error

These technical dates give us the basic integrity of our Star Schema. We can refer to them as *warehouse effective dates*.

However, sometime the business has recorded dates that will reflect when the event happens - or even a time-period over which it applies. This can lead to multiple business dates being needed. For example, a customer may pay an insurance premium today (*payment date*) for a policy that begins in a week from now (*effective begin date*) and runs for three months (until an *effective end date*). If the data warehouse is being loaded weekly, it is likely that the *warehouse effective date* does not match any of these!

So be prepared to have several dates to record (especially for Facts - see below). It is therefore crucial that you choose meaningful names and labels for these (just calling a column 'date' is clearly not enough!)

Also consider whether you will use Date values alone, Date-time values or separate Date and Time values. The decision will often depend on your business data, but again it is important to be clear what kind of values are being used. See also *The Trouble with Time Dimensions* below.

### Surrogate Keys - to keep or assign new values?

Related to the history/dates question is the assignment of surrogate key values in Type 2 SCD's. Whenever a new row is added to a Dimension table a surrogate key value will be assigned; but when that row is later changed, a new "edition" of the row is written to supersede the earlier one. The two alternatives here are to either allocate a new surrogate key every time a Dimension record is changed, or to keep the same surrogate and qualify the row with 'effective-from/effective-to' date pairs.

Those who advocate assigning a new key value to each "edition" of a Dimension record generally cite two reasons for this - the Dimension surrogate key can be subject to 'unique index' constraints, and a single date-stamp avoids the need for 'between' comparisons. This latter is largely because some relational DBMS's did not in the past support *value between column1 and column2* constructs, but only *column between value1 and value2*. This is not an issue with SAS<sup>®</sup> PROC SQL (or indeed with many current RDBMS versions).

I tend to side with those who advocate retaining the same surrogate key for a business key value. A pair of dates allows the values within a single row to directly indicate for what time period it was true, without needing to refer to other rows - especially important in SQL, with its inability to perform inter-row calculations (cf. data step 'lag' functions). The 'unique index' constraint is of little real value - processing logic should ensure that a surrogate key is never duplicated in the Dimension. The 'between' problem can be avoided, even in a database that doesn't support the required syntax. Most importantly, by retaining the same surrogate key, all Fact rows for that business key are directly available in a single query.

Thus a Dimension with preserved surrogate keys more easily supports the running of queries that select on current attributes of the Dimension, but aim to retrieve all Fact rows for selected Dimension rows. An example of this would be *value of all transactions this year by customers in profile-group X* - easy with a preserved surrogate key, but requiring a two-stage query when new keys are allocated.

Preserved surrogate keys also help with late arriving Dimension changes - rendering it unnecessary to find and re-write already loaded Fact records (see *Changing the Past* below).

#### Just the Facts... Time in relation to Fact records

I mentioned above the variety of dates (or date-time combinations) that may apply to a Fact; specifically:-

- Date the Fact transaction is originally recorded - *business\_date*
- Date the Fact row is loaded - *warehouse\_date*
- Time-period for the business to count the value - *business\_from\_date* and *business\_to\_date*

Late-arriving data (or retrospective changes) can add extra dates to this (see *Changing the Past* below). These include:-

- Booking date for late transaction - *booked\_date*
- Reversal effective date - *reversal\_date* (or it may be possible to use *business\_date*)

All these dates help fully identify the Fact row and allow appropriate reporting and calculations to be done.

#### The Trouble with Time Dimensions - 'Date-time', the Monster Dimension

Operational OLTP systems often use a combined time stamp (date-time as a single value) for each transaction or change. So it would seem the obvious way to handle this would be a Date-Time Dimension to key these Facts.

On a small scale, this can work. But consider the number of unique values that will be stored. OLTP database time-stamps are often down to the micro-second or less. So a million transactions recorded with such a time-stamp will produce a million-row Dimension table, since the chances of an identical time-stamp value appearing are vanishingly small. Even rounding the values to whole seconds will produce a very large Dimension table to cover, say, 5 years history (there are 157.7 million seconds in 5 years! - even if only 10% of the possible values actually appear, that is still over 15 million rows).

Avoid this issue by creating separate Calendar Date and Time of Day Dimensions, as well as truncating Time of Day values (it is very rare even that to-the-second time values are needed, often to-the-minute

is good enough). There are only 86,400 seconds in each day - so a Time of Day to the Second Dimension is small; likewise 5 years equals a mere 1,827 days - so a Calendar Date Dimension for this is trivially small.

The overhead of using two keys in the Fact table is more than offset by smaller indexes and faster queries. Hour-of-the-day or day-of-the-week will often be used to group this kind of data, so the full time-stamp is rarely used as it stands.

Note, you can still safely use Date-time values to 'time-stamp' records in the data warehouse - it is as Dimension keys that they will impact performance dramatically.

### Late Arriving Data - Changing the Past

One of the more contentious questions can be *how to deal with late arriving data?*

Examples of late arrival are:-

- Sales transactions from last week arriving with this week's data (can be caused by data transmission delays, or paper-based systems with subsequent data entry)
- Correction of data entry errors (such as missing decimal point leading to value\*100 being entered, or the wrong product being entered for an order)
- Back-dated real world events (such as customer moved house three months ago and just informed us about it, or specification of a product changed some time ago)

This may be contentious because there is often a lack of real agreement on how this change should appear in the data warehouse. Some departments will want to report *as if the change had been recorded when it happened*; others may need *as at the date of loading into data warehouse* or *as at business record date*. There will always be the question of whether (and when) to re-create any summaries of the affected data, and whether a change against figures that have been reported in the past will be acceptable.

If you cannot get a single, enterprise-wide rule that everyone agrees, be prepared to record all the relevant dates and support any perspective. But, as with naming your dates, be very clear to business users what view they will be using!

Correction of errors may need some careful planning. Consider first how you will receive the correction in your incoming data. Will you receive a "reversal" transaction for the original (wrong) amount? Or will you need some other way of identifying the original - now invalid - transaction? Thankfully most modern OLTP systems require a reversal so that changes can be audited, but this may be an issue with older source systems. If no other means exist you may be forced to compare with data already loaded - this should be the last resort, as it is very expensive to process!

Then consider how you will record the update. Will you need to eliminate the original record altogether, as if it never happened? Or is it enough to store the reversal so that it cancels out the value of the original? The latter can be achieved simply by ensuring the correct dates are used for the Facts; but deleting a matching old record may be more difficult. Try to ensure that simple keys (like a posting-number) can be used to match records if this is a frequent occurrence in your data.

A particular issue can be *Facts already stored contemporary with the old Dimension*. Again, multiple dates can help here - especially if you decided to preserve surrogate keys through Type 2 changes (see above). If you are using new surrogates every time and you want to associate the old Fact with the corrected Dimension values you will have to locate the old and new Dimension rows, locate all Facts with the old surrogate key, and rewrite them with the new surrogate key. (This is another reason I prefer to retain keys and use effective-date pairs!)

## Can you ever have too much Time?

Now that you know such a wide range of options for storing and querying history, it is tempting to record absolutely everything. This could be an expensive mistake!

Don't keep history just for its' own sake - you do need solid business reasons, since there will be some costs, both in storage and in querying the schema. Estimate the costs, and then consider what the value is to the business, before making a final decision.

Be on the lookout especially for 'slowly changing' attributes that actually change frequently - perhaps every month; these will be very expensive for larger Dimensions. Consider removing these from Dimension tables and treating them like Facts, or using Type 1 or Type 3 change recording for these. You can mix these types together with a Type 2 Dimension, but it does complicate the update logic, and you will need an extra date for the Type 1 or Type 3 change, separate from the Type 2 effective dates.

An example of this is a *last 36 months payment history* attribute (common with credit card or loan data, where there is an operational requirement to see this history). Even a single change in payment habits once by a customer will lead to 36 separate changes in that field as it ripples along. If it is recorded as a Type 2 Dimension change, the customer's record would be re-written 36 times for that change alone over a 3 year period!

## The Ultimate Star Schema

If you were to decide that all possible perspectives are needed, it would be possible to define this, using all the techniques together. It should be noted that I am **not** recommending you build a Star Schema like this - this is just to illustrate how far you have to go to support everything in a single structure!

The ultimate Dimension would have every possible control date, and could be updated using Type 2 techniques for the Type 2 attribute changes as well as using Type 1 or 3 for those changes (a Type 1 or 3 change would only alter the 'current' row in each case). See the table below for a full list.

In practise, you would try to segregate attributes with different type and rates of change - but those optimisations are beyond the scope of this paper.

<b>The Ultimate Customer Dimension</b>			
<i>Columns</i>	<i>Purpose</i>		<i>Changes?</i>
SK_Customer	Surrogate key for Customer Dimension		Never
Customer_Reference	Customer reference number	Business key - in this case simple, but could be composite	Never
Warehouse_date_stamp	Date of update for Type 1 or Type 3	Generated during Load/Update	No
Business_date_stamp	Date of extracted data for Type 1 or 3	May be an OLTP date-stamp or generated during Extract	No
Warehouse_effective_from	Date when this row became current in the data warehouse	Generated during Load/Update	No
Warehouse_effective_to	Last date on which this row was current in the data warehouse	Generated during Load/Update	When a row is 'expired'
Business_effective_from	Business date when this row became true for the data warehouse	May be an OLTP date-stamp or generated during Extract	No
Business_effective_to	Last business date when this row was true for the data warehouse	May be an OLTP date-stamp or generated during Extract	When a row is 'expired'
<< table continued ... >>			

Current		Has a 'true' value for the unexpired (i.e. currently effective) rows. Usually Current='Y' is used.	When a row is 'expired'
First_Name, Last_Name,..	Descriptive attributes	Type 2 attributes	Very slow change
Social_Security_Number	Descriptive attributes	Type 1 attribute	Never changes, except with error correction
Address, email, phone, occupation/job_code	Descriptive attributes	Type 2	Slow change
Current_credit_rating_score, Credit_rating_when_accepted, Current_market_segment	Descriptive attributes	Choice of type 2 or 3 - type 3 uses the 'original' (when accepted) value and 'current' as a pair, where Type 1 simply replaces 'current' value each time (no history)	Could change often, depending how maintained
Payment_history_36_months	Rolling field of payment habits - most recent month at the left	Type 1	Every month

Then the Fact table could support all Fact time perspectives, using some more dates. Note, we should not need a Warehouse\_fact\_effective\_from/effective\_to pair, since the "counting period" dates are business dates only:

<b>The Ultimate Fact Table</b>		
<i>Columns</i>	<i>Purpose</i>	
SK_Customer, SK_....	Surrogate keys for all Dimensions	
Warehouse_fact_date	Date the Fact record was loaded	Generated during Load/Update
Business_fact_date	Business date stamp for the Fact transaction	Must be an OLTP date-stamp.
Original_business_fact_date	Business date stamp for the Fact transaction which this one reverses.	Must be an OLTP date-stamp. For a non-reversal this will have the same Date as Business_fact_date to ensure query consistency.
Business_fact_effective_from	Business date from which to count the Fact transaction 'earning units'	Must be an OLTP date-stamp
Business_fact_effective_to	Business date from which to count the Fact transaction 'earning units'	Must be an OLTP date-stamp
Fact_Amount	Fact value for the transaction recorded	Original Fact amount from OLTP
Earning_unit_amount	Amount per day of the period between 'effective from' and 'effective to'	Calculated during ETL for easy 'earning per time-period' calculation in queries

This type of Fact table must be used with great care, especially if it has to deal with 'refunds' (reversals). These would work best with a separate reversal row being added with negative Total and Earning amounts, which runs from the refund date to the end of the original period. If this were not acceptable, you would have to update (change) the existing Fact row - very messy indeed!

## Querying the Ultimate Star Schema

Here are a just a few examples of querying different time perspectives; just by considering which dates you need to match/compare you should be able to create any variation you wish from the 'Ultimate Star Schema'. Note, limited presentation time means that I cannot show these in full at SeUGi - they are only here so that they can be in the Proceedings!

Note, in Version 9 SAS PROC SQL it will be possible to influence performance by explicitly declaring which table is 'fact' and which are 'dimensions' - I look forward to trying this!

All known transactions for a Market Segment using as-at-time-of-business attributes:

```
proc sql ;
  create table result_sql as
  select  Fact_amount,
         Calendar_month, Calendar_year,
         Customer_reference, Name, <<other attributes>>
  from    Ultimate_facts facts,
         Ultimate_customer customer,
         Calendar_date date,
         <<other dimensions>>
  where
  /* query for a single market segment*/
  customer.Current_market_segment = 'AB45'
  and
  (Business_fact_date between
   Customer.Business_effective_from and Customer.Business_effective_to)
  and
  (facts.SK_calendar_date = date.SK_calendar_date)
  and
  << other Dimension key-and-date match conditions >>
  ;
quit;
```

The same query, but using 'current' attribute references only:

```
proc sql ;
  create table result_sql as
  select  Fact_amount,
         Calendar_month, Calendar_year,
         Customer_reference, Name, <<other attributes>>
  from    Ultimate_facts facts,
         Ultimate_customer customer,
         Calendar_date date,
         <<other dimensions>>
  where
  /* query for a single market segment*/
  (customer.Current_market_segment = 'AB45'
   and Customer.current = 'Y')
  and
  (facts.SK_calendar_date = date.SK_calendar_date)
  and
  << other Dimension key-and-date match conditions >>
  ;
quit;
```

This time, only for those customers that were in that segment on 1st January:

```
proc sql ;
  create table result_sql as
  select  Fact_amount,
```

```

        Calendar_month, Calendar_year,
        Customer_reference, Name, <<other attributes>>
from    Ultimate_facts facts,
        Ultimate_customer customer,
        Calendar_date date,
        <<other dimensions>>
    where
/*    query for a single market segment*/
    (customer.Current_market_segment = 'AB45'
     and '1Jan2002'd between
     Customer.Business_effective_from and Customer.Business_effective_to)
     and
    (facts.SK_calendar_date = date.SK_calendar_date)
     and
     << other Dimension key-and-date match conditions >>
;
quit;

```

The original query could reproduce the previous results after 'late arriving' data, by using warehouse dates, similar to the following:-

```

proc sql;
    create table result_sql as
    select    Fact_amount,
            Calendar_month, Calendar_year,
            Customer_reference, Name, <<other attributes>>
    from    Ultimate_facts facts,
            Ultimate_customer customer,
            Calendar_date date,
            <<other dimensions>>
    where
/*    query for a single market segment*/
    customer.Current_market_segment = 'AB45'
     and
    (Business_fact_date between
     Customer.Warehouse_effective_from and Customer.Warehouse_effective_to)
     and
    (facts.SK_calendar_date = date.SK_calendar_date)
     and
     << other Dimension key-and-date match conditions >>
;
quit;

```

## Conclusion

There is no single "correct" way to handle Time in data warehousing, but several fundamental techniques applied in combination allow you to implement the correct way for your business.

Do be clear about what you are trying to achieve with History and how your business will utilise what you store. Consider how the business users will work with the data warehouse - access to such a rich source of history is usually a specialist requirement, but make sure your users do understand the different perspectives available.

By all means make a sophisticated Star Schema to record everything - but don't make it more complex than the business can understand, or the history will never be used. And remember to estimate the costs - the more you maintain, the more processing and storage you will consume.

## References

For further reading on Star Schema - from basics to advanced topics - consider the following:

*The Data Warehouse Toolkit*, Ralph Kimball.  
1996, John Wiley & Sons, Inc.

This is the original book by Kimball, now a little dated and much extended by later articles (see website reference below).

*The Data Webhouse Toolkit*, Ralph Kimball and Richard Merz.  
2000, John Wiley & Sons, Inc.

Updates the earlier work, and specifically deals with Internet-related dimensional warehouses - both through the web and using web data.

Also see the Articles section on Kimball's website at <http://www.rkimball.com/html/articles.html>

*Data Warehouse Design Solutions*, Christopher Adamson and Michael Venerable.  
1998, John Wiley & Sons, Inc.

Explains common data models for Star Schema; an excellent practical guide to designing Stars.

SAS<sup>®</sup> is a Registered Trademark of SAS Institute Inc., Cary, NC, USA.

## Contact Details

The author would welcome comments on this paper, and can be contacted as follows:

Steve Morton  
Applied System Knowledge Ltd  
51 Blandy Road  
Henley-on-Thames  
Oxon.  
RG9 1QB  
United Kingdom

Tel: +44 1491 411977

Email: [steve.morton@appliedsystem.co.uk](mailto:steve.morton@appliedsystem.co.uk)

Web: [www.appliedsystem.co.uk](http://www.appliedsystem.co.uk)