Analytics assumes that data movement will follow a linear path into a historical repository to represent the correct sequence of events. That is, if today is Thursday and we have loaded data for Wednesday we then do not expect data from Tuesday to arrive today because all the data up until today should have already been loaded. Whether it be a missing batch file or a missing data record - a Tuesday file may have arrived and loaded but could have been provided without a complete set of records - there are a myriad of reasons why these scenarios might occur (a locked record due to an update in the database may cause a push file produced from the source system to be incomplete). These are real scenarios in loading a data warehouse and it can skew what we know of a business entity (e.g. customer) and may cause erroneous analytics to be derived for that business entity or worse, the wrong facts are reported to a regulatory body or to the customers themselves. The late record is (in data parlance) known as “out-of-sequence” data and to us data professionals this is a **time crime**. The repercussions could have legal ramifications, reputation risk, loss of market share (etc.) if the analytics is influenced by something that might be considered as trivial as data sequencing issues.

The crime scene: “out of sequence” data in data vault 2.0.

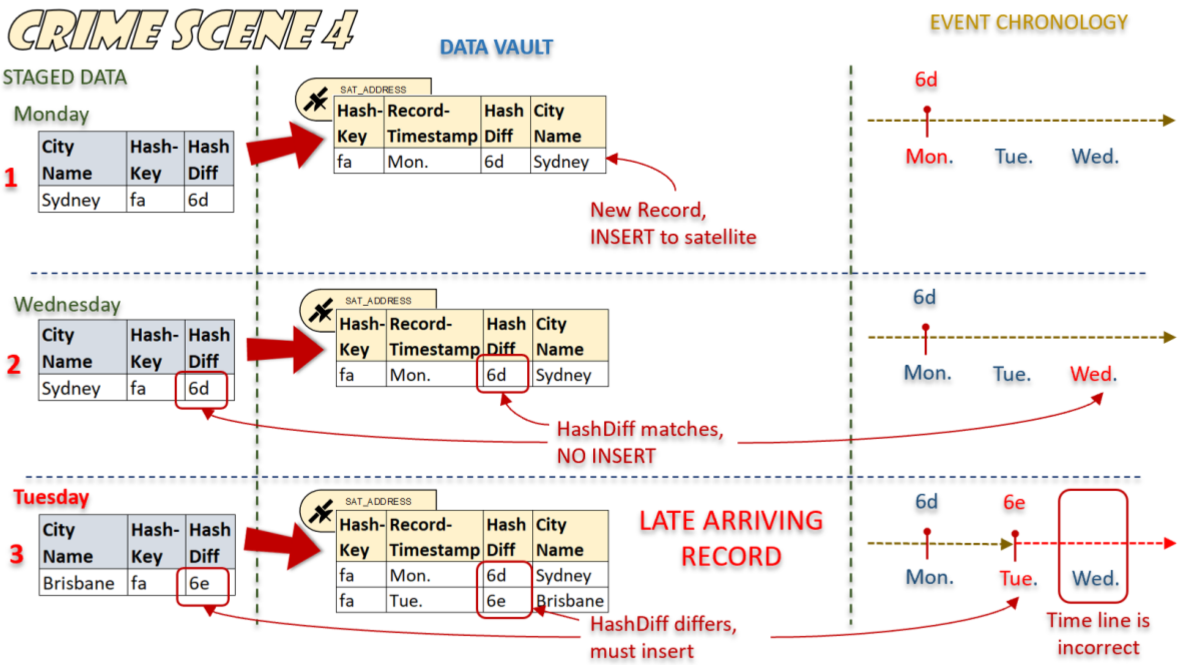
Let’s set the crime scene; we are loading a customer hub and a customer address satellite.

1. For hash key ‘159’ (hashed business key) the data is loaded on Monday with a ‘City Name’=’Sydney’. We have never seen this business entity before, and the address satellite table does not have an entry for this hash key; therefore, we insert the record into the satellite with its record hash (hashdiff) of ‘6d’ (hashed ‘Sydney’= ‘6d’). On Tuesday no file was pushed to the data warehouse from the source system; therefore, there is no update in the address satellite for hash key ‘159’.
2. Wednesday’s file arrived and details of hash key ‘159’ is present in that file; ‘City Name’=’Sydney’. For hash key ‘159’ Monday’s ‘City Name’ is the same as Wednesday’s ‘City Name’ (staged hashdiff ‘6d’ = satellite’s hashdiff ‘6d’) and therefore there is no change and no insert is performed to the address satellite for that hash key.
3. On Thursday, Tuesday’s push file arrives with data for hash key ‘159’; ‘City Name’=’Brisbane’ with a hashdiff = ‘6e’. A comparison is performed between the

staged file and the 'older' record in the address satellite (Monday) and the 'City Name' differs ('Sydney' <> 'Brisbane' | '6d' <> '6e'); therefore, the late arriving record is inserted to show the change. On Friday when we query the address satellite to retrieve the **current** address for hash key '159' the 'City Name' value will return 'Brisbane'.

This is **incorrect**. According to the staged data provided from the source system on Wednesday the staged 'City Name' is 'Sydney' for that hash key; but the latest 'City Name' in the satellite for that hash key is 'Brisbane'.

A brief walkthrough on the crime scene below; on the left is the staged data (deltas), the middle is the data vault domain and on the right is the timeline represented in the satellite for the hash key. Each day (1-Monday, 2-Wednesday, 3-Thursday) is horizontally represented. (3) contains the staged late arriving record; the day the time crime is committed. Below each scene we describe the event chronology.



Monday (1) staged data:

City Name	Hash-Key	Hash Diff
Sydney	fa	6d

SAT_ADDRESS:

Hash-Key	Record-Timestamp	Hash Diff	City Name
fa	Mon.	6d	Sydney

Wednesday (2) staged data:

City Name	Hash-Key	Hash Diff
Sydney	fa	6d

SAT_ADDRESS:

Hash-Key	Record-Timestamp	Hash Diff	City Name
fa	Mon.	6d	Sydney

Tuesday (3) staged data:

City Name	Hash-Key	Hash Diff
Brisbane	fa	6e

SAT_ADDRESS:

Hash-Key	Record-Timestamp	Hash Diff	City Name
fa	Mon.	6d	Sydney
fa	Tue.	6e	Brisbane

1. Monday is an INSERT because the record by hash key is new and there is no hashdiff to run a comparison for this hash key.
2. Wednesday has no INSERT because the staged hashdiff and the satellite hashdiff for the hash key matches; Monday's hashdiff of '6d' = Wednesday's hashdiff of '6d'.
3. Late arriving is an INSERT because the hashdiff is different when compared to the older record (Monday hashdiff='6d' and Tuesday hashdiff='6e'); it now means the timeline recorded for this hash key is wrong as depicted by the timeline on the right.

How do we dynamically deal with this crime? Here we propose a data driven approach.

A few things you need to know about data vault 2.0

Data Vault 2.0 is an INSERT-ONLY paradigm. Data on Big Data platforms is immutable and update operations are performed by persisting the data to a new persistent state (DELETE and INSERT occurs in the background). This makes UPDATEs expensive and therefore avoided especially in environments managing millions of records. For data vault 2.0 this means that records in a satellite table have a business start date but no end date. Instead end dates are simulated using the LEAD() windowing function when querying the data. For example:

```
, coalesce(lead(START-DATE) over (partition by HASH-KEY order by START-DATE), cast('31-DEC-9999 23:59:59.998' as date)) END-DATE
```

This feature in data vault 2.0 is critical because it means that the end-date is dynamic and will change dependent on records inserted after the current record by the same hash-key.

The crime fighter: Record Tracking Satellite eXtended.

Record Tracking Satellite (RTS) in data vault is used to **record every instance a business key** or a relationship was sent from the source system. This can be incredibly handy in detecting if the source system has deleted a business entity or relationship or it can even be useful in detecting if a record affected by GDPR's right to erasure ("right to be forgotten") is still being sent by the source. Data aging rules can be applied to those keys using RTS to identify what may be deleted business entities in the source system.

This blog discusses extending the usage of RTS to **record every instance a record** was sent from the source system. Let's revisit RTS table structure and introduce the column needed to extend RTS (now called XTS) for time crime fighting.

Column	What it does
Hash-Key	Hashed representation of business keys or relationship stored as binary data type for optimal join performance.
Load-Timestamp	The timestamp marked on each record recording the precise date & time the record has entered the data warehouse; the processing time.
Record-Timestamp	The applicable timestamp recording of when the event occurred in the source system; the event time.
Hash Difference	(Amended) The record digest of all attributes on a record except for the business keys, load-timestamp and record-timestamp and other metadata tagged onto each record in data vault.
Record Source	File/Table name of the source system that provided the record that may include file location or schema name.
Record Target	(New) The satellite table associated with the record.

**Data Vault 2.0 has a few more metadata tags but these listed above are needed for illustration.*

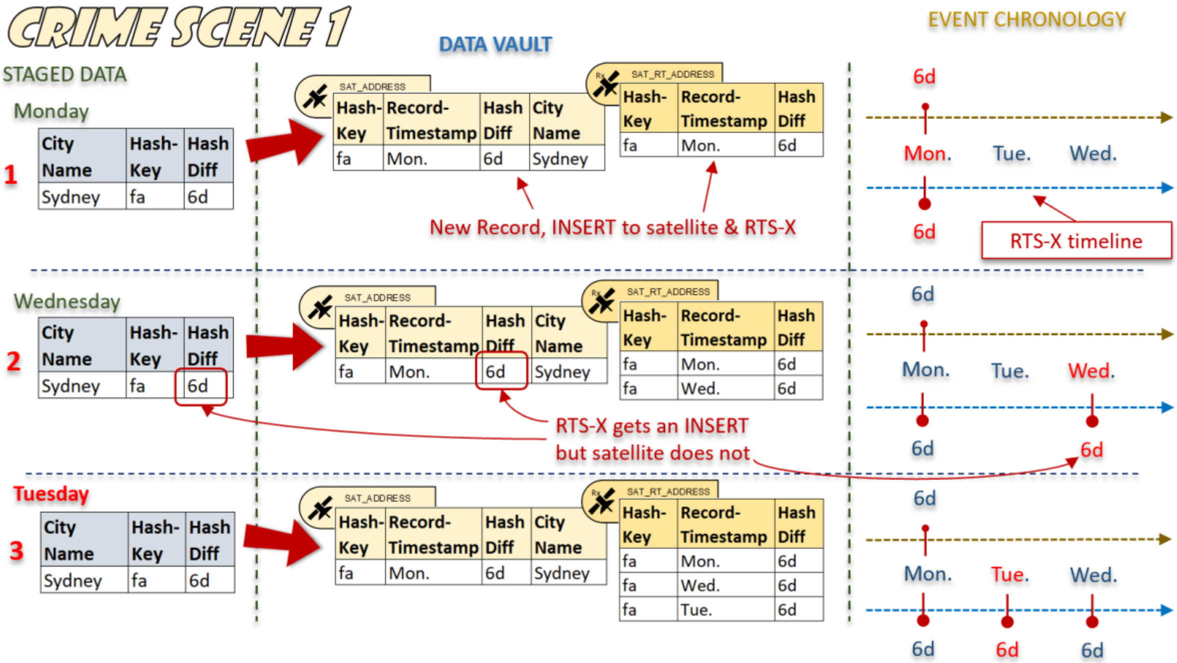
How XTS fights time crime.

Without the need to manually deal with time crime that we saw in the crime scene introduced, XTS will continuously enforce time crime prevention as and when they occur. They are the good Samaritans that react when needed and you don't need to call anyone or stop anything while the time crime is being policed.

We will talk about the various crime scenes XTS is designed to deal with by exploring the customer address scenario we introduced earlier. Think of XTS as those citizens on patrol going about their business of recording record hashes, assisting in detecting changes for satellites and immediately act when a time crime is detected.

Petty crime: late arriving records that does not need special treatment in data vault 2.0. Let's see how XTS gets involved to correct these timelines. We have added XTS to each crime scene in the data vault column and a blue timeline for XTS under the event chronology column.

Crime Scene 1: late-arriving record is the same as the older record and the newer records.



Monday — 1

City Name	Hash-Key	Hash Diff
Sydney	fa	6d

SAT_ADDRESS

Hash-Key	Record-Timestamp	Hash Diff	City Name
fa	Mon.	6d	Sydney

SAT_RT_ADDRESS

Hash-Key	Record-Timestamp	Hash Diff
fa	Mon.	6d

New Record, INSERT to satellite & RTS-X

Wednesday — 2

City Name	Hash-Key	Hash Diff
Sydney	fa	6d

SAT_ADDRESS

Hash-Key	Record-Timestamp	Hash Diff	City Name
fa	Mon.	6d	Sydney

SAT_RT_ADDRESS

Hash-Key	Record-Timestamp	Hash Diff
fa	Mon.	6d
fa	Wed.	6d

RTS-X gets an INSERT but satellite does not

Tuesday — 3

City Name	Hash-Key	Hash Diff
Sydney	fa	6d

SAT_ADDRESS

Hash-Key	Record-Timestamp	Hash Diff	City Name
fa	Mon.	6d	Sydney

SAT_RT_ADDRESS

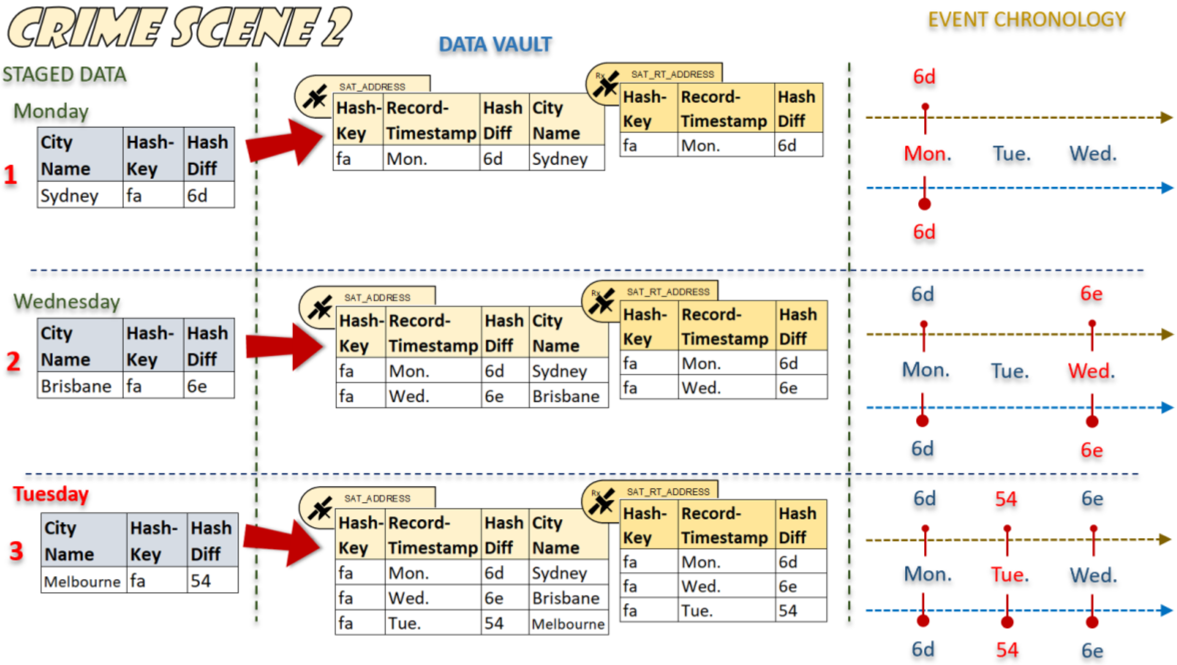
Hash-Key	Record-Timestamp	Hash Diff
fa	Mon.	6d
fa	Wed.	6d
fa	Tue.	6d

1. New record arrives; an INSERT is performed.
2. Wednesday's staged record is the same as the previous record, no INSERT is performed because the value is the same as what is in the satellite; 'Sydney'.

3. The late arriving record from 'Tuesday' is compared to the older record and it is the same and therefore no INSERT is performed.

Notice that the event chronology shows that we have an INSERT for every record received in the XTS timeline but only the *changed* record in the satellite timeline.

Crime Scene 2: late arriving record is different to the older record and the newer record.



Monday

City Name	Hash-Key	Hash Diff
Sydney	fa	6d

SAT_ADDRESS

Hash-Key	Record-Timestamp	Hash Diff	City Name
fa	Mon.	6d	Sydney

SAT_RT_ADDRESS

Hash-Key	Record-Timestamp	Hash Diff
fa	Mon.	6d

Wednesday

City Name	Hash-Key	Hash Diff
Brisbane	fa	6e

SAT_ADDRESS

Hash-Key	Record-Timestamp	Hash Diff	City Name
fa	Mon.	6d	Sydney
fa	Wed.	6e	Brisbane

SAT_RT_ADDRESS

Hash-Key	Record-Timestamp	Hash Diff
fa	Mon.	6d
fa	Wed.	6e

Tuesday

City Name	Hash-Key	Hash Diff
Melbourne	fa	54

SAT_ADDRESS

Hash-Key	Record-Timestamp	Hash Diff	City Name
fa	Mon.	6d	Sydney
fa	Wed.	6e	Brisbane
fa	Tue.	54	Melbourne

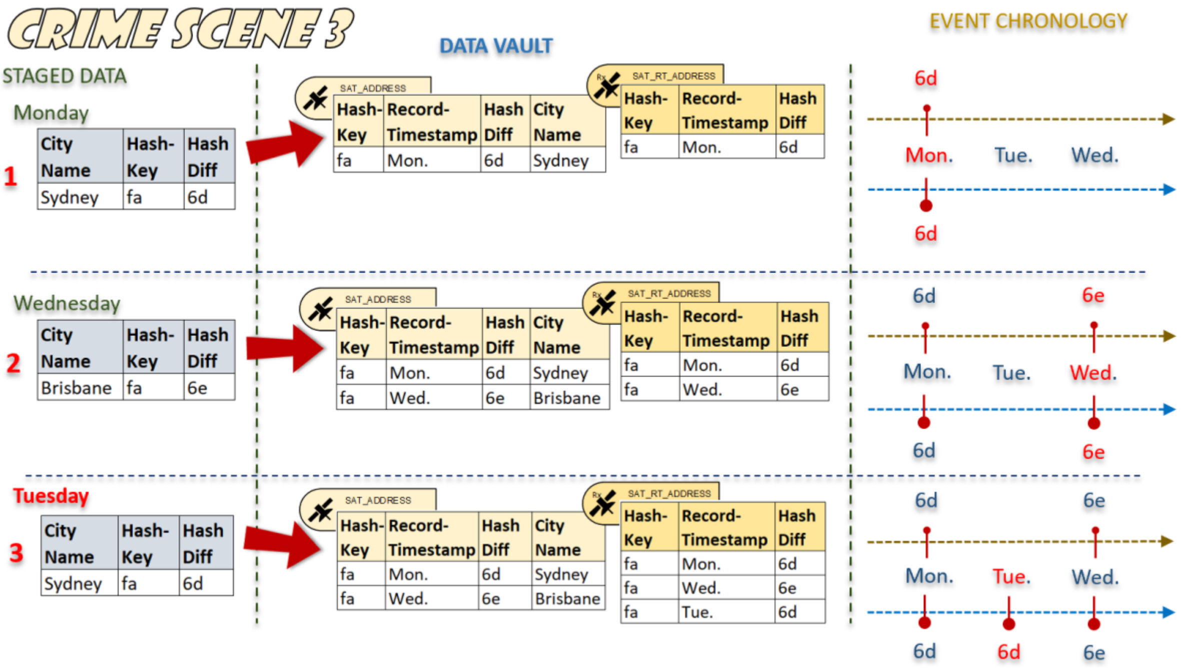
SAT_RT_ADDRESS

Hash-Key	Record-Timestamp	Hash Diff
fa	Mon.	6d
fa	Wed.	6e
fa	Tue.	54

1. New record arrives; an INSERT is performed.
2. Wednesday's record arrives and it is different to Monday's record ('Brisbane' and 'Sydney'). We INSERT the record for 'Brisbane' because the hashdiff generated for 'Brisbane' differs to what was recorded on Monday; the hashdiff for 'Sydney'.
3. The late arriving record is compared to Monday's recorded hashdiff and it differs too ('Melbourne' vs 'Sydney'). An INSERT is performed.

Notice that every day's hashdiff differed and so the timelines for XTS and the satellite are identical. Remember that the satellite will contain the details that make up the hashdiff and XTS only records the delta hashdiffs. Although the example only has 'City Name' as an attribute in reality a satellite's number of attributes can be as wide as the platform supports. The generated hashdiff will always be of the same consistent size regardless of the number of attributes that are hashed together.

Crime Scene 3: late arriving record is the same as the older record but different to the newer record



CRIME SCENE 3

STAGED DATA

DATA VAULT

EVENT CHRONOLOGY

Monday

1

City Name	Hash-Key	Hash Diff
Sydney	fa	6d

SAT_ADDRESS

Hash-Key	Record-Timestamp	Hash Diff	City Name
fa	Mon.	6d	Sydney

SAT_RT_ADDRESS

Hash-Key	Record-Timestamp	Hash Diff
fa	Mon.	6d

Wednesday

2

City Name	Hash-Key	Hash Diff
Brisbane	fa	6e

SAT_ADDRESS

Hash-Key	Record-Timestamp	Hash Diff	City Name
fa	Mon.	6d	Sydney
fa	Wed.	6e	Brisbane

SAT_RT_ADDRESS

Hash-Key	Record-Timestamp	Hash Diff
fa	Mon.	6d
fa	Wed.	6e

Tuesday

3

City Name	Hash-Key	Hash Diff
Sydney	fa	6d

SAT_ADDRESS

Hash-Key	Record-Timestamp	Hash Diff	City Name
fa	Mon.	6d	Sydney
fa	Wed.	6e	Brisbane

SAT_RT_ADDRESS

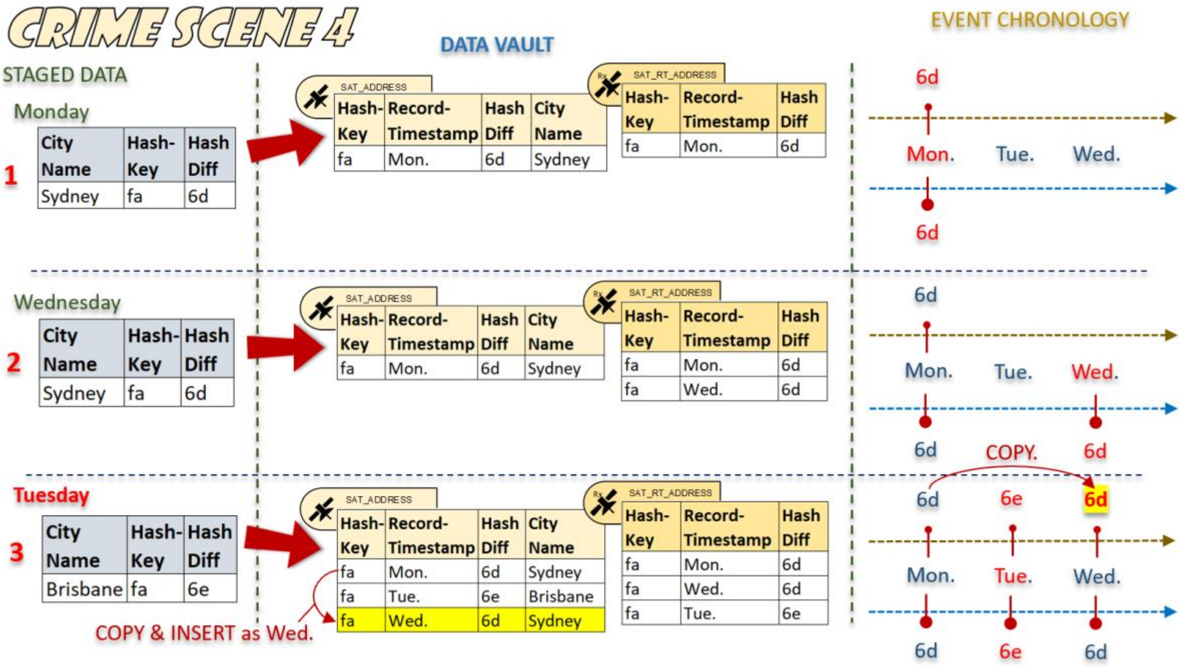
Hash-Key	Record-Timestamp	Hash Diff
fa	Mon.	6d
fa	Wed.	6e
fa	Tue.	6d

1. New record arrives; an INSERT is performed.
2. Wednesday’s hashdiff differs to that of Monday’s hashdiff and therefore an INSERT operation is performed to the satellite.
3. The late arriving record is the same as the previous record (‘Sydney’ and ‘Sydney’); this means that no INSERT is performed because no actual change has occurred between Monday and Tuesday for this hash key.

The chronology shows that only XTS has recorded that there was a delta record staged from the source system, the satellite shows that there was no change between Monday and Tuesday.

Felonies: felonies are more serious and need an additional code snippet to solve the time crime.

Crime Scene 4: the older and newer records are the same, but the late arriving record is different to both.



Monday (1) – STAGED DATA

City Name	Hash-Key	Hash Diff
Sydney	fa	6d

SAT_ADDRESS

Hash-Key	Record-Timestamp	Hash Diff	City Name
fa	Mon.	6d	Sydney

SAT_RT_ADDRESS

Hash-Key	Record-Timestamp	Hash Diff
fa	Mon.	6d

Wednesday (2) – STAGED DATA

City Name	Hash-Key	Hash Diff
Sydney	fa	6d

SAT_ADDRESS

Hash-Key	Record-Timestamp	Hash Diff	City Name
fa	Mon.	6d	Sydney

SAT_RT_ADDRESS

Hash-Key	Record-Timestamp	Hash Diff
fa	Mon.	6d
fa	Wed.	6d

Tuesday (3) – STAGED DATA

City Name	Hash-Key	Hash Diff
Brisbane	fa	6e

SAT_ADDRESS

Hash-Key	Record-Timestamp	Hash Diff	City Name
fa	Mon.	6d	Sydney
fa	Tue.	6e	Brisbane
fa	Wed.	6d	Sydney

SAT_RT_ADDRESS

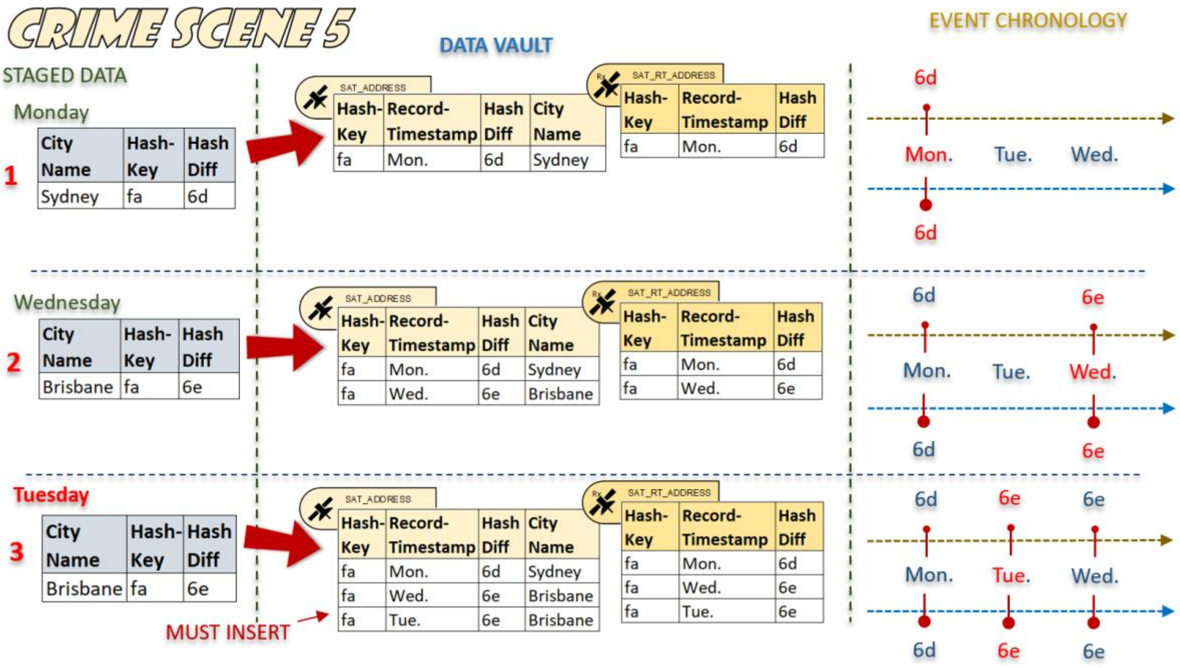
Hash-Key	Record-Timestamp	Hash Diff
fa	Mon.	6d
fa	Wed.	6d
fa	Tue.	6e

This is the example we described earlier in the blog.

1. New record arrives; an INSERT is performed.
2. Like crime scene 1 the record is the same as Monday's record ('Sydney' and 'Sydney') and therefore no INSERT is performed.
3. The late arriving record ('Tuesday') differs from Monday's record, BUT Monday's hashdiff and Wednesday's hashdiff are the same. We know this because this fact is recorded in XTS (Monday '6d' = Wednesday '6d'). We use this comparison to COPY Monday's record from the satellite and INSERT it as Wednesday as if we had received the records in the correct order.

The event chronology will now show that the timeline was corrected.

Crime Scene 5: late arriving record is the same as the newer record but different to the older record.



Monday

City Name	Hash-Key	Hash Diff
Sydney	fa	6d

SAT_ADDRESS

Hash-Key	Record-Timestamp	Hash Diff	City Name
fa	Mon.	6d	Sydney

SAT_RT_ADDRESS

Hash-Key	Record-Timestamp	Hash Diff
fa	Mon.	6d

Wednesday

City Name	Hash-Key	Hash Diff
Brisbane	fa	6e

SAT_ADDRESS

Hash-Key	Record-Timestamp	Hash Diff	City Name
fa	Mon.	6d	Sydney
fa	Wed.	6e	Brisbane

SAT_RT_ADDRESS

Hash-Key	Record-Timestamp	Hash Diff
fa	Mon.	6d
fa	Wed.	6e

Tuesday

City Name	Hash-Key	Hash Diff
Brisbane	fa	6e

SAT_ADDRESS

Hash-Key	Record-Timestamp	Hash Diff	City Name
fa	Mon.	6d	Sydney
fa	Wed.	6e	Brisbane
fa	Tue.	6e	Brisbane

SAT_RT_ADDRESS

Hash-Key	Record-Timestamp	Hash Diff
fa	Mon.	6d
fa	Wed.	6e
fa	Tue.	6e

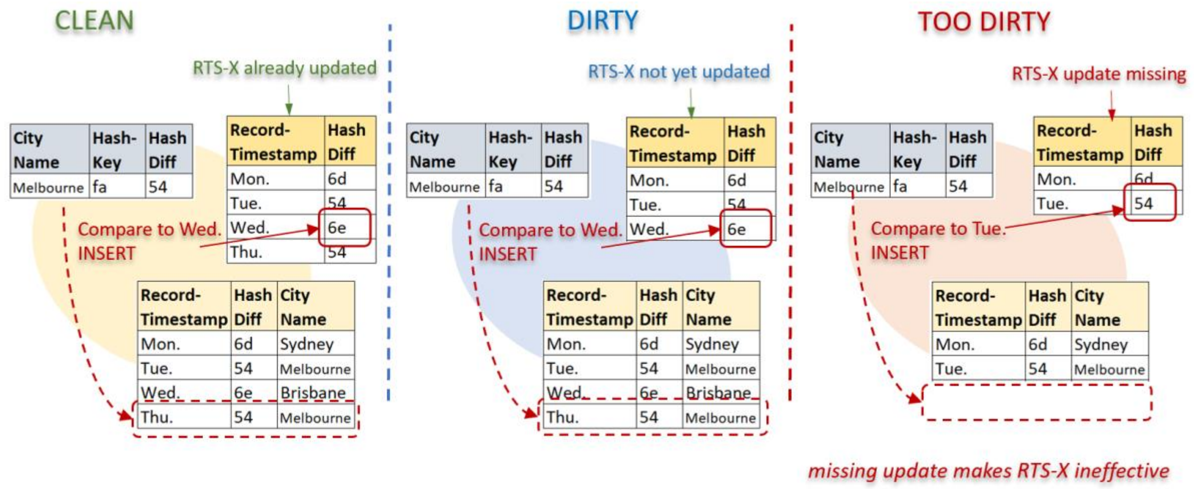
1. New record arrives; an INSERT is performed.
2. Wednesday's record differs to that of Monday ('Brisbane' vs 'Sydney') and thus an INSERT is performed.
3. The staged late arriving record carries the value 'City Name' = 'Brisbane'. This differs to Monday's record too and in fact shows that the change to the hash key occurred earlier than Wednesday. An INSERT operation must be performed to show this fact but consequently the satellite will contain a duplicate.

The event chronology now shows that a duplicate record now exists in the satellite as Tuesday and Wednesday's hashdiff is the same. This duplicate is intentional but unlike dimensional modelling the impact is not exaggerated and in fact can be discarded with a well-constructed view and/or an update to an adjacent point in time (PIT) table.

Keeping Data Vault 2.0 free of Time Crime

Now that you have seen the value of XTS let's discuss time crime **prevention**. XTS is law enforcement and to ensure that all satellites are protected against time crime we will have an XTS posted at every hub and link apart from the non-historized link. The solution proposed in this blog is completely data-driven but with a minor caveat; implementing this data-driven self-healing capability does introduce a bi-table dependency.

A dirty read from XTS is ok, a clean read is ok too, but a read that is too dirty is not ok (pun intended).



City Name	Hash-Key	Hash Diff
Melbourne	fa	54

Record-Timestamp	Hash Diff
Mon.	6d
Tue.	54
Wed.	6e
Thu.	54

Record-Timestamp	Hash Diff	City Name
Mon.	6d	Sydney
Tue.	54	Melbourne
Wed.	6e	Brisbane
Thu.	54	Melbourne

City Name	Hash-Key	Hash Diff
Melbourne	fa	54

Record-Timestamp	Hash Diff
Mon.	6d
Tue.	54
Wed.	6e

Record-Timestamp	Hash Diff	City Name
Mon.	6d	Sydney
Tue.	54	Melbourne
Wed.	6e	Brisbane
Thu.	54	Melbourne

City Name	Hash-Key	Hash Diff
Melbourne	fa	54

Record-Timestamp	Hash Diff
Mon.	6d
Tue.	54

Record-Timestamp	Hash Diff	City Name
Mon.	6d	Sydney
Tue.	54	Melbourne

It's about the past and the future but not the present records in XTS. XTS has been deployed at a client to stop time crime from occurring in satellites, multi-active satellites, satellites with dependent-child keys, business vault satellites, status-tracking satellites, record tracking satellites (the regular ones) but not effectivity satellites. Effectivity satellites derive start and end dates that make it too difficult to implement using XTS. The time correction in effectivity satellites must occur in the start and end date derivation already and therefore the additional code snippet used with XTS will never be utilised.
XTS can be used conditionally, meaning if your ingestion pipeline sees a late-arriving record then XTS can be utilised instead of the regular ingestion patterns for the noble satellites. But then again if you do that you do have two sets of code to maintain. Code using XTS to update satellites and code that does not. We have seen XTS outperform the regular loading patterns for staged files as big as 20 million records but degrades thereafter (on SQL Server). That is where things like partition strategies can be used to keep the train rolling at the pace you want.

Wrapping up the scene
This is a data driven approach and not without caveats; as you have seen this makes the satellite loading dependent on how up to date XTS is.
Bi-table dependency - if your operational processes ensure that time crime is prevented by checking that the next file in a sequence is the next file expected then XTS will not be needed; but it does mean you have to wait for the file to arrive before you can continue processing subsequent files.
Storage - compression is required and XTS is wafer thin and the more history it has the more effective it can be. In a historical backloading scenario you may consider loading to a new satellite instead and therefore not need XTS to ensure adequate historical blending occurs during the backload.
Number of tables – yes, an additional table to each hub and link is proposed however in the era of commodity hardware and scalability made available through storage mediums such as AWS S3, is that really a worry?
Availability of data – if your platform is constrained and inserts to XTS hold up the availability of the satellite XTS is associated with then consider enforcing that the satellite is updated before the XTS, ala dirty reads.
Table locking – a central XTS to each hub or link will have the same table contention as hubs or links and therefore a form of pooling is required that ensures not more than one task is

updating XTS. Alternatively you could deploy an XTS to each satellite around a hub and link and this would negate the need to record which satellite a hashdiff in XTS is associated with.

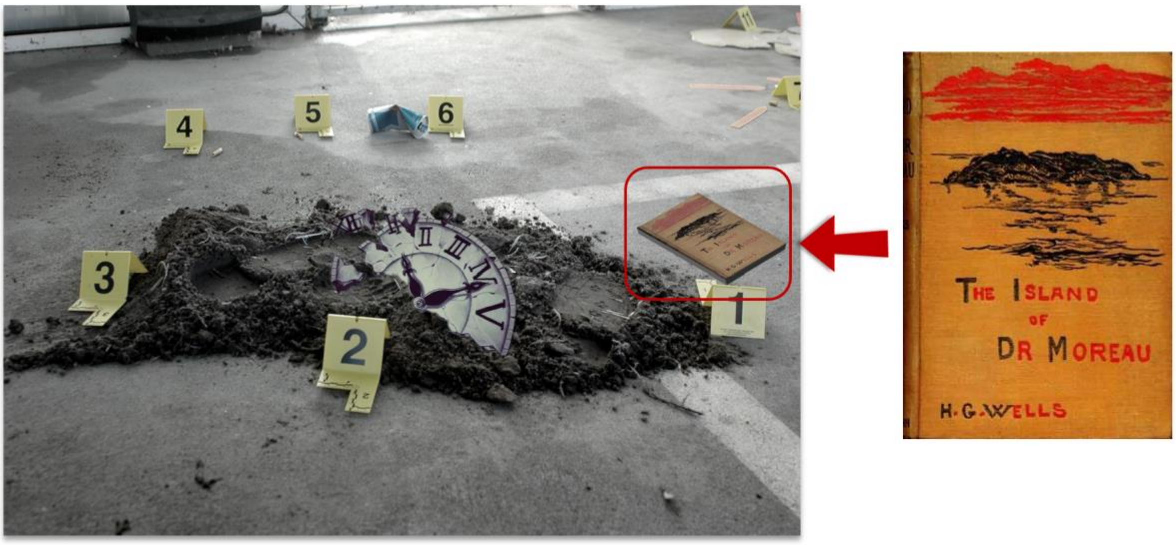
Reports, dashboards and views – updates to the past may cause a time-ripple. These time ripples may have to be explained; do not allow this approach to allow automated operational procedures to suddenly become lacking because "XTS will fix it".

Snapshots need updating – depending on how far back your logarithmic point-in-time tables (PITs) go you may need to swap out and swap in an updated set of keys and dates. There is still a cost to performing back-in-time corrections to the data warehouse.

XTS is merely an unsung hero in the fight against time crime.

Not for streaming data – as discussed non-historized links are not a candidate to have an XTS posted to. These links contain descriptive attributes within the link itself and never expect the number of attributes in the stream to change. The data loaded to non-historized links are immutable, transactional and data ordering is expected to be managed **before** the data reaches the data vault.

Did you find the clue in the crime scene?



This extension to data vault 2.0 was presented by Certus Solutions at wwdvc 2019 and I am a co-author.