

How Many Dates Do You Want?

The Multiple Personality of Time in Your Data warehouse

Steve Morton, Applied System Knowledge Ltd., Henley-on-Thames, U.K.

ABSTRACT

Data warehouses maintain History – we all learned that in “data warehousing 101” – and History means Dates! But how many dates do we need to support an organization’s need for history? Do we need to reconcile the data warehouse with current (and maybe past) operational reports? Must we reconcile with the General Ledger (sometimes a key Data Quality metric to prove the data warehouse is “correct”)?

Using appropriate dates can be the key to answering all your history questions. You could just record every imaginable date, with all the potential complexity that involves, but that can create as many problems as it solves.

This paper draws on the answers the author has discovered over the past several years to these date-related questions, and seeks to explain why you might want to use – or to avoid – certain date constructs in your data warehouse.

INTRODUCTION

A data warehouse is full of dates – arising from the business systems data, from the data warehouse operations, from the need to keep history.

So it’s obvious – we keep all the dates, right? Yes, but what do we mean by “all the dates”?

The closer you look at data, and particularly data with history, the more dates you seem to find!

Do we need *everything*? Over the course of this paper I hope to give you some clues to finding those dates and deciding which ones you need to keep.

THE BASICS - DATE OR DATE-TIME?

A basic question: should we use date values, or date-time values? Sometimes this is dictated by technology – for example, many relational DBMS’s only have one data-type for date and time, whereas SAS[®] has date, time and date-time as distinct types.

The discussion here will only differentiate when it matters – almost all examples are equally true, whether date or date-time is used in each case. I will simply use the word ‘date’ to keep the text simple.

However, I would recommend that you standardize within your data warehouse, especially for those columns used in history management, and if possible for all date columns. This will be most important when querying and joining data – a data warehouse which mixes the two types will often need more complex query code than one which is consistent throughout!

UNDERSTANDING THE TIMELINE

Understanding how and when dates will occur in your data is fundamental to any decision about the use of dates in the data warehouse. This requires understanding both *what each date represents* and *how and when data can change* (both date information itself and other columns). These will help us with decisions about history management, and how we will organize tables for efficient storage of that history.

To gain a better understanding of these changes, I often draw a timeline diagram (example below) to indicate all real-world events and the computer records of those events as they relate to an entity in the business – for simple cases you might be able to imagine all this without needing a picture, but as the cycle becomes more complex a diagram really does help a lot.

Try to identify all real-world and business systems events that lead to data being recorded. For a complete understanding identify which system will record each event and how – this will support your data acquisition from those systems.

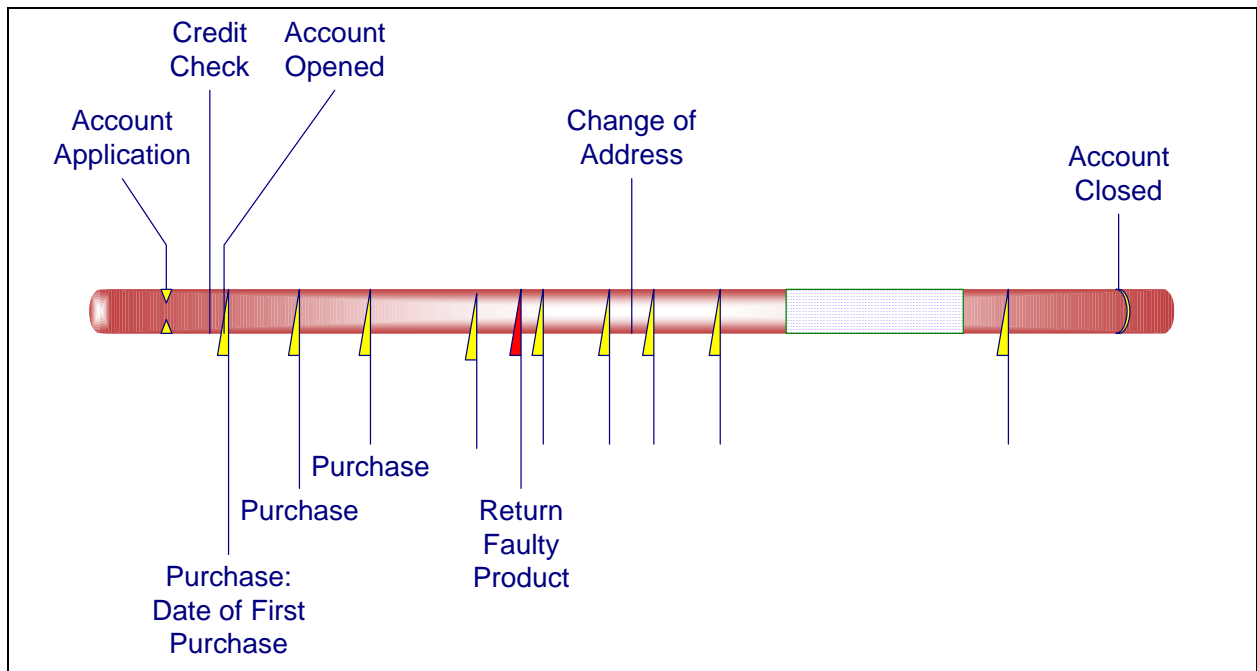


Figure 1. Simple Customer Account timeline.

Below the line are ‘retail transactions’, Above the line are ‘account events’.

The simplest dates we find are those which are purely **descriptive** in nature - the most obvious example being **date of birth**. This kind of date is often a fixed value (once recorded it never changes) and a single data record can contain several of these.

For example, a personal customer record could have the following **fixed descriptive dates**:

- date of birth
- date the person first became a customer
- date the person was first granted credit

It could also have some **changing descriptive dates**:

- date of last change-of-address
- date of last purchase
- date of last credit check

Clearly these changing dates could be updated frequently – especially ‘date of last purchase’ (we hope!). We might even choose not to store such a volatile date in the data warehouse, but to derive it dynamically from purchase data. If we do choose to store it we would have to consider the effect on other historical data storage. More of this later.

Most of these changing dates are actually a specialized form of **business activity** dates. Business activity dates are often the most interesting for users of the data, but can be the most troublesome to handle. They are also our main providers of history within a data warehouse. Common examples are:

- Order and delivery dates
- Contract begin and end dates
- Retail purchase dates (for goods or services)
- Invoice and payment dates

So, we have a large number of dates that may be interesting – some of which come from the customer as personal information, but most come from the business’s interaction with the customer. These are the dates we have most confidence in, and are often the most interesting to analyze or report.

WHAT IS BEING REPRESENTED?

For any date we use we should always understand what it represents. This might seem obvious – but isn't always as easy as it appears!

There are two common types of date representation – *point-in-time* (otherwise known as *event*) and *time period*. The combination of these occurs in all systems, including our data warehouse. How much you will encounter each type is very dependent on what type of business you work in and the operational systems that support the business.

Many operational software packages will implement several business and system dates for their own purposes – providing representation of real-world enactment of business, or as an audit-trail of data entry and change. These may be “purely operational” in nature – and thus not passed to the data warehouse at all – or they may provide essential business information, when they must be faithfully recorded in the data warehouse for later use.

OCCURRENCE OF DATES

Dates can be used to record many things, among them:

- A real-world business date when some information was recorded (event or time-period)
- When an operational system processed the data (event)
- When a record was true in the operational system data (time-period)
- When a record was considered effective for the business (time-period)
- DW data acquisition, when the item was extracted from the operational system (event)
- DW load date, when it was loaded into the DW; possibly also another date when it was changed, if error correction is needed (one or more events)
- When a record was true in the DW (event or time-period)

We can choose to record all possible dates in our DW – but it is then easy to find we have more dates than actual data! This may take some explaining, so it is important to make an informed decision.

SINGLE EVENT, POINT-IN-TIME

Most obvious examples of these are ‘retail events’ – sales or payments. Other common events are account opening or moving address. These happen at a single time in the real world, and are recorded at a single time in computer systems.

Usually only a single date-stamp is needed to document the event, although the real-world event may happen earlier than the computer system record when some manual processes are involved. In that case the two event dates may be needed – the real-world date and the computer record date.

For example, if a payment cheque is received by a sales representative it may be physically dated when the rep accepts it. However, it may be a day or two later when the transaction is entered to the head office system – so now two event dates may be identified: *payment received* and *payment entered*.

Whether both event dates are stored in the data warehouse is a business decision, based on how the data will be used. If there is a need to exactly reconcile DW with operational data, both dates will probably be needed. Otherwise it may be enough to record just the one event date, probably *payment received* if a customer-oriented perspective is required. However, if an accounting perspective is required then the *payment entered* will be essential, since this corresponds to the ledger date.

Simple retail events, such as a supermarket sale recorded by EPOS in the store, might be expected to only have one date-stamp – since the event and the recording of it are simultaneous. However, we might also have to consider some asynchronous transfer from in-store systems to central systems. If the transactions are collected by the store and then forwarded as a complete set – say, overnight for a whole day's business – then there will be a second date recorded when the data arrives. This becomes very significant if there is any possibility of late arriving data, for example due to a communications link failure or other operational problem preventing scheduled transfer at the planned time.

MULTIPLE EVENTS?

It is important to note that some data have more than one event date associated to a single record. An example might be for an Account record at a supplier of goods or services. Relevant dates include:-

- Application date – when the account was requested
- Acceptance date – when the account was approved
- Opening date – when the account was opened (available to use)

- First Used date – when the account was first used for an order
- Credit Reference date – when the account holder's credit rating was checked
- Last Used date – when the account was most recently used for any transaction (e.g. order, payment of invoice, credit of a refund)
- Review date – when the account was reviewed (e.g. for increased credit)
- Closing date – when the account was closed (no longer available for new transactions)

Almost all of these do not change after they have been recorded – unless there was an error – although they will be recorded at different points in the time-line.

However, in this example the Last Used date will change every time a transaction is recorded and the Review could happen more than once, so this date might change several times over the life of an Account. The Credit Reference date might be a fixed value (when the credit rating was first checked) or it might change occasionally if it represents the *last time* the credit rating was checked – so it is important to know the exact definition for such a column!

We will need to consider whether any historical track needs to be maintained for these dates, and this will depend upon what use we expect to make of them in reporting or analysis (described below – see **What History Should Be Kept?**).

TIME PERIOD, FROM ONE DATE TO ANOTHER

Examples of these include effective dates for an insurance policy, duration of a promotional period, the time over which a list-price is used or a subscription period for a service. Time-period data always depends on recording both the beginning of the period and the end.

This type always requires two dates to fully document them; however we cannot always depend on the operational system having both dates. In some businesses it will be normal to have both – for example, insurance systems usually document the period of cover provided by a policy and these *effective from* and *effective to* dates are therefore used widely in those systems.

For this case we will always want to record both these business dates in the data warehouse. This is an example of a closed time period – it has a pre-determined end date.

In contrast, a retail system might only need to record a single date when a list price for an item becomes effective. The latest price has no end date, until it is superseded by a later price. This is an example of an open time period – it remains effective indefinitely, until replaced.

Now we will have a choice in the data warehouse – do we use only the single date on each price record? or do we create an 'effective to' date based on when an older record is superseded? For certain applications it will be easier to use the data if both from- and to-dates exist on each record. This is explained in the section on **Versioning of Data Rows** below.

MULTIPLE VERSIONS OF THE TRUTH?

A final complication for retail systems is potential co-existence of two – equally true – states for a record. The best example is a price-change for an item. Goods on the shelf may have already been price-marked, using the price in effect when the goods arrived in the store, but a new list price may come into effect before these items have all been sold. Now new items will be price-marked with the new price, but the store may continue to sell the older stock without re-pricing it – so the old price remains effective until all older stock has been eliminated. In this case you could not assume the price from the date of sale, but would need the actual price used for each transaction.

In practise a data warehouse would probably record the selling price for each transaction, as would the point-of-sale system. This will be more expensive in storage terms, but avoids the time related problem for this kind of data. It also copes with other issues, such as in-store price reductions or 'managers specials'.

HISTORY IN THE DATA WAREHOUSE

VERSIONING OF DATA ROWS

The most common use of dates in any data warehouse is to record history by creating a new version of a record when some value changes. This allows new information to be recorded without overwriting prior information, and is the basis of Type Two slowly changing dimensions as described by Dr Ralph Kimball in *The Data Warehouse Toolkit* and later writings.

Although Kimball does not describe any usage other than dimension data versioning, these techniques can also be used with normalized data models to provide similar history recording. [Note, Kimball uses the term 'partitioning' to

describe this; however, I find that 'versioning' is more readily understood, since that is what is actually done in maintaining the data rows]

Kimball's technique adds a single date-stamp to the record, but he combines this with a unique surrogate key for the row – effectively substituting this for a composite key of the business-key and date-stamp to act as a foreign key into the fact table. The date-stamp may use **Extracted date** or **data warehouse updated date** – depending on which is appropriate; if the two are sufficiently different you might choose to keep both, but only one is needed for the versioning date.

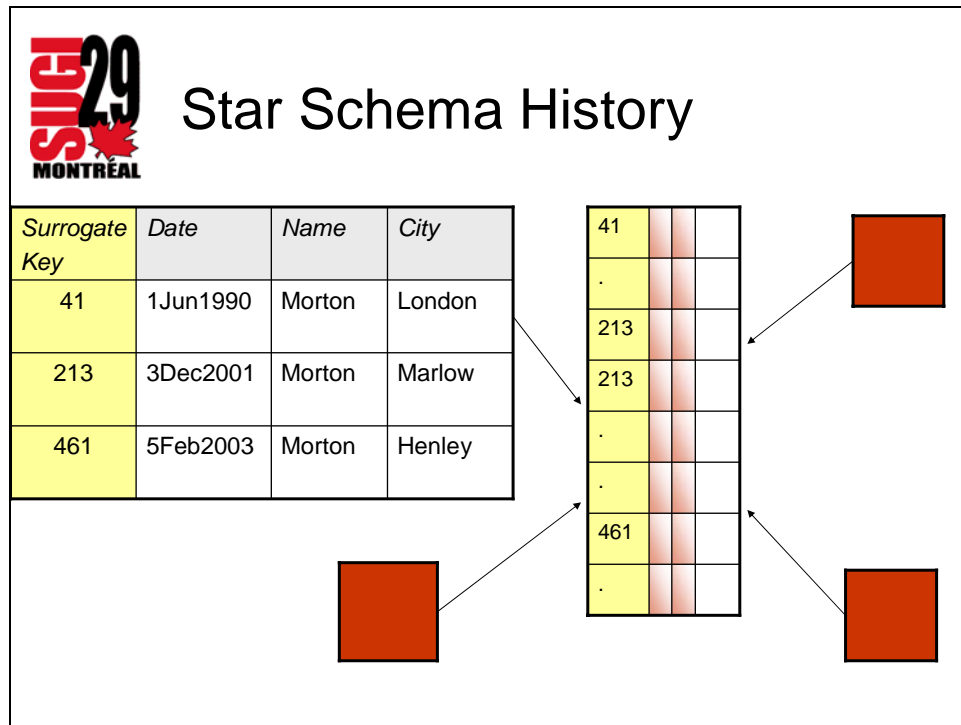


Figure 2. Versioning Dimension records - Kimball style

Here each new version of a single customer's dimension record has a unique surrogate key, so any row in the fact table links to only one version of a dimension row using the key to this dimension.

This technique can be extended by the use of **validity-date pairs** (*valid_from_date*, *valid_to_date*). Although this makes the data model a little more complex, it makes it much easier to query for data values at a given point in time. Paired dates are also especially important with normalized data warehouse data models, since these cannot use the unique surrogate keys that a star schema may implement.

Key	FromDate	ToDate	Name	City
24961	1Jun1990	2Dec2001	Morton	London
24961	3Dec2001	4Feb2003	Morton	Marlow
24961	5Feb2003	<high date>	Morton	Henley
.

Figure 3. Versioning with Two Dates

Now it is the date alone that differentiates the versions, and joining transactions with the correct version requires a condition that uses:

Transaction_key = *Key* and **Transaction_date** between *FromDate* and *ToDate*

We can also query this for a specific state of the table by a condition that uses the required date:

*Transaction_key = Key and **Required_date** between FormDate and ToDate*

This allows easy query of any point-in-time perspective. By choosing a past date here you can reproduce a past structure – for example, to report a whole year’s sales using the 1st January product group assignments, even when they changed later.

With the last version of a row there may be a need to close off the final version, as a form of “logical deletion” when there should be no new data arriving related to the row concerned. In this case the ETL should verify that no new data does arrive; if it did then that data might never join with a valid version of the closed sequence.

<i>Key</i>	<i>FromDate</i>	<i>ToDate</i>	<i>Name</i>	<i>City</i>	<i>Status</i>
24961	1Jun1990	2Dec2001	Morton	London	Open
24961	3Dec2001	4Feb2003	Morton	Marlow	Open
24961	5Feb2003	30Jan2004	Morton	Henley	Open
24961	31Jan2004	31Jan2004	Morton	Henley	Closed
.

Figure 4. Versioning with a closed sequence.

Note that, in contrast to the example given above (Multiple Versions of the Truth?) the versioning dates must **never overlap** as this would break a fundamental rule of version management. Usually the “old” version of a data row has *valid_to_date* set to a value 1 time-unit less than the *valid_to_date* of the next version (1 time-unit being relative to the data-type – 1 day if a date-only value is used, 1 second or even 1 microsecond if a date-time value is used).

It is sometimes possible that there might be a gap, with *valid_from_date* in the new record being some time after the *valid_to_date* of the previous version. However this is a special case – it would indicate a closed sequence being reactivated, such as a long-cancelled account being re-opened, and is associated with business systems that allow such situations to be recorded.

HISTORY WITHOUT VERSIONING

Do we always need versioning? Not unless we need to query using many past versions of information. If we only require a ‘current view’ we can simply overwrite each record when values change

Kimball described these as Type One – when an update simply replaces the record and no historical data is recorded – and Type Three.

With Type Three, a single column’s historical value can be tracked by having an additional column for each old version of the value. The basic description of Type Three just uses a single “old value” or “original value” column, but it is possible to have several old values. At any time there is only one version of a data row – it is simply updated in place when the values change.

For Type Three records we are often interested in knowing when the record was last changed, so a single **data warehouse updated date** is used. In some cases we might also need the date the previous value was recorded – so a *date of previous value* would be added to record this.

<i>Key</i>	<i>UpdatedDate</i>	<i>Name</i>	<i>City</i>	<i>Previous City</i>	<i>Status</i>
24961	5Feb2003	Morton	Henley	Marlow	Open
.

Figure 5. Type 3 example - single field history.

This limited style of history cannot tell us true history – just a past-and-present snapshot.

An extended example may appear when multiple previous states are required, such as several past months or quarters values. This is common in data for credit card payment history – indicating whether past months’ bills were paid in full or not. Each month has 0 if not paid in full, 1 if paid in full, and the values roll down the list as each new month is recorded. This technique is colloquially known as “duck eggs”, because of the appearance when printed

side-by-side (000010001000). Again you will need to know when this data was last changed, but this will usually be provided by the *dw_updated_date* for the whole record.

Key	UpdatedDate	Name	12 Months Payment History												
24961	1Feb2004	Ms. P. Always	1	1	1	1	1	1	1	1	1	1	1	1	1
24961	1Feb2004	Mr. P. Sometimes	0	0	0	0	0	1	0	0	0	1	0	0	

Figure 6. Type 3 'payment history'

Key	UpdatedDate	Name	12 Months Payment History											
24961	1Mar2004	Ms. P. Always	1	1	1	1	1	1	1	1	1	1	1	1
24961	1Mar2004	Mr. P. Sometimes	1	0	0	0	0	0	1	0	0	0	1	0

Figure 7. Type 3 'payment history' - updated for the next month

HISTORY AS A TRAIL OF TRANSACTIONS, EVENTS OR SNAPSHOT SUMMARIES

The final type of history is simply recording a series of transactions – either atomic (unit level) or summarized. In Dimensional data models these are characterized as Fact tables; in Normalized models they appear as transaction or summary snapshots.

These typically require only a single 'activity date' (often the **transaction date**) so that each record can be associated with other contemporary historical information for the customer, product or whatever the data relates to.

RECONCILIATION

We always want our data warehouse to be an accurate representation of data. This usually means it must agree with the operational source system. However, the operational system data continues to be updated after we have extracted from it. So the most obvious requirement is to know **when the data was extracted** which should allow us to reconcile with a point in time for that source data.

Sometimes we are asked to reconcile the data warehouse with the General Ledger (system of financial accounts). *This can only be assured if the data comes from the General Ledger!* That is because the **booking date** used by the ledger does not always correspond with the **customer transaction date**; in which case the other operational system will not agree with the GL – so what chance does the data warehouse have? We can't perform the impossible. So if this becomes a requirement for you, make sure the customer transactions and GL transactions can be matched and that you can extract data from both systems.

ERROR CORRECTION

In a perfect world there would never be errors, and we would never load bad data. However, even in the best run systems there is some risk, no matter how small. So the final contributor to dates in a history-managed environment is error correction.

You may choose to correct errors simply by providing correct data and running the normal process to apply this change. That would be easy for a Type One style table, less easy for a Type Three table (since you would probably want to use the previous good value in place of the bad value). However for Type Two the normal update adds a new version and the bad record will stay for ever unless action is taken.

An alternative with a bad Type Two update is simply to rewrite the bad record with correct data values, using a special error-correction job or process. If this is done you probably want to note that the record has changed, and you can do this by date-stamping the record with a **data warehouse updated date** that is separate from the versioning date.

WHICH DATES SHOULD BE KEPT?

FIRST, THE OBVIOUS!

Obviously you will need to keep all the **business event dates** that a user of the data warehouse might be expected to query or report. There may also be **external event dates** recorded in your business data – for example *date of birth* or *date passed driving test* which will almost always be needed.

When storing these, determine whether they ever change and how often; this is part of your general rate-of-change assessment for history management.

There is also no question that you will need the **transaction dates** for the business activities, in some businesses (such as Telco, for calls) this may be both **date and time**. However, depending on choices of time granularity these might be summarized into short time periods rather than distinct events; for example, phone calls in hour-of-day, rather than exact time.

DISCRETIONARY DATES

This group includes any date that you could derive from other records in the data warehouse. Each of these could be omitted, but there will be penalties for doing so – either in performance or in loss of older data.

If the derivation is expensive (involving large data volumes, such as retail basket purchases) then dates such as **last_purchase_date** should be kept as a matter of routine. However you could choose to derive and store them into a data mart instead, if that is the place where almost all queries will run.

If the derivation relies on historical data that might have a limited lifetime in the data warehouse, likewise this should be kept. Otherwise it will appear that the event never happened, rather than that it last happened a longer time ago. In this case a data mart would not offer alternative storage for the derived value, since it would probably be overwritten.

HISTORY DEPENDENT DATES

The next group depend on what kind of history the table maintains; the choices will be dictated by the chosen history management style.

Versioning history (Type Two style) will always require at least a single **updated date** for the record, and two **valid from / valid to dates** if using a normalized data model. You will also require two dates in a star schema if past point-in-time history queries are to be easily run in a single query.

Business history may also dictate a need for **effective from / effective to dates** when these may differ from the data warehouse versioning dates. This is especially true in certain sectors, like insurance, where value of business is counted over a period rather than a single time. It is also common when such changes are 'forward-dated' in operational systems.

Tables using Type Three history usually need a separate **field updated date** for the columns maintained this way. If more than one or two columns are involved then this starts to become a burden! There might then be a case for changing this into a versioning history with limited number of versions instead.

Best practise for Type One and Type Three is to always include the **updated date** for the record. Whilst this could be left out it is so useful that we generally keep it.

DATA WAREHOUSE RECONCILIATION DATES

Dates that allow data warehouse reports to be reconciled with operational systems and reports are in a complex area of consideration. These depend greatly on how and when operational systems can change data, and especially how they deal with error corrections and adjustments.

The simplest to keep are the operational systems' own **record updated date and time** values, which are often accompanied by the identity of the person entering the change or transaction for operational audit purposes. A good alternative is to keep the **extracted date and time** to identify when we read data from the operational system. This will usually be just as useful in reconciling systems and is controlled by us in our ETL processing. Since Extraction is normally timed to correspond with key business cycles, such as close-of-day, this gives a natural synchronisation point.

The use of **record updated date and time** becomes essential if the data warehouse becomes 'real-time' or 'near-time' updated during the active business day.

If an operational system performs *adjustments* – such as reassigning from business unit to another – then these must be captured into the data warehouse with the associated **adjustment transaction date**. You will also require the original and adjustment data to have an **extracted date** so that reports can be created including or excluding the later adjustment. You will probably also need an **original transaction date** so that the adjustment can be shown either in the original business period or in the period when the adjustment took place. You may even require an **adjustment effective date** if this has to be different from the adjustment transaction date.

If your design will allow it, you may wish to segregate adjustment data from 'normal' transactions so that other transaction records don't have to carry the penalty of these extra dates. In this case your main table will usually hold the adjusted value, so that routine use sees only the final data.

Similar requirements are imposed by error correction in the operational system. The equivalent dates for this are **error correction date** and **original transaction date**.

DATA WAREHOUSE AUDITING DATES

Use of these depends on how the data warehouse is managed. Often your auditing requirements will be met by dates you are already maintaining for history management (see above).

The exception will be if you have to allow error correction in the data warehouse by special update-in-place processes. History managed tables will then need not only their versioning dates but also a separate **dw-record updated date** for use when such updates occur. Note: for a fully auditable environment such updating should always be accompanied by an audit trail table that records the changes made and who authorized them!

SUMMARY OF THE MAJOR DATES USED IN A DATA WAREHOUSE

The following table summarizes the dates described above.

Data warehouse defined dates are created by the data management processes based on the processing dates; Business system defined dates come to the data warehouse in the extracted data from the operational systems.

Column purpose (with typical column name examples)	Description	Usage notes	Data warehouse or Business system defined? (D or B)
Data warehouse versioning (Valid_from_date, Valid_to_date)	Record validity period (when the data were in the data store)	Use for versioning data rows, usually mandatory when using 'type 2' style history. Valid_to_date will have a real date for old versions of a row; for the current row it may be null-valued or have a "high- value" date far in the future.	D
Extraction time-stamp (Extract_dttm)	When read from operational system	Optional, but can help reconcile data warehouse and operational data. Must be both date and time.	D
Loading time-stamp (Dw_Load_date)	When this row was loaded to the data warehouse	Optional. May duplicate <i>valid_from_date</i> if used in 'type 2' style history tables. Often better to use <i>dw_updated_date</i> as a more general-purpose column	D
Update time-stamp (Dw_Updated_date)	When this row changed in the data warehouse (error correction or the last date this record was changed by normal process)	Important for audit of data warehouse. Sometimes an alternative to <i>valid_from/valid_to</i> with Kimball-style 'type 2' history in star schema. Used with <i>valid_from/valid_to</i> when in-place error correction must be supported. May be mandatory if 'type 1' or 'type 3' history is used and it is important to know when the row was last changed.	D

<p>Various business event dates that do not change once recorded</p> <p>(names should reflect the information stored, such as: account_opened_date, account_first_used_date, transaction_date etc.)</p>	<p>Business real-world event dates</p>	<p>Consider which ones of these are of value to users of the data warehouse – important ones must be kept for use in query and reporting.</p> <p>Often we keep all such dates, in case they may be needed – but consider the rate of change when maintaining history. Frequently changed dates should be in separate tables when 'type 2' history is maintained</p> <p>Transaction dates can be summarized into short periods if granularity choice requires less detail.</p>	<p>B</p>
<p>Various business event dates that change</p> <p>(such as last_purchase_date, credit_rating_date, account_valuation_date)</p>	<p>Business dates, sometimes derivable from customer or company activity</p>	<p>When the corresponding event/action data is also in the data warehouse these may be derived only when needed, or derived into a data mart.</p> <p>However if derivation is expensive we often choose to maintain such derived values as last_purchase_date; storage cost is often small for these.</p> <p>Consider the rate of change when maintaining history. Frequently changed dates should be in separate tables when 'type 2' history is maintained</p>	<p>B</p>
<p>Various routine system event dates</p> <p>(such as record_entered_date, updated_date)</p>	<p>Business system internal event dates.</p>	<p>Consider whether these are needed at all in the data warehouse.</p> <p>If considered “purely operational” they can be left out.</p> <p>However they may be important when reconciling to operational systems.</p>	<p>B</p>
<p>Special-purpose system dates</p> <p>(such as adjustment_date, correction_date)</p>	<p>Business system dates that override previous transactions.</p> <p>Note, these may need to be derived from transaction codes along with a simple date-stamp</p>	<p>Important for reconciling with operational systems.</p> <p>May cause removal of 'bad' original transaction from data warehouse</p> <p>Always need special handling</p>	<p>B</p>

Business time periods (Effective_from_date, Effective_to_date or Begin_date, End_date)	Business effective period	Use for time-spanned business data, such as insurance cover, duration of a sales price promotion or subscription to a service. May be open-ended period dates (the End_date remains null-valued or high-valued until an actual date is provided) or closed periods (End-dt has a real date to end the period) – depending on the business context. Usually mandatory in business that record these; unnecessary in those that do not.	B

ADVANCED HISTORY, ‘THE WORKS’

In a data warehouse that sets out to record everything you could expect to have all of the columns shown in the table above. Whether you go this far depends on your business and data warehouse requirement – assess according to need, and try to understand how each date will be used before adding it to your model. Don’t just add them all regardless!

CONCLUSION

How many dates you keep, and which dates you choose, will be dependent on three factors:

- The natural use of dates in the business – what they represent, how are they recorded, when they occur
- Whether you require all historical states of data to be tracked in the data warehouse
- How you intend to query and use date information from the data warehouse
- Whether the data warehouse must reconcile with operational systems or reports

Beware impacts on your data and performance:

- Avoid mixing fast-changing data (dates or other variables) with slow-change history, break them out into a separate table
- Derive customer-activity-related dates and store in the data warehouse or mart rather than dynamically when queried

A FEW RULES-OF-THUMB

To help you decide what you will keep:

- Understand how data changes its values – in the real world, in operational systems.
- Most importantly, understand *when* and *how often* data changes. Use a time-line to assist in this.
- Determine how much you need to represent history in the data warehouse and
- Understand with what time-perspectives it will be queried
- Determine your reconciliation and auditing requirements – cross-refer dates to operational systems if necessary

REFERENCES

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

ACKNOWLEDGMENTS

I would like to acknowledge the patience and perseverance of many co-workers over the years who have raised questions or provided answers, and helped to work through the consequences of design decisions involving dates and history.

You are too many to mention individually, but you know who you are – thank you all!

CONTACT INFORMATION

Your comments and questions are valued and encouraged.

Contact the author:-

Steve Morton

Applied System Knowledge Ltd.

51 Blandy Road

Henley-on-Thames

RG9 1QB

United Kingdom

Work Phone: +44 1491 411977

Email: steve.morton@appliedsystem.co.uk

Web: www.appliedsystem.co.uk

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.