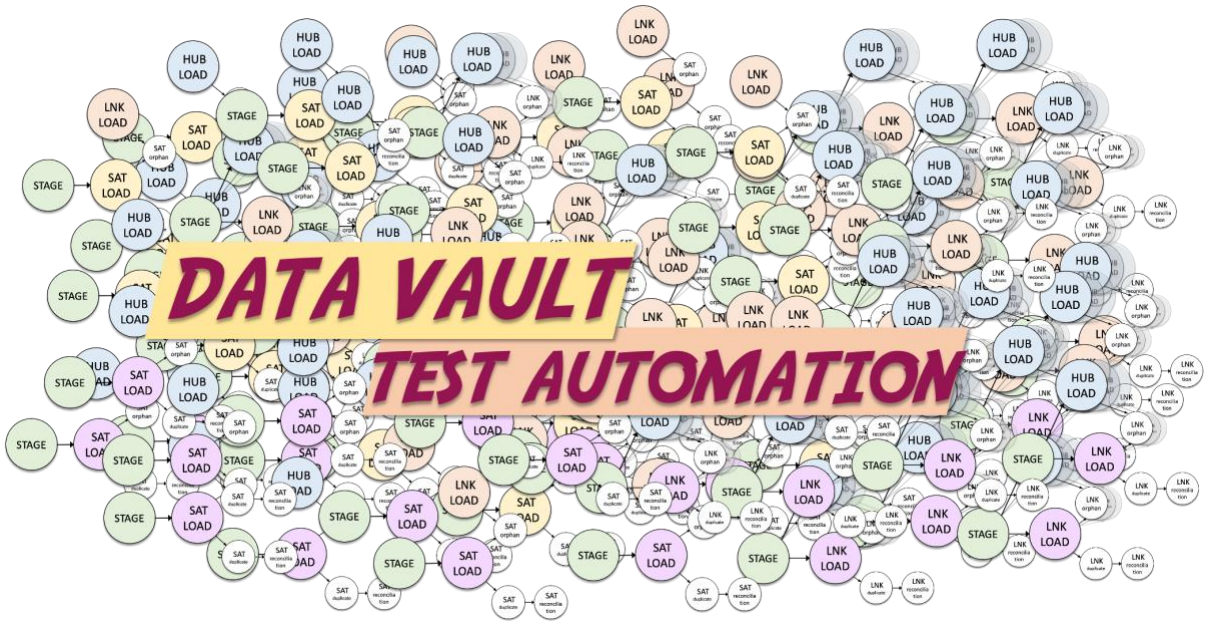


Data Vault Test Automation

Modern day data analytics platforms sometimes do not enforce ***referential integrity*** (foreign key ***constraints***), the idea is that such restrictions enforce strict rules on the underlying data that the latency from business event to analytic value suffers. BUT we still need to have ***faith*** on the data that is loaded onto the platform, no less!
Data Vault 2.0 ***does not impose restrictions*** either! It is as scalable and flexible as the platforms hosting it. A data vault has ***repeatable patterns*** for data modelling, for data architecture, for data loading and of course, ***test automation***! Each component is an ***independent*** unit of work, certain loading patterns can also be ***anti-patterns*** in certain situations, let's explore that and a framework for efficient test automation!

Post pre-reading:

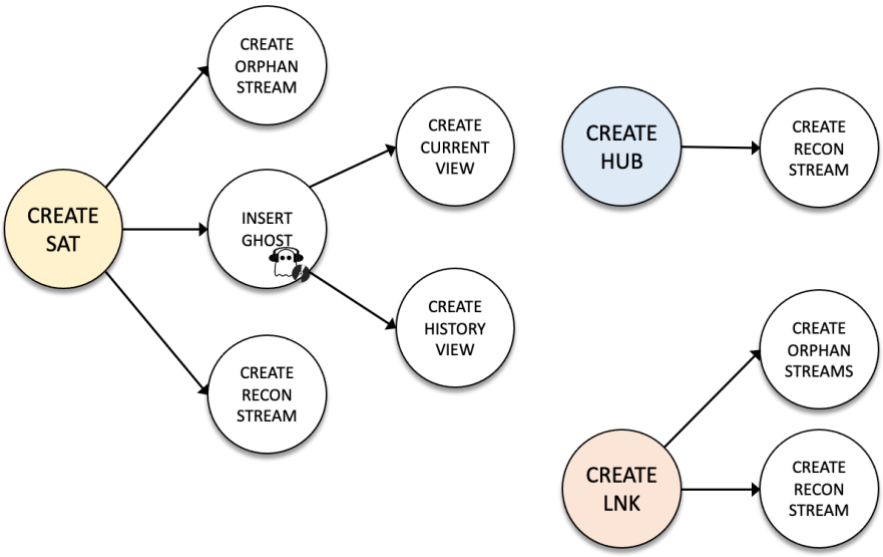
- Snowflake Streams: bit.ly/3hdW79o
- Snowflake Multi-table insert: bit.ly/3h97Ypz
- Read isolation: bit.ly/2UlGFz8
- Direct Acyclic Graphs (DAG): bit.ly/369Gb1F
- Apache Airflow pools: bit.ly/3dGBhxj

Patterns, Patterns, Patterns!

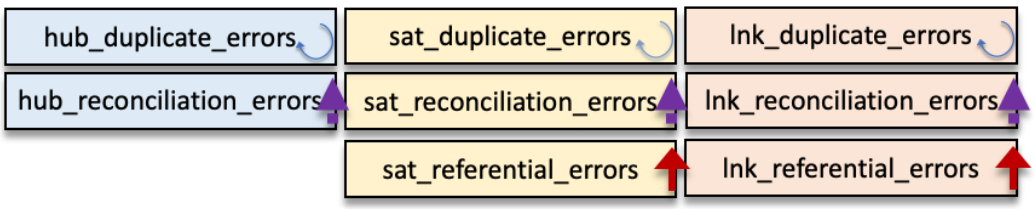
Data Vault 2.0 is delivered in patterns.

- Patterns for **modelling**,
    - **Hub tables**, *Business Objects* based on *Business Architecture* (the first block of *Enterprise Architecture)*
    - **Link tables**, *Unit of Work* of *Business Processes* that influence *Business Objects* and may take a *Business Object* from one state to another
    - **Satellite tables**, change descriptive details of either the *Business Objects* or the *Unit of Work*.
- Patterns for **table creation,**
    - **Hub** - define hub table with standard data vault metadata tags and the business key.
        - Define a *hub reconciliation stream* on the hub table

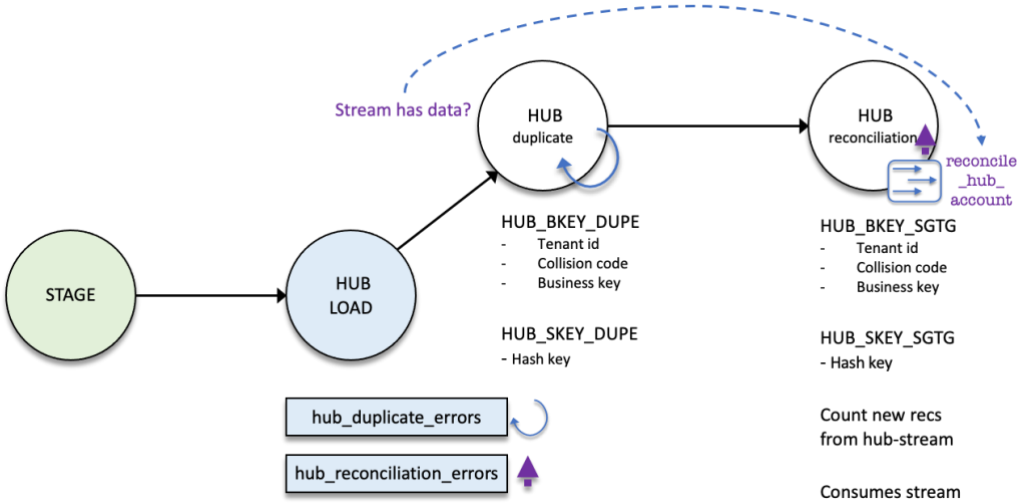
- **Link** - define link table with standard data vault metadata tags
  - Define a *link reconciliation stream* on the link table
  - Define an *orphan stream* on the link table for every parent business key the link relates to
- **Satellite** - define satellite table with standard data vault metadata tags and column attributes
  - Insert a *ghost record* (see: bit.ly/3vjTXdg)
  - Create a *current view* on the satellite
  - Create a *history view* on the satellite
  - Define a *satellite reconciliation stream* on the satellite table
  - Define an *orphan stream* on the satellite table for the parent hub or link table the satellite table relates to



- Patterns for **reconciliation tables**

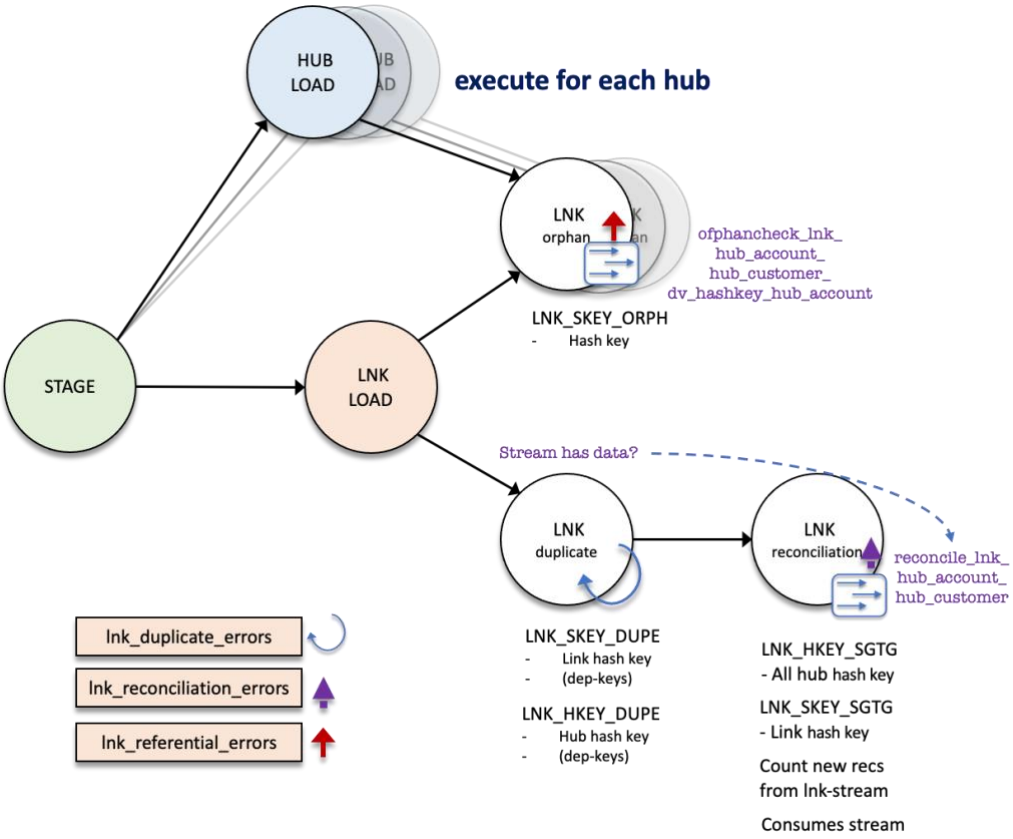


- Patterns for **automation & testing**,
  - **Hub load** + tests



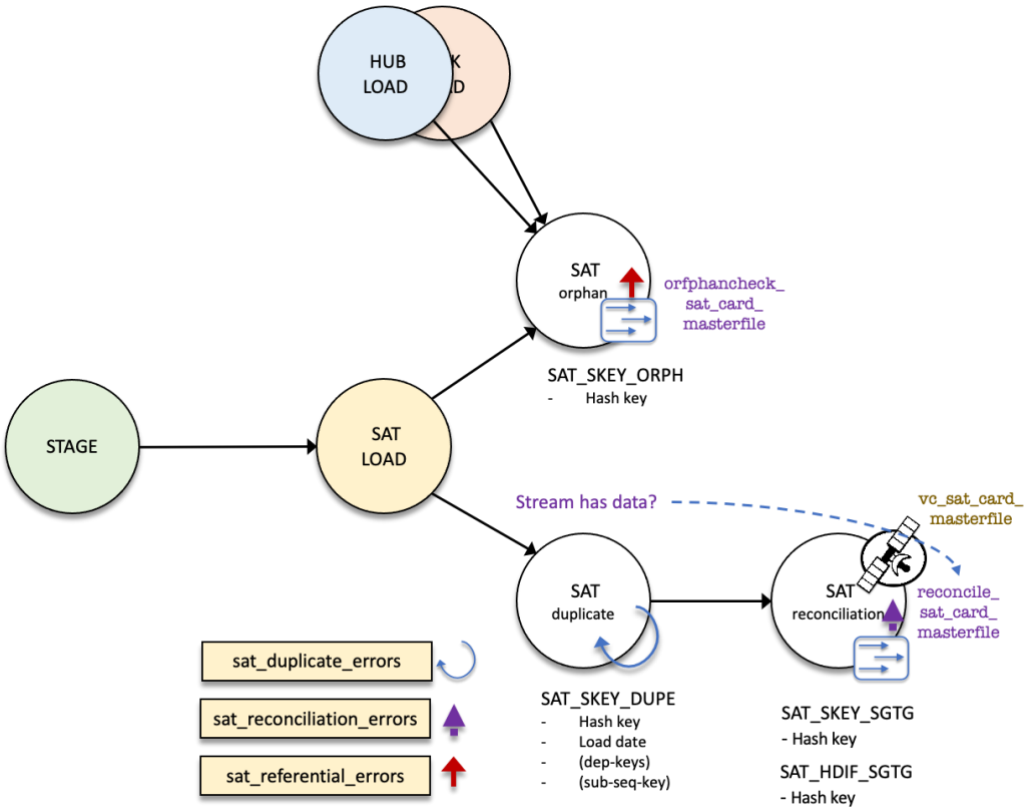
    - If recon stream has data, then do:
      - ***Check*** business key duplicates
      - ***Check*** hash key duplicates
      - Reconcile staged ***hash and business keys*** are indeed in the hub table
      - Count ***new keys*** in stream
      - Count ***total keys*** in hub table after load

  - **Link load** + tests



- If recon stream has data, then do:
  - ***Check*** hash key duplicates
  - Reconcile staged ***link and hub hash keys*** are indeed in the link table
  - Count ***new keys*** in stream
  - Count ***total keys*** in link table after load
- For every related hub ***stream*** check:
  - Is the link table's ***hub hash key*** in the ***related*** hub table

- Satellite load + tests

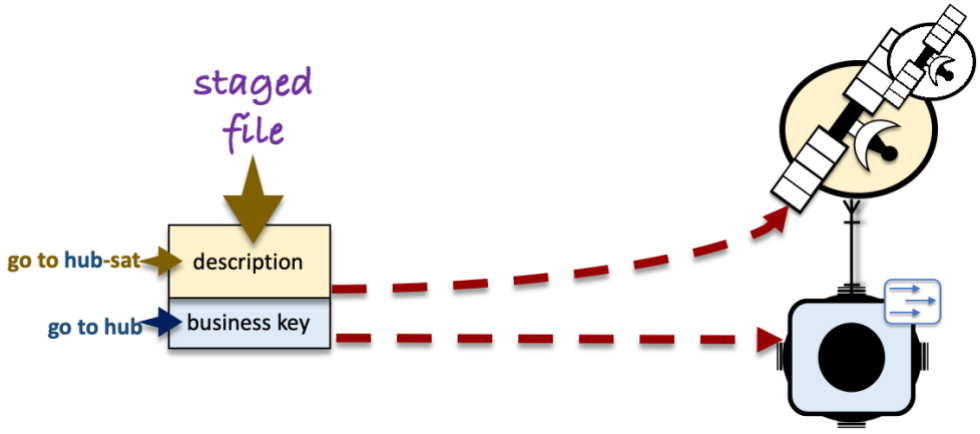


- If recon stream has data, then do:
  - Check ***hash key + load date*** duplicates of the ***current*** data
  - Reconcile staged ***link and/or hub hash keys*** are indeed in the ***parent*** table
  - Reconcile staged ***record hash*** is indeed in the satellite table (based on current view)
  - Count ***new records*** in stream
  - Count ***total records*** in satellite table after load
- Check if satellite ***hash key*** is in the parent ***hub or link*** table.

Now that we have established the standard loading and testing patterns let's see how we can combine them for some common data vault ***modelling*** scenarios.

Modelling scenarios

Business key and descriptive attributes



This will load to a hub table and a satellite table.

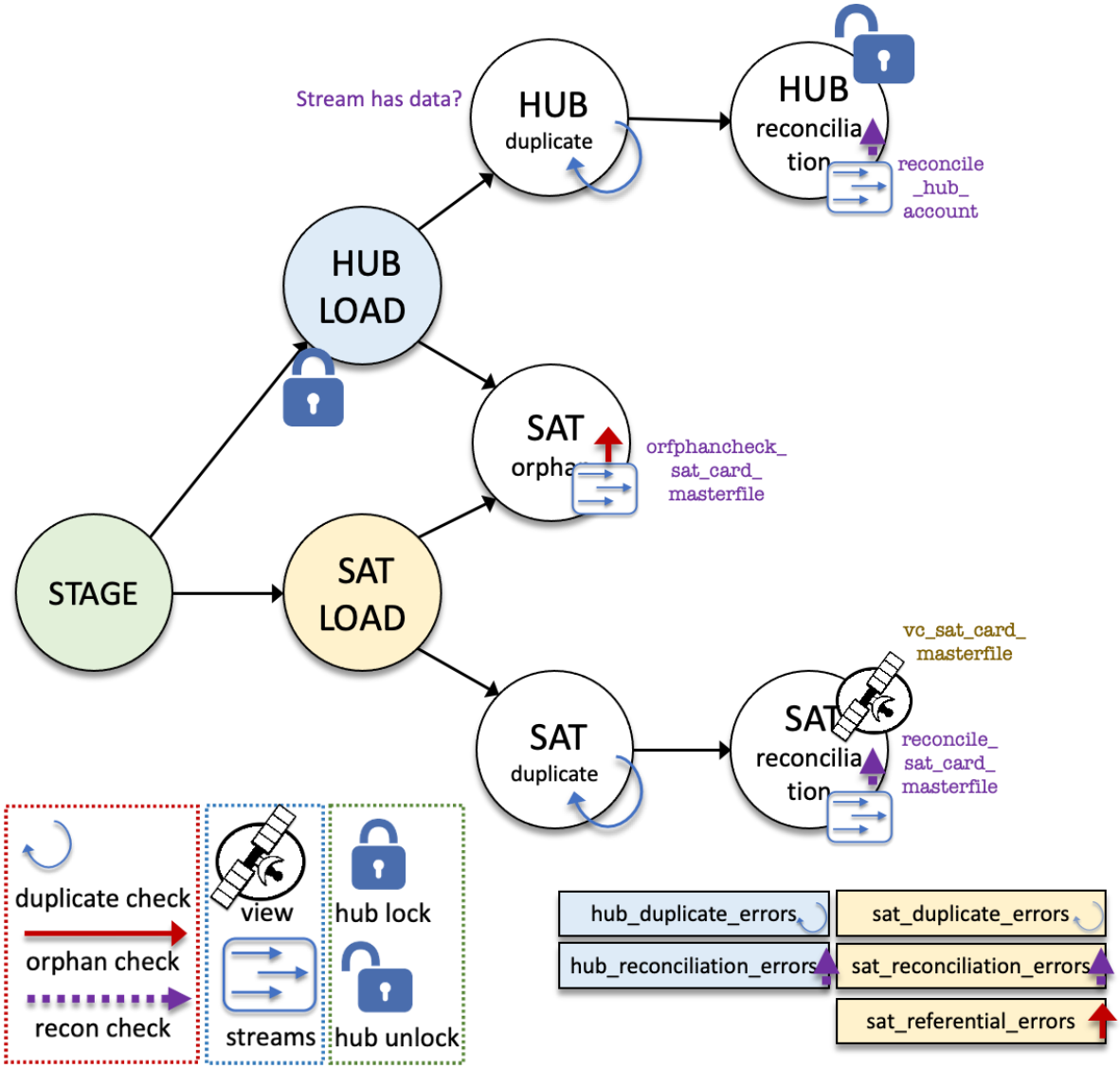


Figure 0-1Hub and Sat DAG

After the content is staged (data vault metadata tags added) independent hub and sat loaders take their respective content and load them to the target entities.

- Crucially a hub table lock is established (transaction / pool / semaphore) to populate the hub table, test for duplicates (only runs if the hub recon stream has data) and checks that all configured business and surrogate hash keys have been loaded. *At the same time…*
- Satellite test for duplicates (only runs if the satellite recon stream has data) and checks that the staged hub attributes exist in the target satellite table
- Satellite orphan checks only execute after the hub table has loaded and only if the satellite orphan stream has data (no need to check, an empty stream will run against nothing!)

Why is the hub table lock crucial? Multiple workstreams may be attempting to load the **same** hub table at the **same** time! See: bit.ly/3xlFK0s
If two or more workstreams attempt to load the ***same*** hub table at the ***same*** time ***without*** table locking and have the ***same*** business keys the competing processes ***cannot*** guarantee that the hub table will remain a unique list of business object keys.

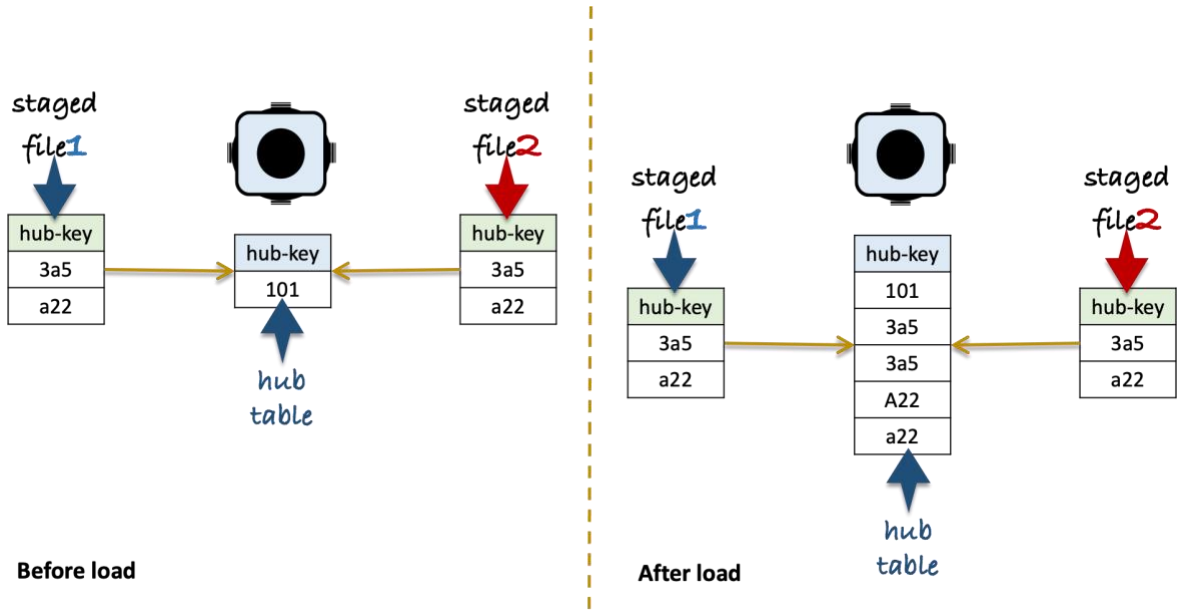


Figure 0-2 Multi-thread load allows for hub table duplicates, anti-pattern – fails hub duplicate test

Restricting the hub loads to single threads guarantees only one hub loader can load keys to the same entity at the same time. Of course, if different source systems will never have a business key clash (same (or similar format) business key representing two or more different business keys) then no such locking is needed, but then consider why you would have a "switch" architecture? Having a switch architecture further complicates the implementation unnecessarily especially if you consider that the hub table is so small and the cost of executing semaphore / transaction / pool locks is minimal. Having a ***single implementation pattern*** that handles both scenarios reduces the number of patterns needed and avoids a switch architecture.
Why no locking for satellite tables? They are source system bound... no need to lock those! Loading and checking is against the ***current active records*** for a satellite table, this is an important point because business objects can in fact ***revert*** to a ***previous state***. If the satellite loads and tests are based on hash-key and record hashdiff alone you will miss when reversion happens!

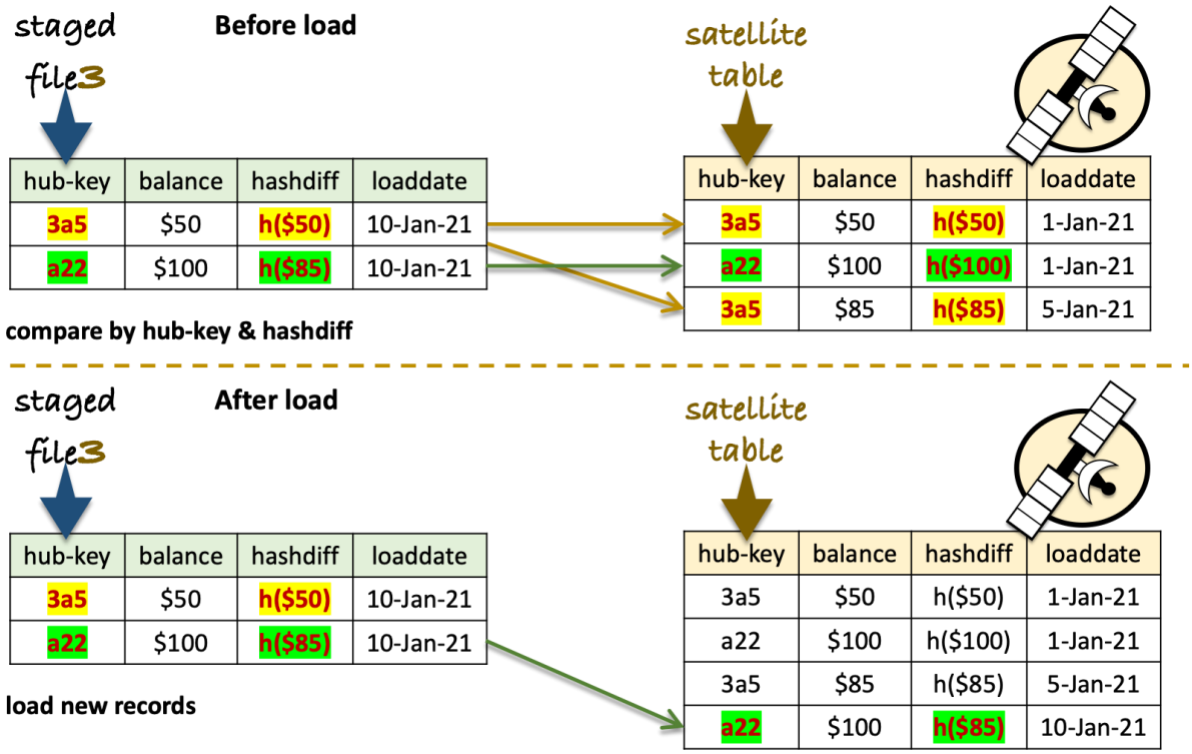


Figure 0-3 Comparison of key + hashdiff on whole of satellite, anti-pattern -- fails reconciliation test

Key '3a5' reverted to $50, but a comparison on all hashdiff for this key reveals it was this state before and thus does not load the new state, <u>of course this is incorrect!</u>

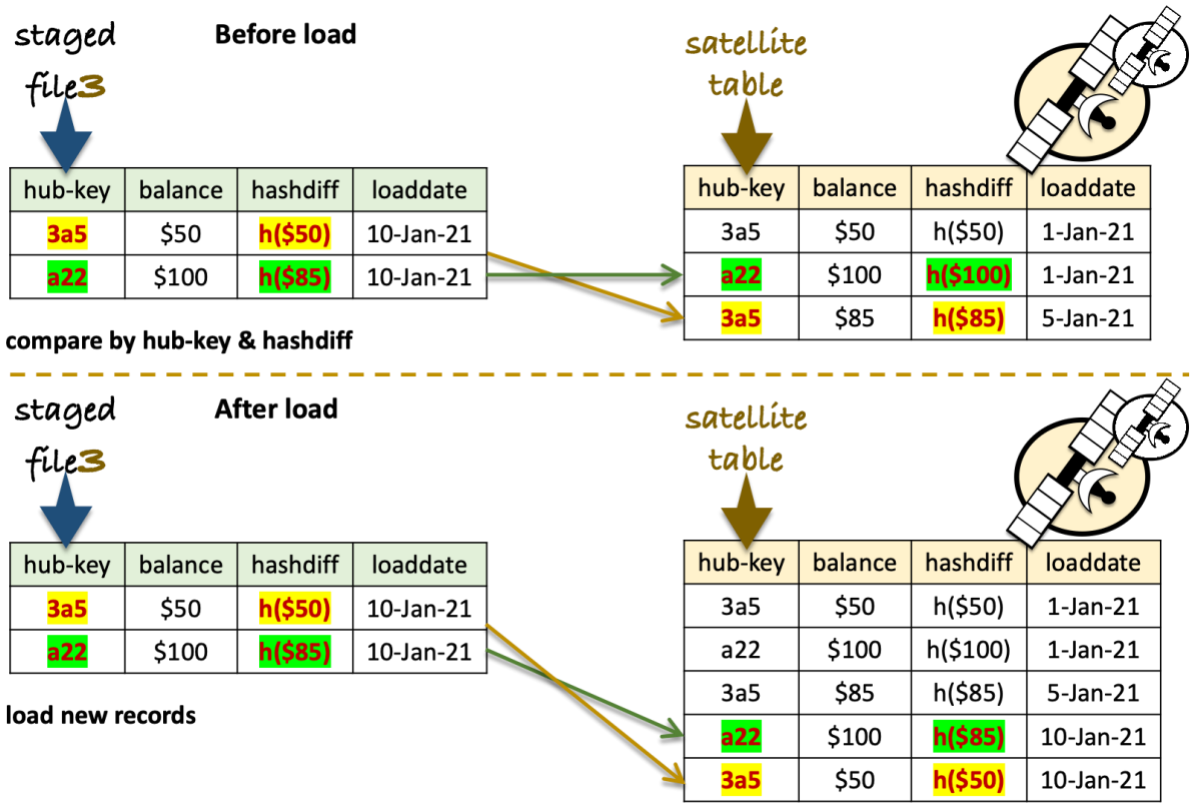
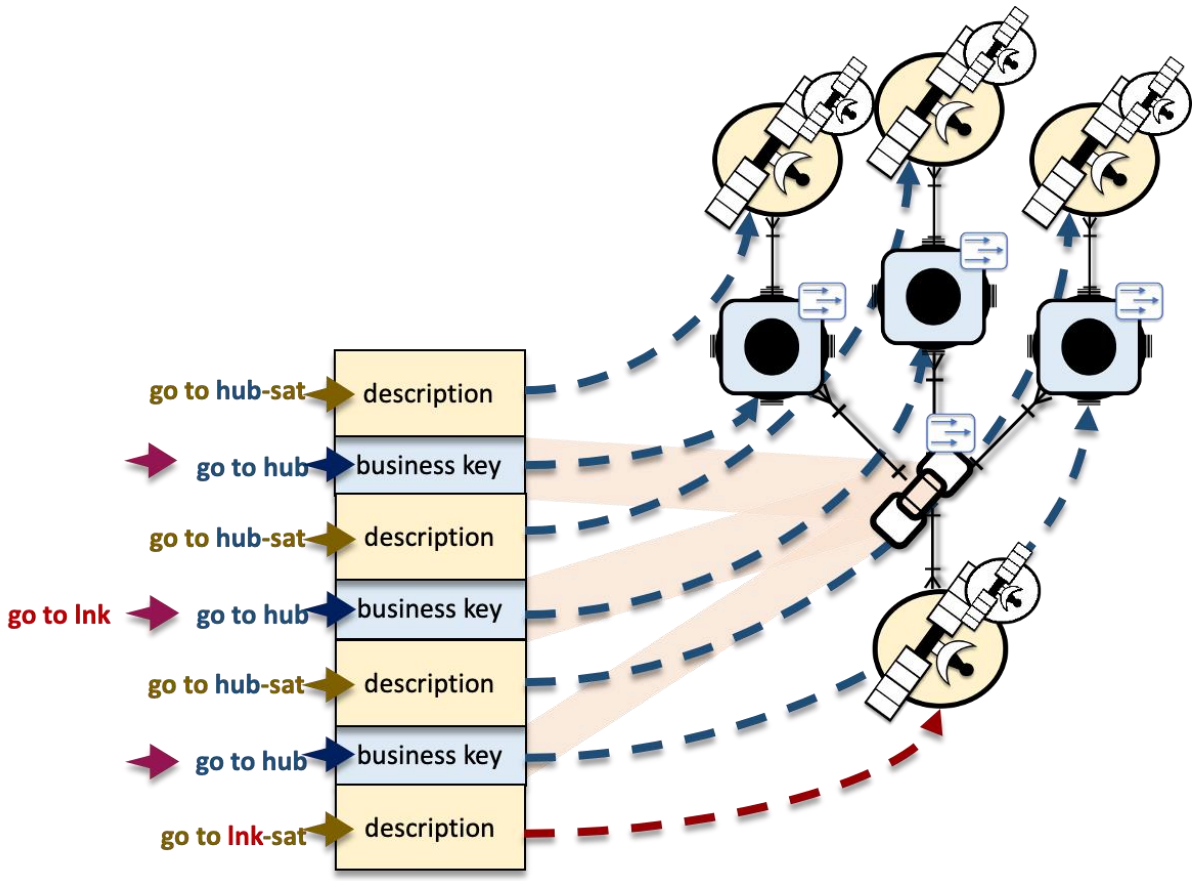
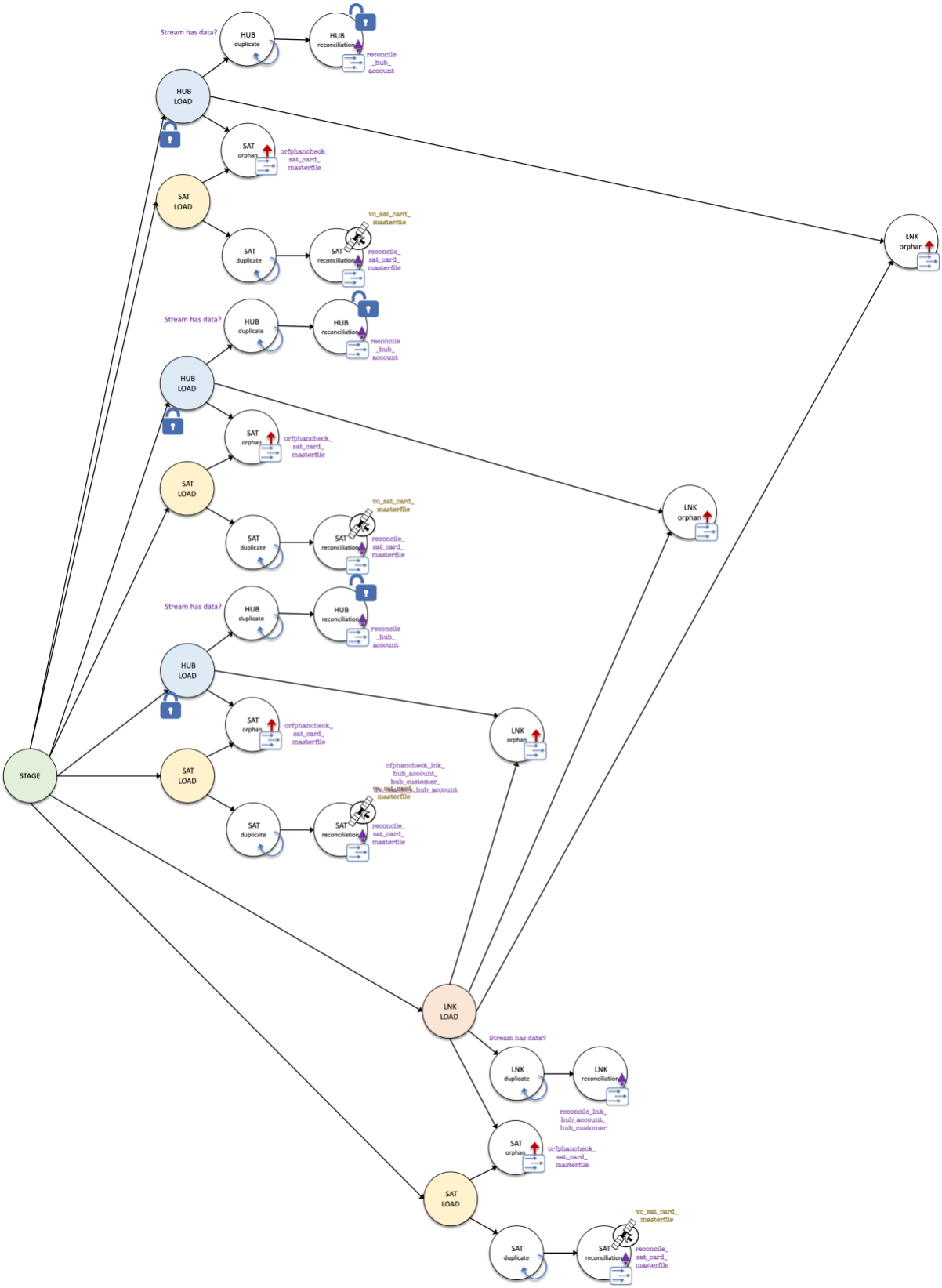


Figure 0-4 Comparing against the **current active record** *yields the correct result! Makes this load idempotent!*

A unit of work (relationship) and descriptive attributes



This scenario will load a link table, three separate hubs tables, three hub-satellite tables and a link-satellite table.



After the content is staged (data vault metadata tags added) independent hub, link and sat loaders take their respective content and load them to the target entities.

- Each hub table lock is established (transaction / pool) to populate the hub table, test for duplicates and checks that all configured business and surrogate hash keys have been loaded. *At the same time…*

- Each satellite test checks for duplicates and checks that the staged hub attributes (for each hub-satellite) exist in their respective target hub satellite tables and the staged link attributes exists in the link satellite table. *At the same time…*
- Link table test for duplicates (only runs if the link recon stream has data) and checks that all configured surrogate hash keys have been loaded.
- Satellite orphan checks only execute after their respective hub or link tables have been loaded and only if the satellite orphan streams have data

Hub and link satellite table content is coming from the same staged content is split, what this means is that in each split there may in fact now contain ***duplicates***. It is important to ***condense*** this content before loading them to their respective satellite tables which can be achieved in the loaders themselves!

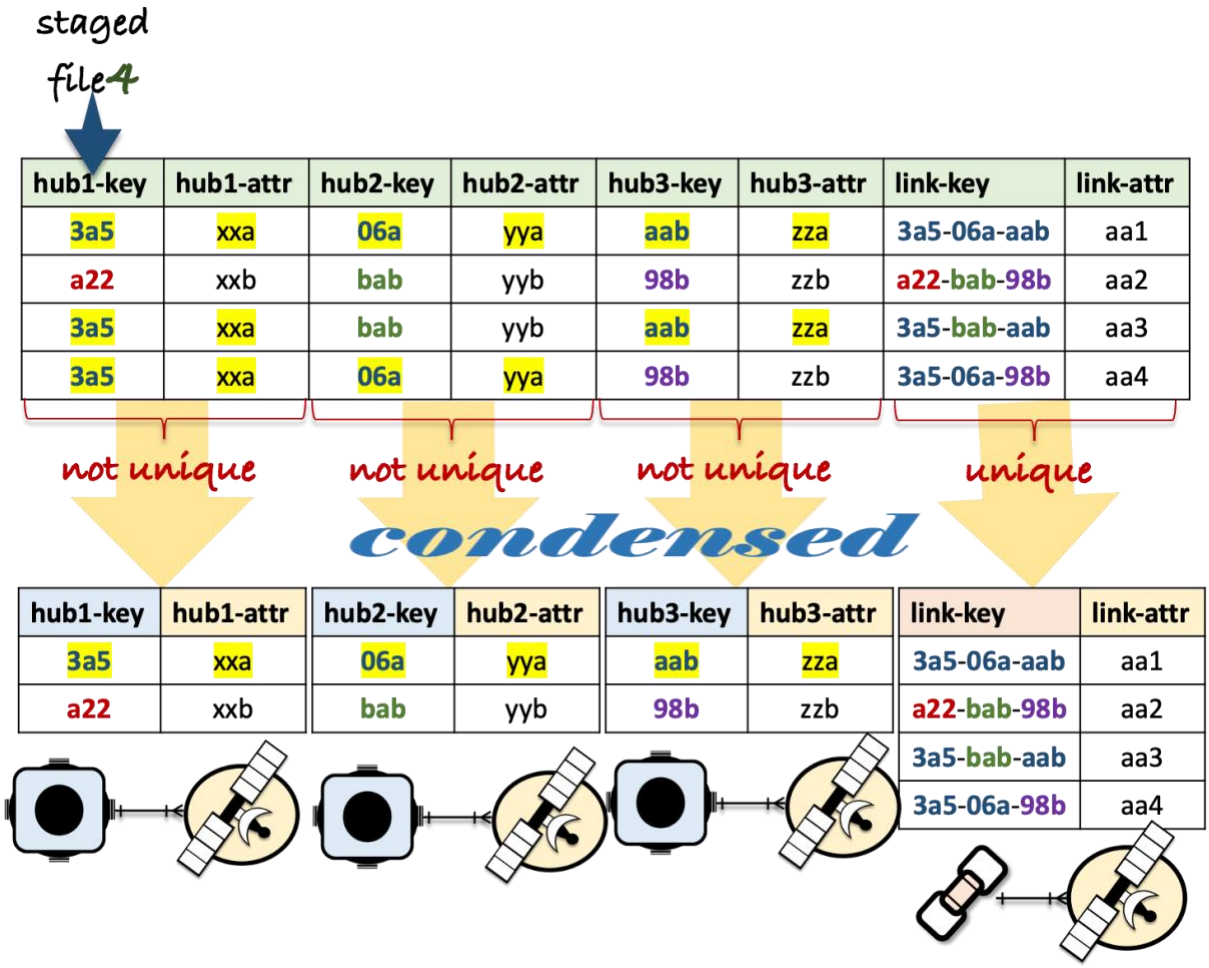
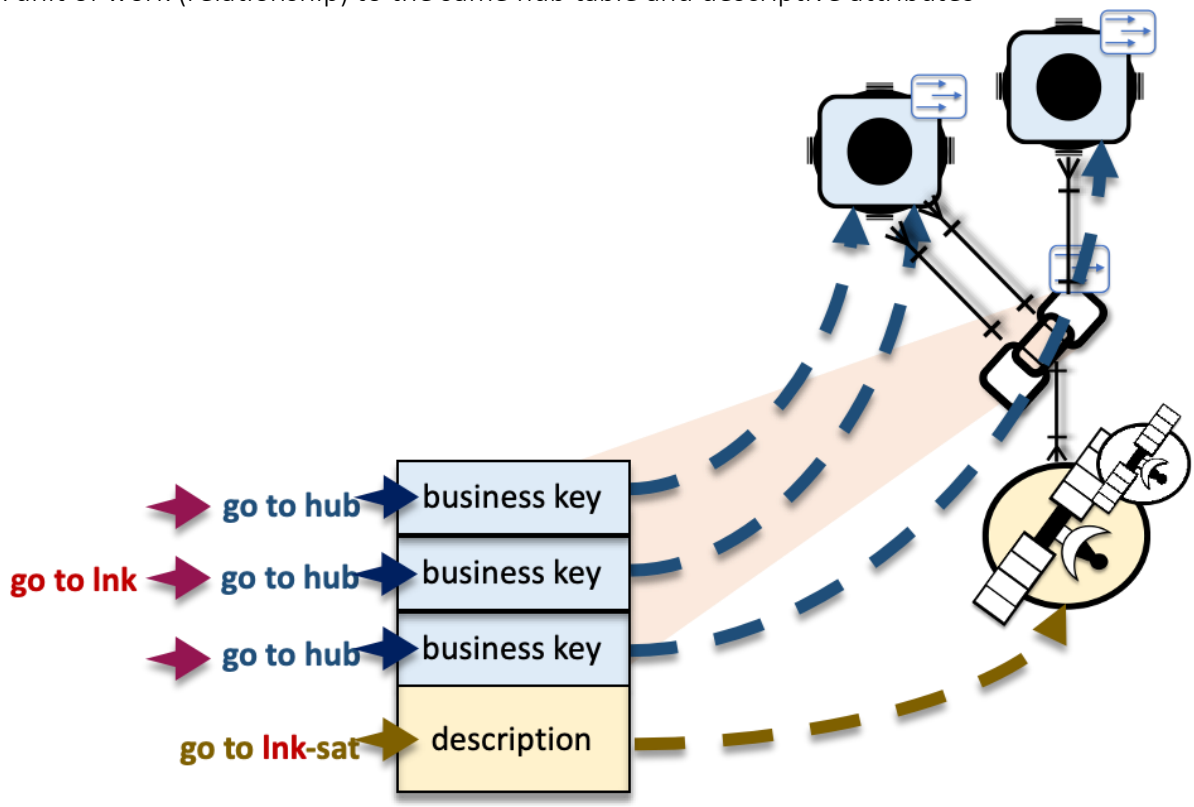
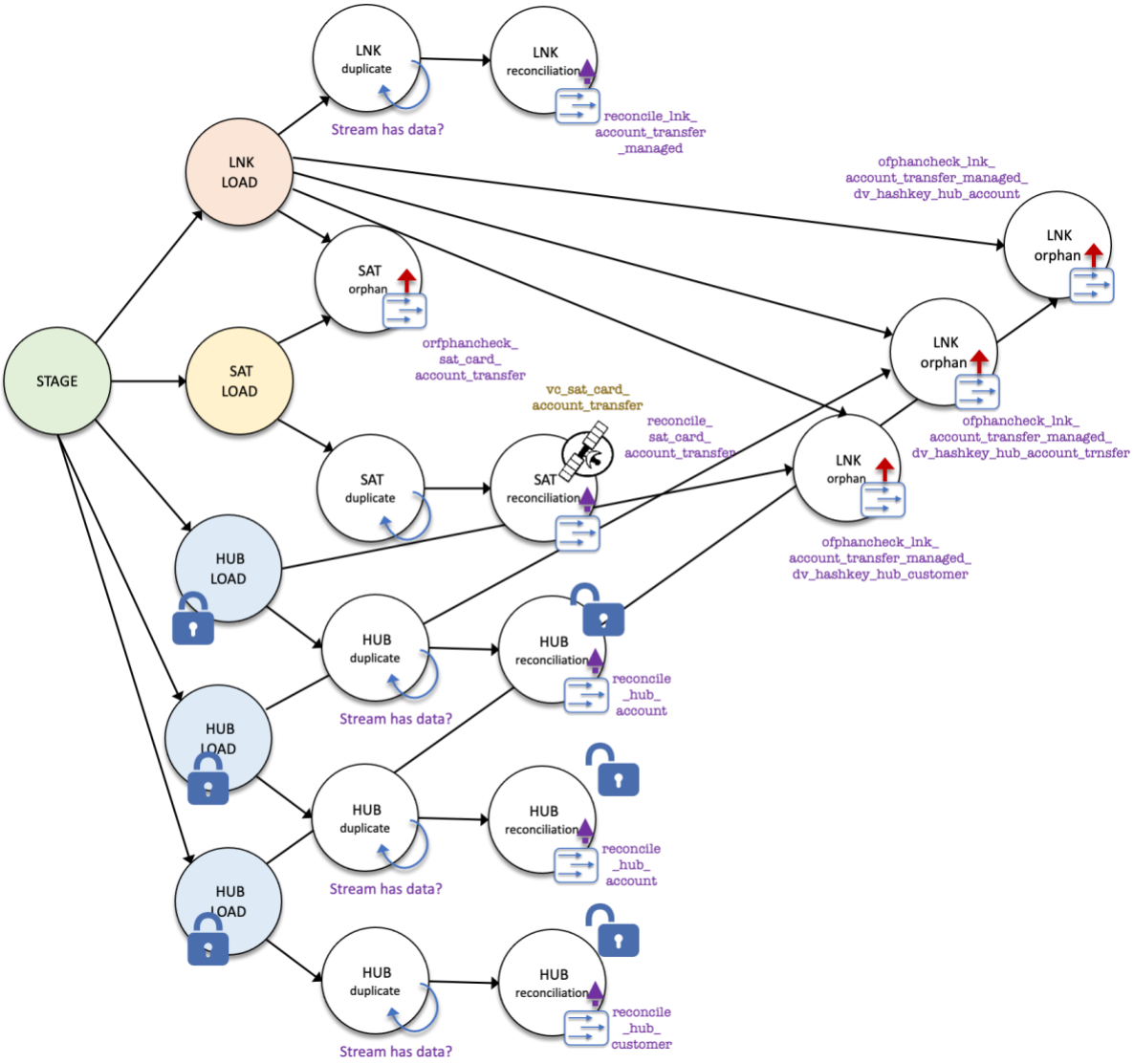


Figure 0-5 Modelled data condensed, eliminates duplicates, ensures uniqueness, the case for satellite splitting, see: bit.ly/3xnbcLH

A unit of work (relationship) to the same hub table and descriptive attributes



This scenario will load a link table, two separate hubs tables (one of them twice) and a link-satellite table.



After the content is staged (data vault metadata tags added) independent hub, link and satellite loaders take their respective content and load them to the target entities.

- Each hub table lock is established to populate the hub table, test for duplicates and checks that all configured business and surrogate hash keys have been loaded. *At the same time…*
- Satellite test checks for duplicates and checks that the staged link attributes exist in the link satellite table. *At the same time…*
- Link table test for duplicates checks that all configured surrogate hash keys have been loaded.
- Satellite orphan checks only execute after its respective link table has been loaded and only if the satellite orphan streams have data

A naïve strategy to loading content from a common staged table is to attempt to use ***multi-table insert*** in this scenario (a single table insert SQL statement). Two different sets of columns for two related business entities (card account and card account transferred) will attempt to load the same hub table at the same time and the same pattern we saw without target table locking occurs, duplicates *can* be loaded.

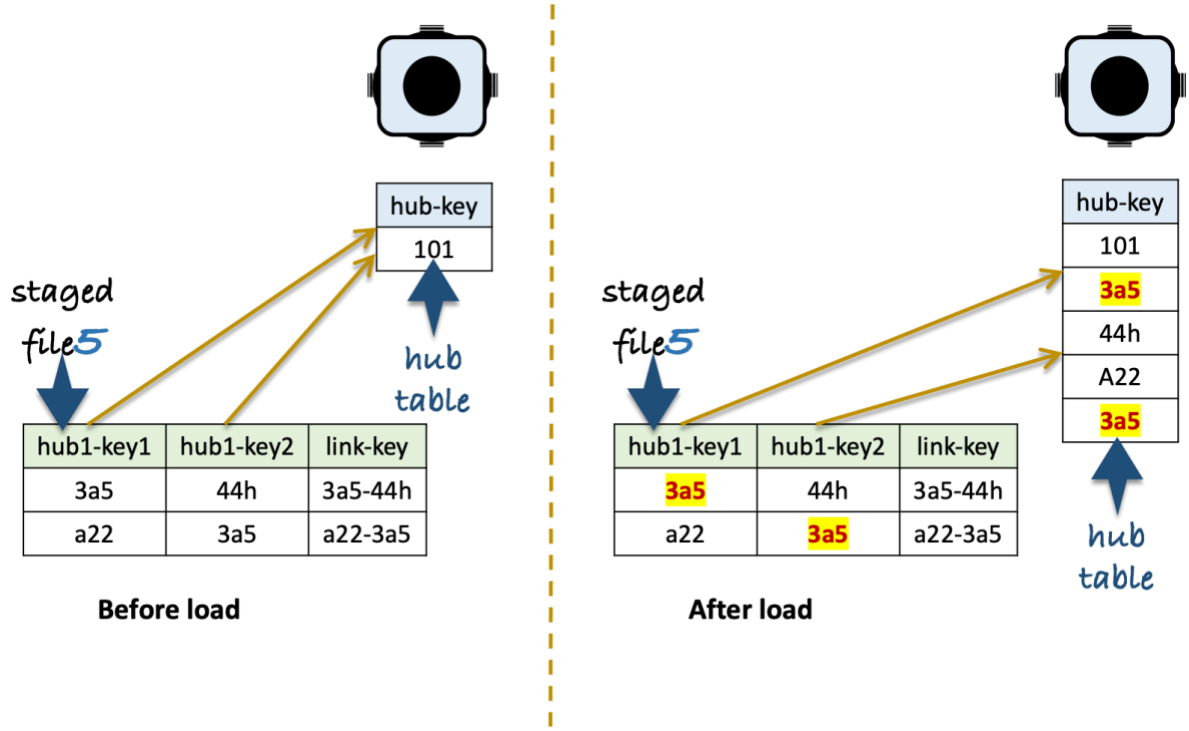


Figure 0-6 Multi-table insert allows for hub table duplicates, anti-pattern – fails hub duplicate test

Loading to a common hub table at the same time means that each hub loader portion of the same multi-table insert statement determines that their content is new, alternatively one loader must execute at a time. *Same loading pattern we have seen before*
The dreaded "switch" architecture rears its head again! Do you use a multi-table insert for everything else but this modelling scenario? No. A ***single strategy*** for loading data vault must exist or you risk introducing ***technical debt*** into your architecture (complicated loading patterns). This ensures the number of patterns is ***minimal***, minimal means ***less prone to failure***, ***easier to manage*** and the patterns are ***easily repeatable***. Each vertex in a DAG should be ***single purpose*** and pattern based, as we saw in the above example patterns a hub and satellite should have their own ***independent*** hub and satellite loader respectively and therefore function ***autonomously***. If it were based on a multi-table insert then if ***any portion fails*** the ***whole load fails***, this is ***not*** an agile approach to loading a data vault!

Know your data

If you use streams on top of staged / landed content care must be taken that the content is properly loaded to their target entities. Hubs and links are unique lists, and the same code *pattern* can apply to both loads, satellite tables however must be modelled to the expected grain. If multiple states of a parent entity (hub or link) are pulled from the stream to load to a target satellite, *then is that the expected grain?*
Let's see what happens by using the record reversion example.

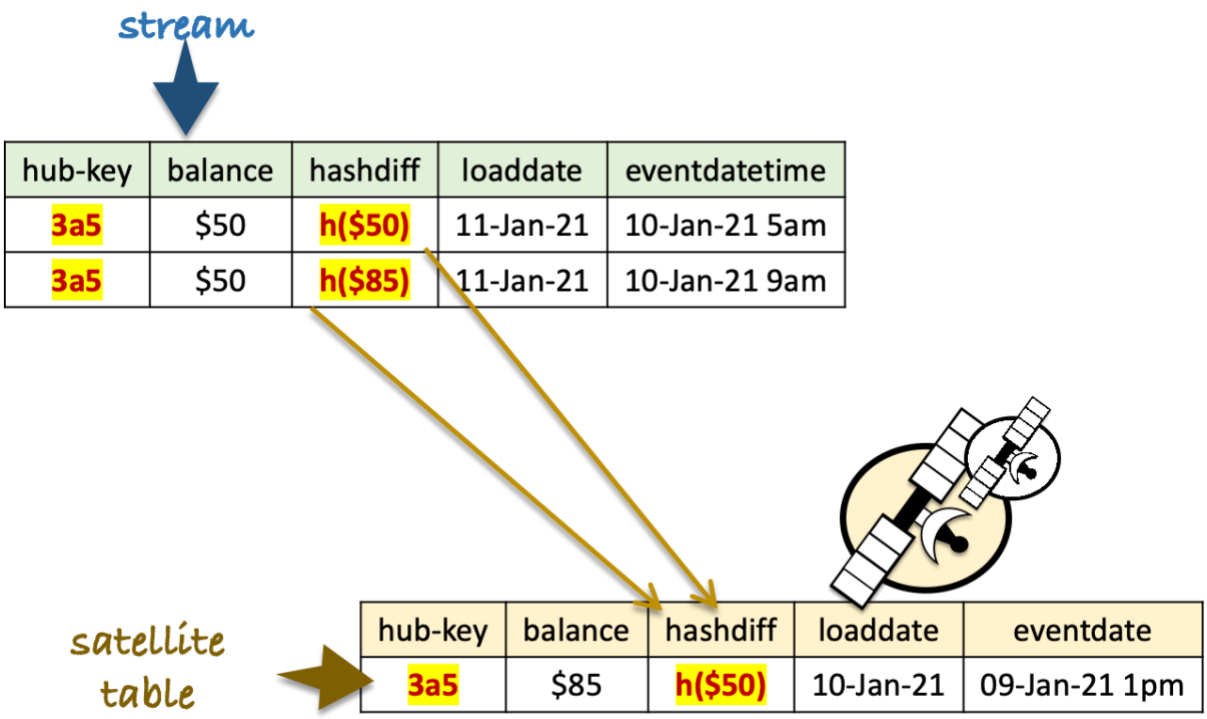
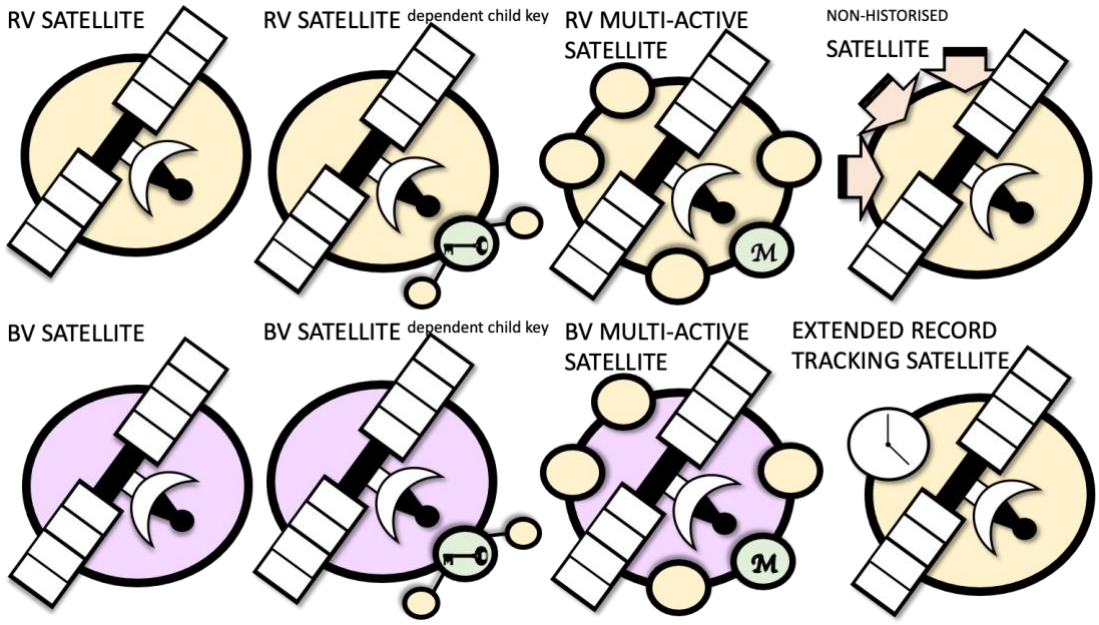


Figure 0-7 Workflow error? Or expected outcome?

Therefore, the load from a stream must form a pipeline, as data is landed, the stream content must be pulled into the next stage and loaded to the target satellite table(s) ***before*** new content is added to the stream. Inserting into a stream more than once before consuming the stream could compromise what gets loaded into the target satellite table(s). Do take care that in your modelling scenarios that you have however considered a multi-record state load into a satellite table, and there are generally two kinds in data vault. Do not model these satellite table patterns by default, they should only be modelled *by design*.

- ***Satellite with a dependent-child key*** (can be configured with an intra-day key).
- ***Multi-active satellite*** – an active set of records for a parent entity.

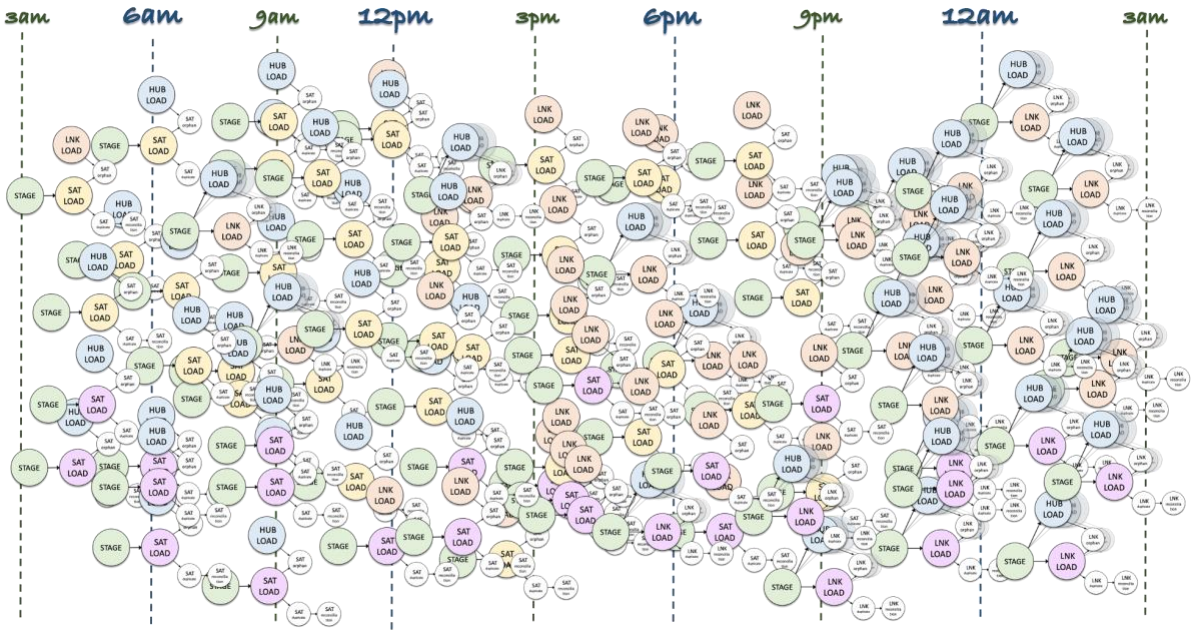
The other such pattern is if the content expected is always new (by definition), i.e., modelled as a CDC Satellite (or non-historised satellite / link) because it is what you're expecting from the source.



In closing

Except for non-historised structures, all variations of satellite table loading still follow a similar pattern, whether it is loading ***raw or business vault***. The additional pattern that you *could* consider adding to your data pipeline are query assistance tables, ala ***point-in-time*** (PIT) and ***bridge*** tables (see: bit.ly/3dBxOQK). Views above these structures are built once as information marts and thus inherit data as it is loaded.

Thus, if the data vault stage, loading and testing happen throughout the day the enterprise data vault model is in constant state of ***eventual consistency*** (see: bit.ly/3htcwpw).



What's more the existing satellite loading standards can be extended with extended record tracking satellites that provide data-driven timeline correction for the loads within the same DAG (see: bit.ly/3dlVDJn). Thus, data vault can ***self-heal***, this should not be the default fallback mind you, ideally out-of-sequence data arrival should not occur at all!

For more resources click through to my medium here: https://patrickcuba.medium.com/
Get a copy of "the Data Vault Guru" here amzn.to/3d7LsJV, amzn.to/3nsqTfR, amzn.to/30IxOYF

And other useful resources and points to consider, see

- Understand database isolation levels, bit.ly/3hCjitd
- Practical Guide to SQL Transaction Isolation, bit.ly/2TBcBj5
- Why write skew can happen in Repeatable reads? bit.ly/3hxe1Dc
- Linearizability, serializability, transaction isolation and consistency models bit.ly/3yjoatU

The views expressed in this article are that of my own, you should test implementation performance before committing to this implementation. The author provides no guarantees in this regard.