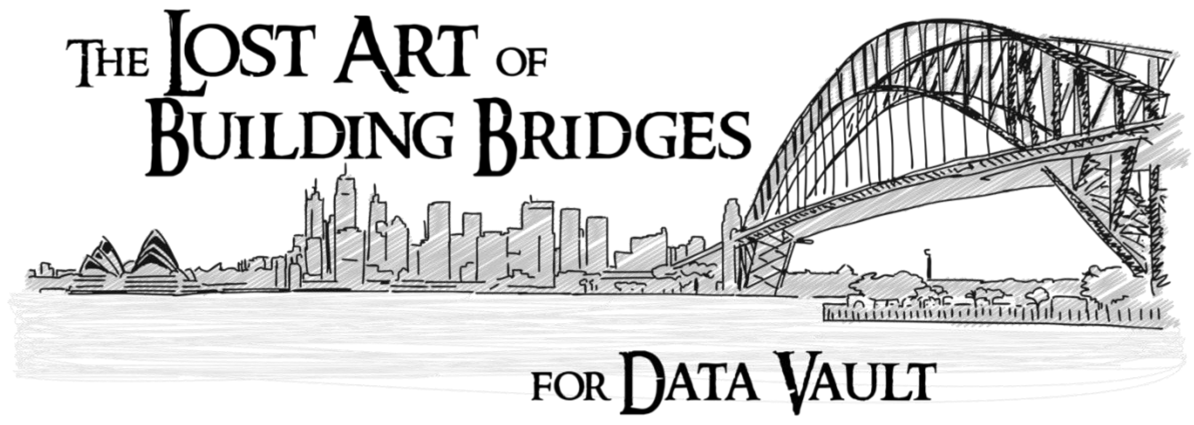
The Lost Art of Building Bridges



As a data vault grows with your business whilst by solving multiple business requirements it may be time to consider the need to build a bridge between the business objects to serve an information mart requirement. A **bridge table** in data vault is one of the two constructs designed to serve an **information mart** layer of your analytics platform by

- Using keys to take advantage of the platform's features to improve querying performance
- Taking the *complexity* of performing these joins repetitively away from the business users
- Shortening the distance between hub tables

The other query assistance construct is the point-in-time (**PIT**) table which we have discussed quite a bit before. Like the PIT table the bridge table is **disposable**, which is why it is *not* a data vault table, but rather a construct designed to leverage *both* raw and business vault to optimize query performance from data vault and to simplify SQL joins for the general information mart user. We also have the option to include *lightweight* derivations/aggregations/calculations in the bridge table, nothing too complex. Basically, the difference between including calculations in a bridge table versus using a business vault link table is threefold:

- Is the business rule outcome **idempotent**?
- Do we need to maintain an **audit history** of the business rule outcome?
- Can we **simplify** the relationship for querying and deal with business object relationships that *flip-flop*?

Traditionally, building bridge tables in **dimensional modelling** is done to simplify many-to-many relationships into one-to-many or one-to-one relationships. As such if a many-to-many relationship does exist between concepts, then a **step-bridge** may be used to consolidate multiple entities into a single entity (hierarchy) to correctly allocate and count metrics (facts) to that dimension in the relationship. Another technique involving bridges in dimensional modelling is the need for a **multi-valued bridge**, a single fact maps to multiple applicable dimensions. Lastly, a more recent type of bridge table called the "**Puppini" bridge** table was designed to dedupe the metrics around relationships when dealing with the classic **fan-trap** and **chasm-trap** data modelling problems experienced on 3rd normal form data models.

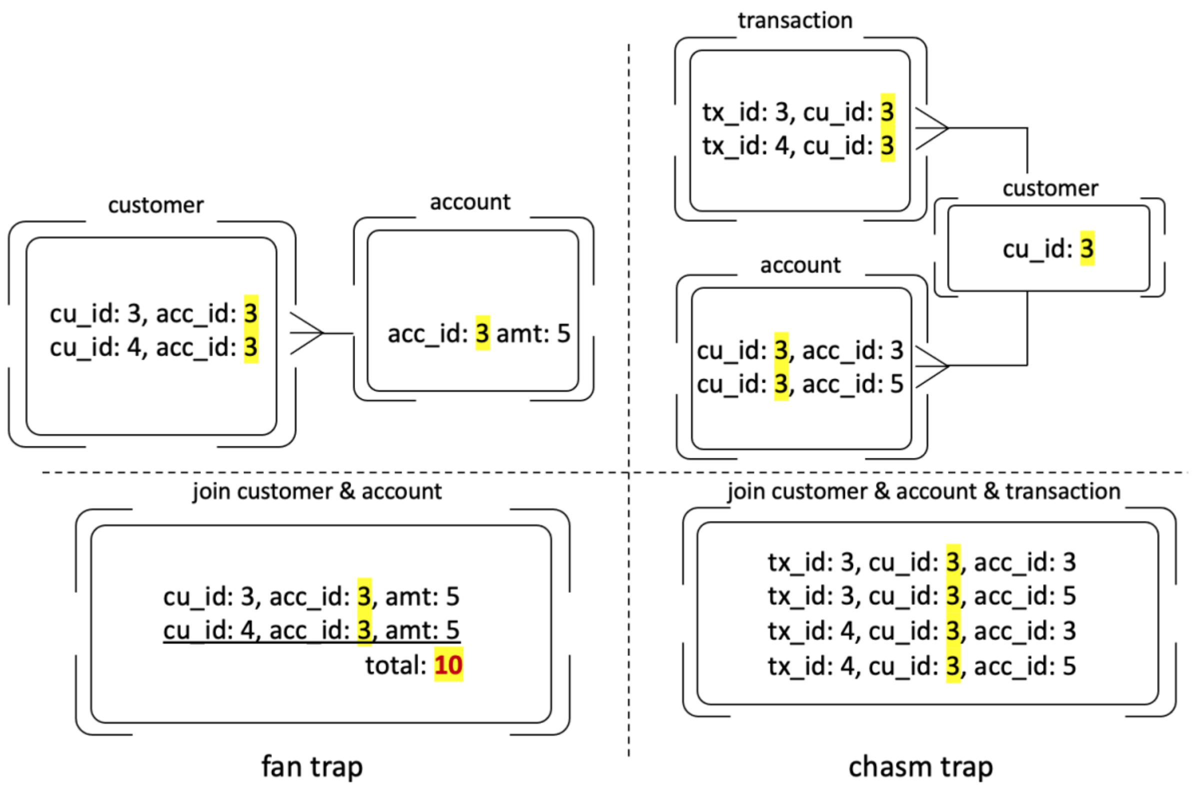


Figure 0-1 Fan-Trap, Chasm Trap

A fan trap can occur in an SQL join between tables that results in **double counting** metrics, in the above illustration a shared account is depicted, and the balance for the customer should be 5 and not 10.
A chasm trap occurs when a modelled relationship requires a join to an associative entity but results in a join-explosion, aka *unintentional* **cartesian product**.
Note these are simplistic examples to illustrate the point, not an actual model!

Let's turn our attention back to data vault bridge tables, shall we?

Shorten the distance

Point A to Point E,

- What is the **link-to-link** journey your query needs to traverse every time your reports are run?
- Is this journey repeated for multiple information mart requirements?
- Are there "stops" along the way?

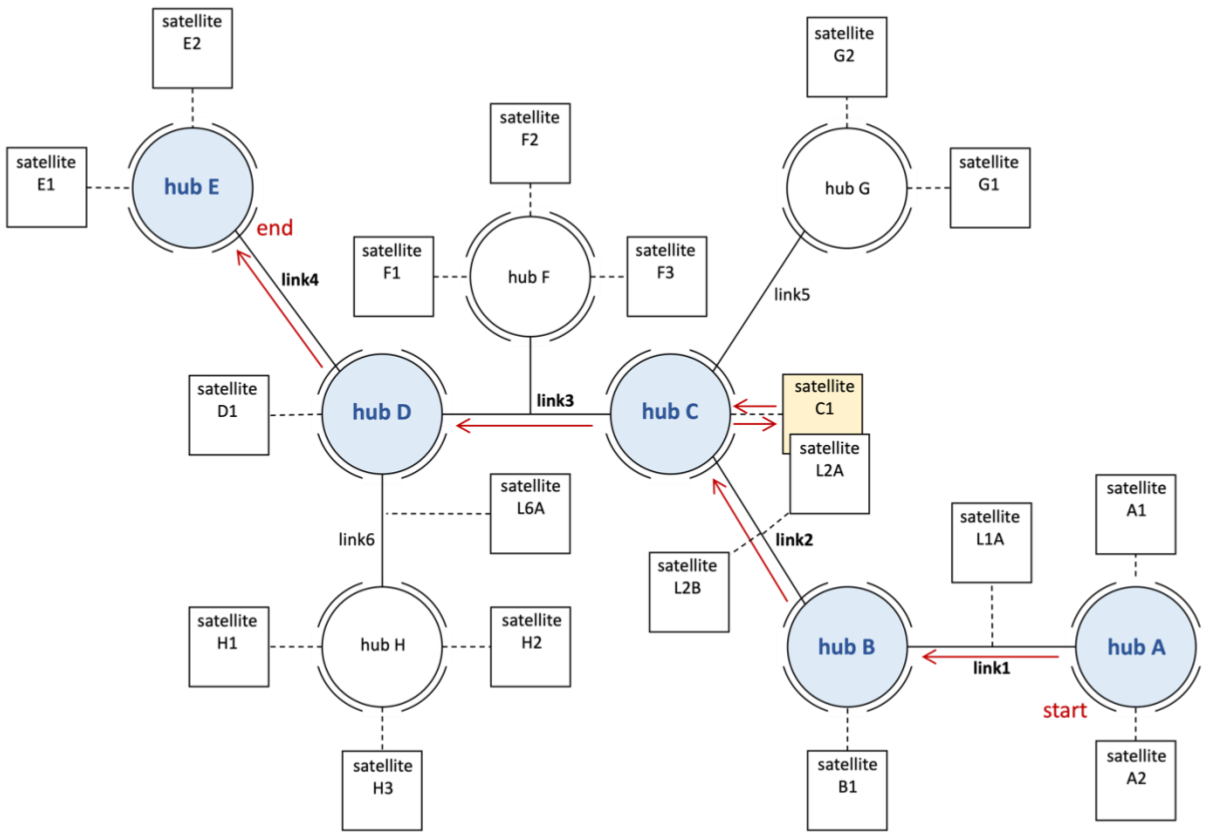


Figure 0-1 Shorten the distance for downstream reporting, greedy path

When multiple relationships have been processed and persisted as auditable raw and business vault link tables, a requirement may be to traverse these many link tables to return a **shortest path** for an information mart needed, like *folding space*. In your query construction if some hub table's business keys are not needed then you will not need to include those hub tables in the equi-join *journey*. Because each participating hub-hash-key in a link table comes from a hub table they will naturally inherit the business key collision code and multi-tenant id used to build those hub-hash keys in the first place. In other words, the equi-join guarantees you will only be joining business objects that are *really* related. A **link-to-link** join is *allowed* in this case.

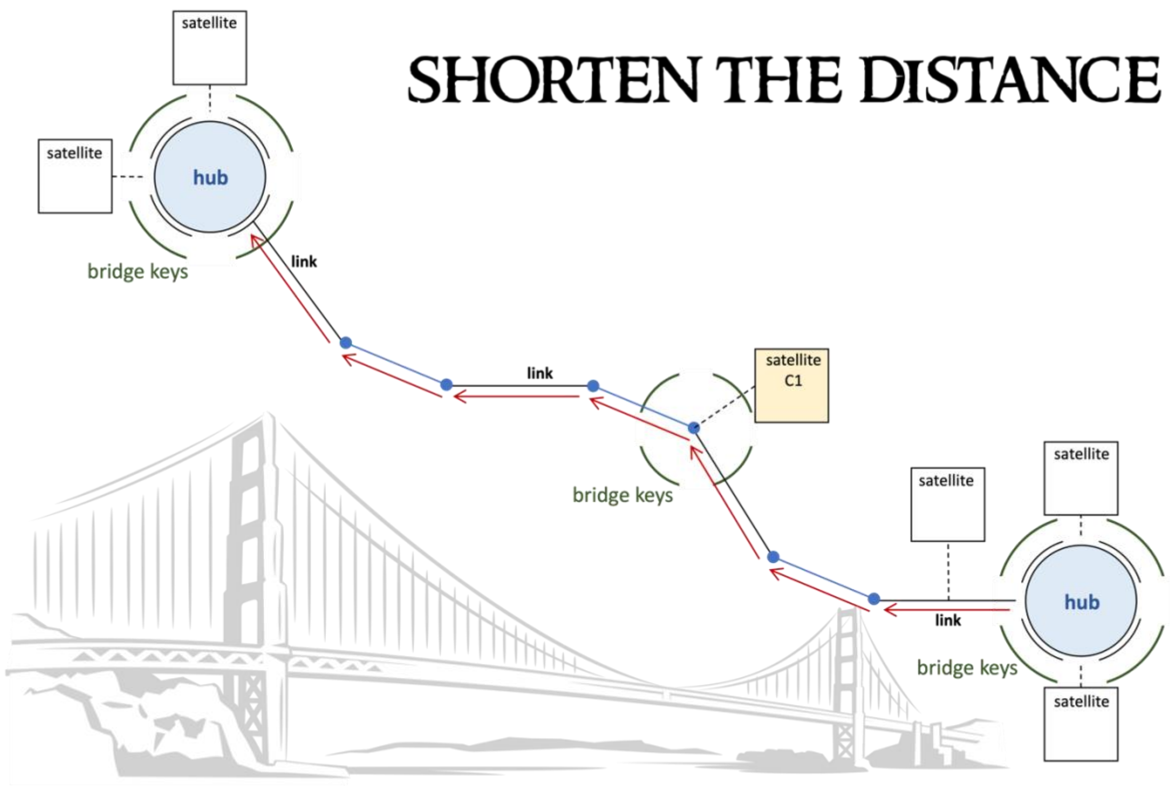


Figure 0-2 Link-to-Link

Information marts no longer need to traverse the data vault here, now they can rely on the bridge table to fetch descriptive content, which will drastically improve querying your data vault!

Persisted to BRDG	Bridge Build Only	**Persisted to BRDG**	Bridge Build Only	**Persisted to BRDG**
hub A	hub B	**hub C**	hub D	**hub E**
A101	B101	C101	D101	E101
A102	B101	C102	D102	E102

Hub C's hash key is persisted to fetch the content from satellite C

Example Bridge Build	**Example Information Mart Build**
create table **BRDG** as select hub_A.business-key_A , hub_A.hashkey_hub_A , hub_E.business-key_E , hub_E.hashkey_hub_E , lnk2.hashkey_hub_C from hub_A inner join lnk_hub_A_hub_B lnk1 on hub_A.hashkey_hub_A lnk1.hashkey_hub_A inner join lnk_hub_B_hub_C lnk2 on lnk1.hashkey_hub_B = lnk2.hashkey_hub_B inner join lnk_hub_C_hub_D_hub_F lnk3 on lnk2.hashkey_hub_C = lnk3.hashkey_hub_C inner join lnk_hub_D_hub_E lnk4 on lnk3.hashkey_hub_D = lnk4.hashkey_hub_D;	create view IM as select BRDG.business-key_A , BRDG.business-key_E , *satC1.<attributes>* from **BRDG** *[inner join satellite_C1 satC1* *on BRDG.hashkey_hub_C = satC1. hashkey_hub_C]*;

The piper is paid upfront! Now if we needed to include multiple satellite tables, they would likely need to have their timelines aligned if based off the same business object or unit-of-work. This is where you *could* need a PIT table too… but first.

Flip-flopping relationship

A **flip-flopping relationship** is not a faulty business rule, rather it may be the nature of the industry data model being modelled that such a business rule exists. To keep data vault flexible the tracking of relationship changes must be tracked by a **link-satellite** with a business date that is provided from the source. Only if such a business date is *not* retrievable from the source (unlikely) do we consider building an effectivity satellite and identifying the associated **driver key** of that relationship. The other situation is where an **effectivity satellite** *may* be considered is if the driver key(s) needed for reporting *differs* from what the driver key(s) is coming from source. An effectivity satellite and driver key deployment means you will have to manage this in data vault.

*Note, an effectivity satellite is the only data vault satellite with a **physicalised end-date column** and is still **INSERT-ONLY**.*

Now that you have this relationship tracked in a link-satellite (or effectivity) do you expose the business user/analyst to the need to query this relationship + effectivity, or do you *simplify* that query for your end-users? Yes, effectivity satellites are complex, and this is a candidate for simplifying data vault consumption via a bridge table.

Note, a business vault link remains auditable, a bridge table does not need to record history in this example. We may only be interested in the current relationship between business objects.

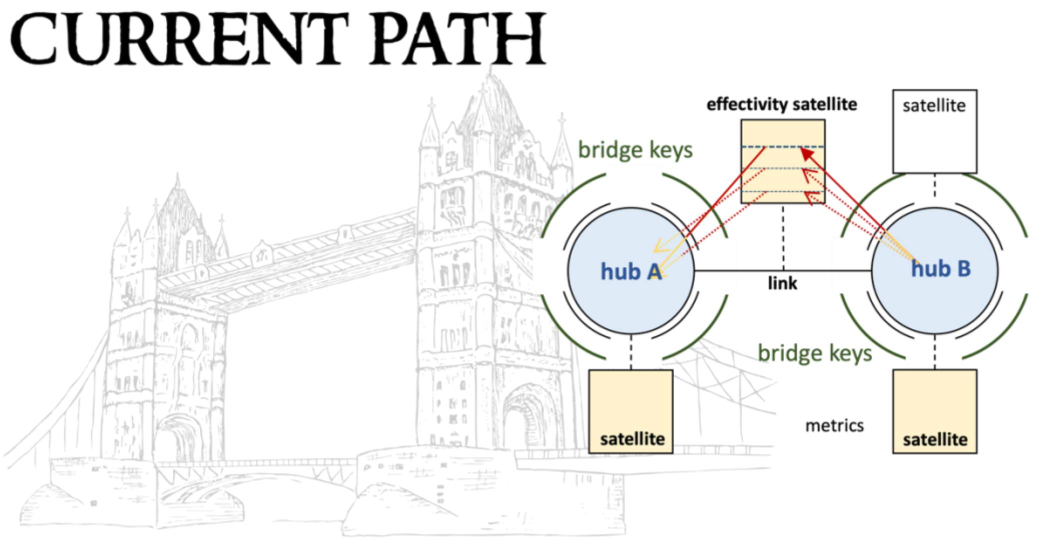


Figure 0-1 What is the current relationship?

Information marts no longer need to select the current path, now they can rely on the bridge table which will simplify querying!

Persisted to BRDG	**Persisted to BRDG**				**Persisted to BRDG**
hub A (driver key)	**hub B**	EFS: Start Date	EFS: End Date	EFS: Load Date	Active
A101	AAA	2022-01-01	9999-12-31	2022-01-01	No
A101	AAA	2022-01-01	2022-10-09	2022-01-10	No
A101	BBB	2022-01-10	9999-12-31	2022-01-10	No
A101	BBB	2022-01-10	2022-10-14	2022-01-15	No
A101	**AAA**	**2022-01-15**	**9999-12-31**	**2022-01-15**	Yes

Hub A: A101 – Hub B: AAA is the current relationship

A **link table is a unique list of relationships**, the fact that the relationship 'A101'-'AAA' returned on 15 Jan 22 would not be visible in the link table alone, therefore the relationship *state* is needed in the adjacent satellite table, preferably if it comes from source. Alternatively, we must build an effectivity satellite to track this movement against the driver key.

Example Bridge Build	**Example Information Mart Build**
create table BRDG as select hub_A.business-key_A , hub_B.business-key_B from hub_A inner join link lnk on hub_A.hashkey_A = lnk.hashkey_A inner join hub_B on lnk.hashkey_B = hub_B.hashkey_B inner join ef_satellite ef on lnk.hashkey_lnk = ef.hashkey_lnk where current_date() between ef.start-dt and ef.end-dt;	create view IM as select BRDG.business-key_A , BRDG.business-key_E *, sat.<attributes>* from **BRDG** *[inner join satellite_B1 satB1* *on BRDG.hashkey_hub_B = satB1. hashkey_hub_B]*;

Bridge records are replaced at every run keeping the slice of applicable relationships to a minimum. If the requirement needed history *and* satellite table data, then you'd need to consider the effective relationship by driver key when combining regular satellite table records.

Deploying a Bridge with Aggregates

Reporting analytics can often include cumulative totals, sub-totals, and summaries but we advocate for the **lowest grain** measures to be stored in data vault, *why?* Because the data vault contains the single version of the facts. Should we need aggregations then we can easily re-create them when needed. What's more because data vault is bitemporal any *correction* does not need to be replayed. Only the correction is processed in place by applied date, the load date of course will be newer than the previous version of the record. Business events or transactions usually occur between participating business objects and for that reason these events are loaded to a link-satellite table. Now we could simply make the

link-satellite table available *"as is"* but we can also pre-calculate aggregates for the analytical users and supply that in the form of additional metric columns alongside the lowest grain measures in a fact table.
Because we use surrogate sequence ids in each satellite table, we can include those needed by the facts into the bridge table. Add the snapshot date as the date dimension surrogate key and we now have a fully formed Kimball star schema without even needing to physical create slowly changing dimension tables (SCD Type 2)!
Using the PIT flow manifold we can construct a very similar pattern for bridge tables and create multiple bridge tables based on reporting wants.

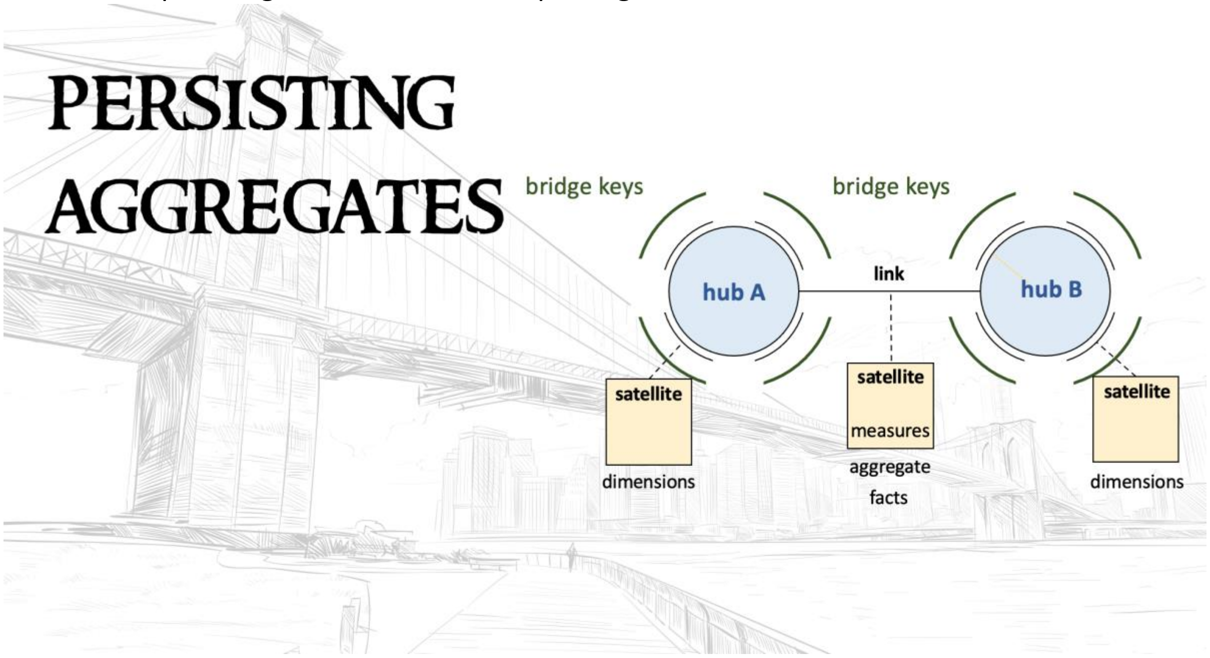


Figure 0-1 Not relying on the view query to run the aggregations

With the aggregations feeding into a fact table that we can easily tear down and rebuild the fact table as needed. It will likely be that the fact table contains more records than its surrounding fact dimensions and therefore the join algorithm used on the platform could end being a "Right-Deep Join Tree", a *hash join*.

satellite A dim A	**Persisted to BRDG**	**Persisted to BRDG**	**Persisted to BRDG**	**Persisted to BRDG**	**Persisted to BRDG**	satellite B dim B
satellite A SID	satellite A SID	hub A	metric	aggregated metric	satellite B SID	satellite B SID
555	555	A101	25	25	56	56
555	555	A101	13	38	212	212
7888	7888	A101	-20	18	212	212

On large workloads you could incrementally populate the fact table based on something like a Snowflake stream on the link-satellite.

In Closing

- **Are dimensional bridges different to data vault bridges?**

No, in fact if you encounter situations where a dimensional model bridge structure may solve a data vault problem then by all means go for it. PIT and Bridge tables are in the realm of query assistance and information marts, i.e., enhancing the reporting experience for the business. PITs and Bridges are *disposable* because of the underlying data vault will always support the information marts with. Is the relationship being solved by the bridge an *auditable* relationship? Do we need to go *back in time* to a *different* version of the relationship? Maybe in this instance a business vault link *could* be better utilised to resolve this.

- **Can we combine PIT and Bridge tables?**

Yes! A PIT table includes the load dates and hash keys for surrounding satellite tables of a hub or link table, simply include those columns in the bridge table you're constructing, and they will become the join keys to those satellite tables. What's more is if you have deployed auto-generated sequence ids in each satellite table we can use those ids instead of the load date and hash key in the bridge table and therefore narrowing the width of the bridge table even further, reducing the join columns and the bridge table now looks like the classic fact table!
PITs and Bridges mimic the dimensional data model structures that have been developed over decades and they too can take advantage of the data platform's algorithms to enhance your information mart experience just like dimensional marts have.

- Data Vault brings the malleable table structures that is conditioned to change.
- PITs & Bridges offer the query assistance to enhance your data vault querying experience; and
- Information Marts provide the data in an easier to consume structure

Bridge tables are just another tool you can utilise to optimally get the data out of data vault through equi-joins! These examples in this article are some of the ways you can simplify querying a data vault, should you have other methods don't let the *standard* ways limit you!

References:

- Unified Star Schema, - bit.ly/39IPWJ9
- Business Vault Link example "Apache Spark GraphX and the Seven Bridges of Königsberg", bit.ly/3ezZ6Wh

- Data Vault Mysteries… Effectivity Satellite and Driver Key, bit.ly/3oS4k70
- Data Vault PIT Flow Manifold, bit.ly/3iEkBJC
- WHY EQUIJOINS MATTER! bit.ly/3dBxOQK
- A Rose by any other name… wait.. is it still the same Rose? bit.ly/3xlFK0s