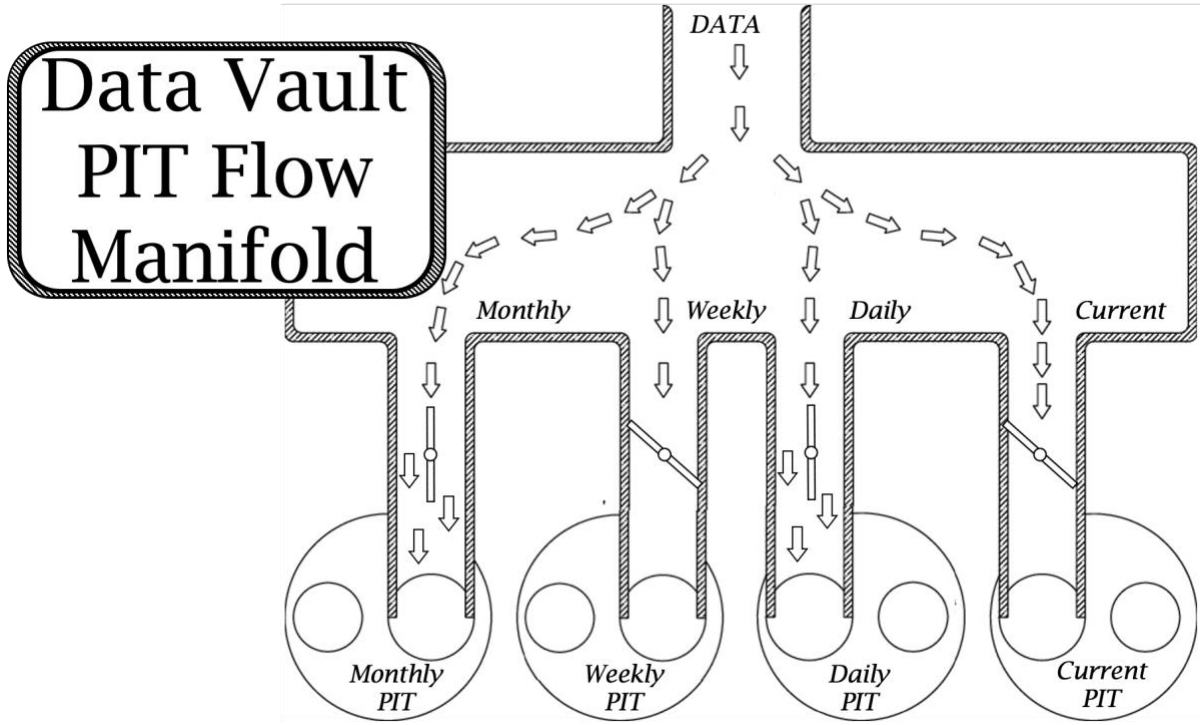
Data Vault PIT Flow Manifold



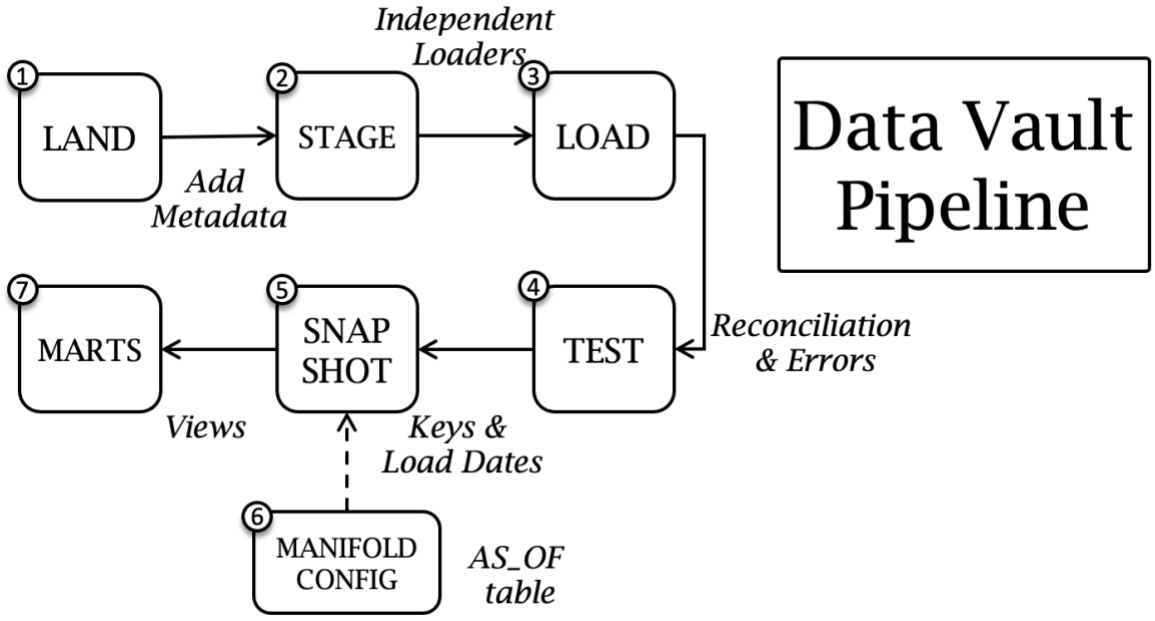
Streams and Tasks (see: bit.ly/3fqV9oZ) form a part of Snowflake's Data Pipelines and from a Data Vault 2.0 perspective they're a perfect fit and a templated approach to getting data staged, loaded, and tested through repeatable patterns. Here we will use another piece of Snowflake ingenuity to build a **PIT Table Manifold**.

This article builds on the previous three Snowflake Data Vault articles highlighting how the two are a great fit!

- Data Vault Test Automation, bit.ly/3dUHPIS
- Why EQUIJOINS Matter! bit.ly/3dBxOQK
- Data Vault 2.0 on Snowflake… to hash or not to hash… that is the question, bit.ly/3rH7gS5

Snowflake includes the **Multi-Table Insert** (see: bit.ly/3h97Ypz) that allows for a single source (can be a complex join) to feed multiple target tables at the same time. It may be tempting to load Data Vault hubs, links, and satellites from a single staged file, but we have shown in the automated testing article why this is **folly**! Instead, we *do want* to use it when loading PIT tables but with a *twist*… **a data driven** twist! And we will show how!

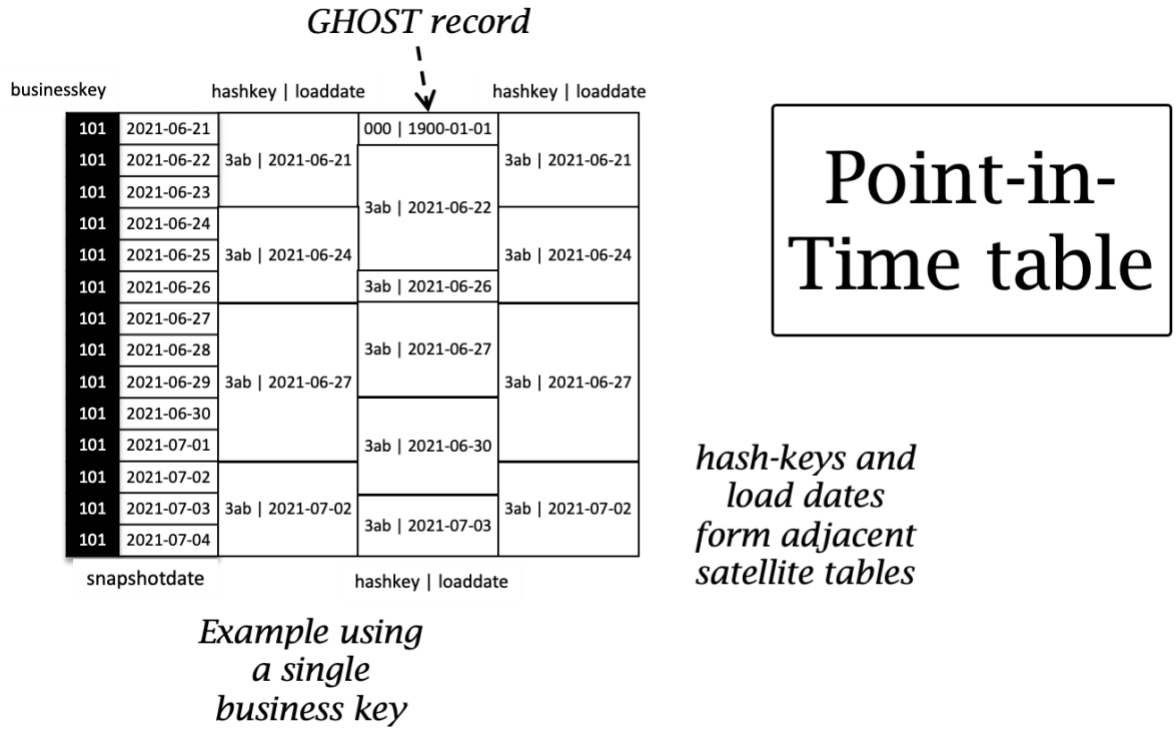
Data Vault Pipeline



Let's run through each stage of the pipeline

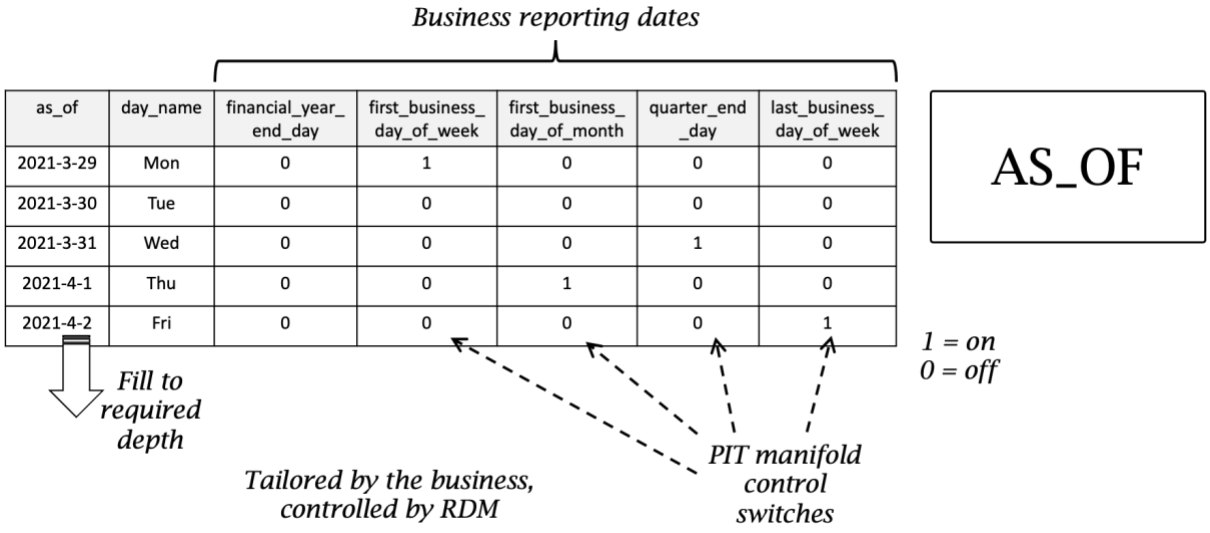
1. Data is landed either as an INSERT OVERWRITE or INSERT ONLY. Without dropping the target's contents then this is the first place we can use Snowflake's Streams (see: bit.ly/3lsdlm6) to process new records only downstream.
2. Landed content is staged with data vault metadata tags, some of these are:
    a. Surrogate hash keys – for joining related data vault tables
    b. Load date – the timestamp of when the data enters the data warehouse
    c. Applied date – the timestamp of the landed data
    d. Record source – where the data came from
    e. Record hashes – a single column representing a collection of descriptive attributes
3. Autonomous loads through
    a. Hub loaders – a template reused to load one or many hub tables
    b. Link loaders – a template reused to load zero or many link tables
    c. Sat loaders – a template reused to load zero or many satellite tables
4. Test Automation measuring the integrity of all loaded (and related) data vault artefacts from the staged content
5. Snapshot is taken of the current load dates and surrogate hash keys from the parent entity (hub or link) and adjacent satellite tables.
6. Use the AS_OF date table to control the PIT manifold to periodically populate target PIT tables at the configured frequency.
7. Information Marts that are defined once as views over a designated PIT table.

Point-in-Time table



A Point-in-Time (PIT) table is a **physical** collection of applicable surrogate keys **and** load dates for a snapshot period. It **must** be a **table** otherwise the potential benefits of **EQUIJOIN** are not realised. See bit.ly/3vjTXdg
It improves join performance and forms the base for information mart **views** and easily allows you to define marts to be built for specific ***logarithmic time windows*** by a snapshot date. The traditional approach to building a PIT table makes use of an adjacent date table to define the logarithmic period and the PIT windows itself.

Configure AS_OF



AS_OF table controls the PIT snapshot in a combination of two ways

- ***By Window:*** defining the start and end date of the AS_OF period, the window of snapshots to take. You can define a much larger than needed table but subset the window in execution.
- ***By Frequency:*** defining at what frequency snapshot keys are to be sent to a PIT target. This is tailored by the business and is a part of the report frequency and depth. Ideally this would not have any involvement by engineering teams, only to set this up. From there the business controls the 1 and 0 switches.

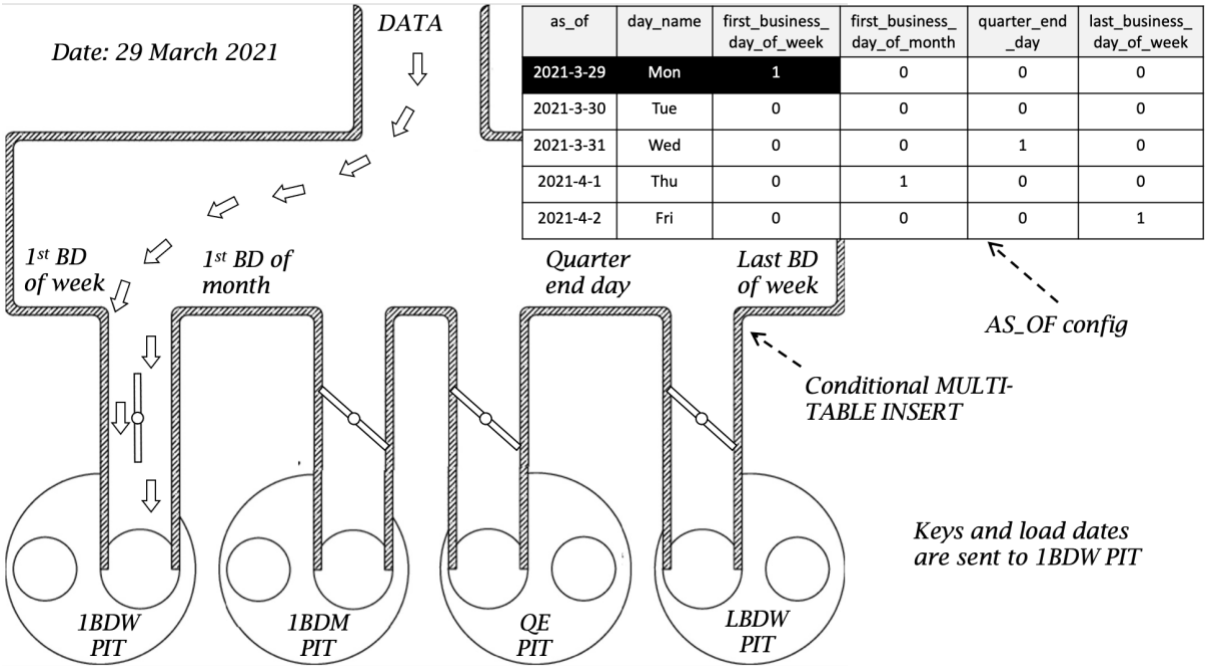
The PIT Flow Manifold

Assembling the components together needs one more artefact to make the pipeline work, Multi-Table Insert syntax that uses AS_OF to complete the PIT Flow Manifold!

```
insert all
when (first_business_day_of_week=1)
and (select count(1) from PIT_1BDW tgt where tgt.snapshotdate=src_snapshotdate) = 0
then into PIT_1BDW
(

…PIT Columns…

)
values(

…PIT Columns…

)
when (quarter_end_day=1)
and (select count(1) from PIT_QE tgt where tgt.snapshotdate=src_snapshotdate) = 0 then
into PIT_QE
(

…PIT Columns…

)
values(

…PIT Columns…

)
when (last_business_day_of_week=1)
and (select count(1) from PIT_LBDW tgt where tgt.snapshotdate=src_snapshotdate) = 0
then into PIT_LBDW
(

…PIT Columns…
```

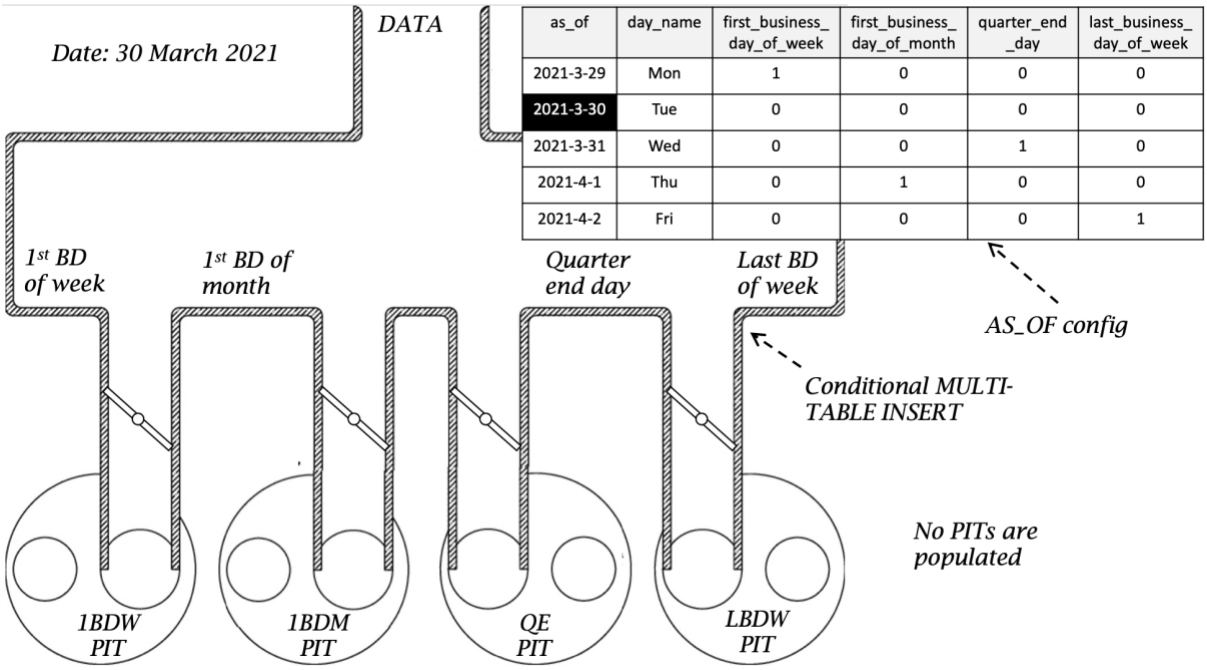
```
)
values(

…PIT Columns…

)
with as_of as (
  select *
  from rawvault.as_of_date aof
  where as_of=to_timestamp($my_loaddate)
)

…PIT Code…
;
```

The components together will do the following



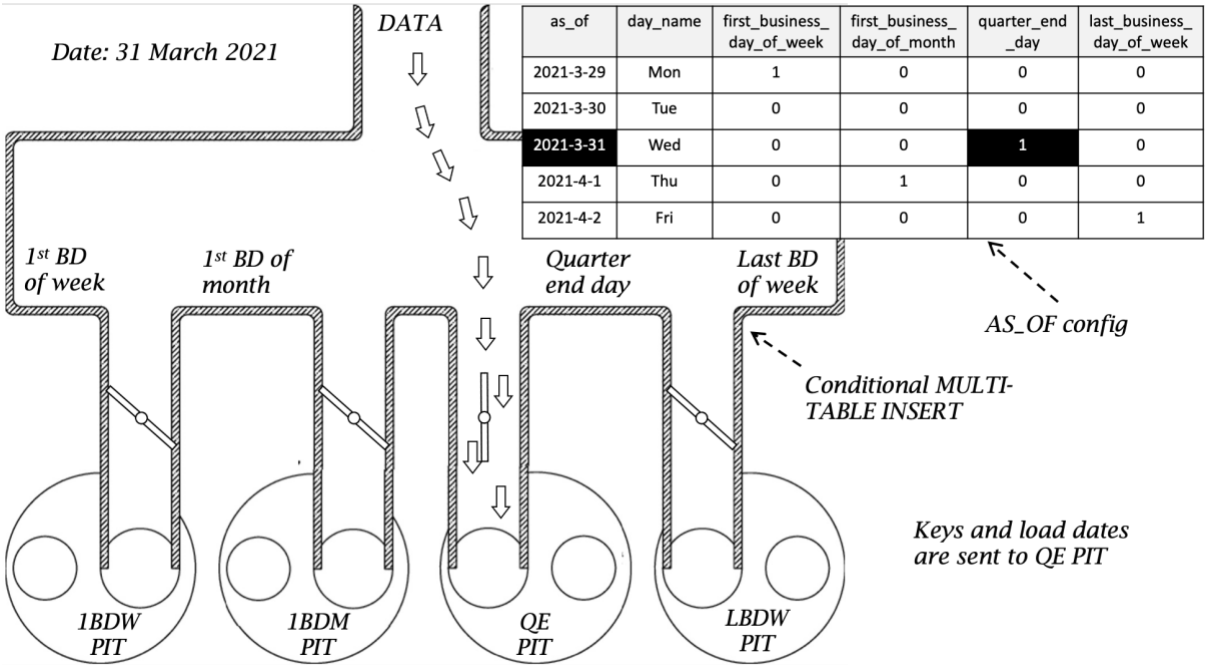
as_of	day_name	first_business_day_of_week	first_business_day_of_month	quarter_end_day	last_business_day_of_week
2021-3-29	Mon	1	0	0	0
2021-3-30	Tue	0	0	0	0
2021-3-31	Wed	0	0	1	0
2021-4-1	Thu	0	1	0	0
2021-4-2	Fri	0	0	0	1

No flow if the configuration is set as such.



as_of	day_name	first_business_day_of_week	first_business_day_of_month	quarter_end_day	last_business_day_of_week
2021-3-29	Mon	1	0	0	0
2021-3-30	Tue	0	0	0	0
2021-3-31	Wed	0	0	1	0
2021-4-1	Thu	0	1	0	0
2021-4-2	Fri	0	0	0	1

And the flow continues the next day



as_of	day_name	first_business_day_of_week	first_business_day_of_month	quarter_end_day	last_business_day_of_week
2021-3-29	Mon	1	0	0	0
2021-3-30	Tue	0	0	0	0
2021-3-31	Wed	0	0	1	0
2021-4-1	Thu	0	1	0	0
2021-4-2	Fri	0	0	0	1

Information Mart Views

Now that we have built the manifold, the last part of the data vault pipeline is the information mart views. These are created only **once** and as the data vault is updated underneath it all, the manifold pumps the applicable keys into PITs and the view of course are immediately up to date.

```
create view my_information_mart as
select stem.dv_hashkey_hub_account
      , stem.account_id
```

```
      , leaf1.purchase_count
      , leaf1.refund_count
      , leaf2.card_type
      , leaf2.card_status
      , leaf3.balance_category
      , leaf3.offer_id
      , leaf4.comprehensive_credit_score
from pit_1BDW stem
inner join sat_card_transaction_header leaf1
on stem.sat_card_transaction_header_dv_hashkey_hub_account = leaf1.dv_hashkey_hub_account
and leaf1.dv_loaddate = sat_card_transaction_header_dv_loaddate

inner join sat_card_masterfile leaf2
on stem.sat_card_masterfile_dv_hashkey_hub_account = leaf2.dv_hashkey_hub_account
and leaf2.dv_loaddate = sat_card_masterfile_dv_loaddate

inner join sat_card_balancecategories leaf3
on stem.sat_card_balancecategories_dv_hashkey_hub_account = leaf3.dv_hashkey_hub_account
and leaf3.dv_loaddate = sat_card_balancecategories_dv_loaddate

inner join sat_bv_account_card_summary leaf4
on stem.sat_bv_account_card_summary_dv_hashkey_hub_account = leaf4.dv_hashkey_hub_account
and leaf4.dv_loaddate = sat_bv_account_card_summary_dv_loaddate
```

Conclusions

Data Vault provides the proven data architecture templates that your enterprise data platform can be built on. Like agile sprint iterations and DevOps engineering these templates are ***repeatable***, ***reduce*** the probabilities of errors and are ***scalable***.
PIT flow manifold introduced here uses these templates together to make the management of information marts on a data vault completely ***data-driven!*** And on a platform designed to scale why not adopt a data modelling methodology that is designed to scale as well!

#snowflakedb #datavault

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.