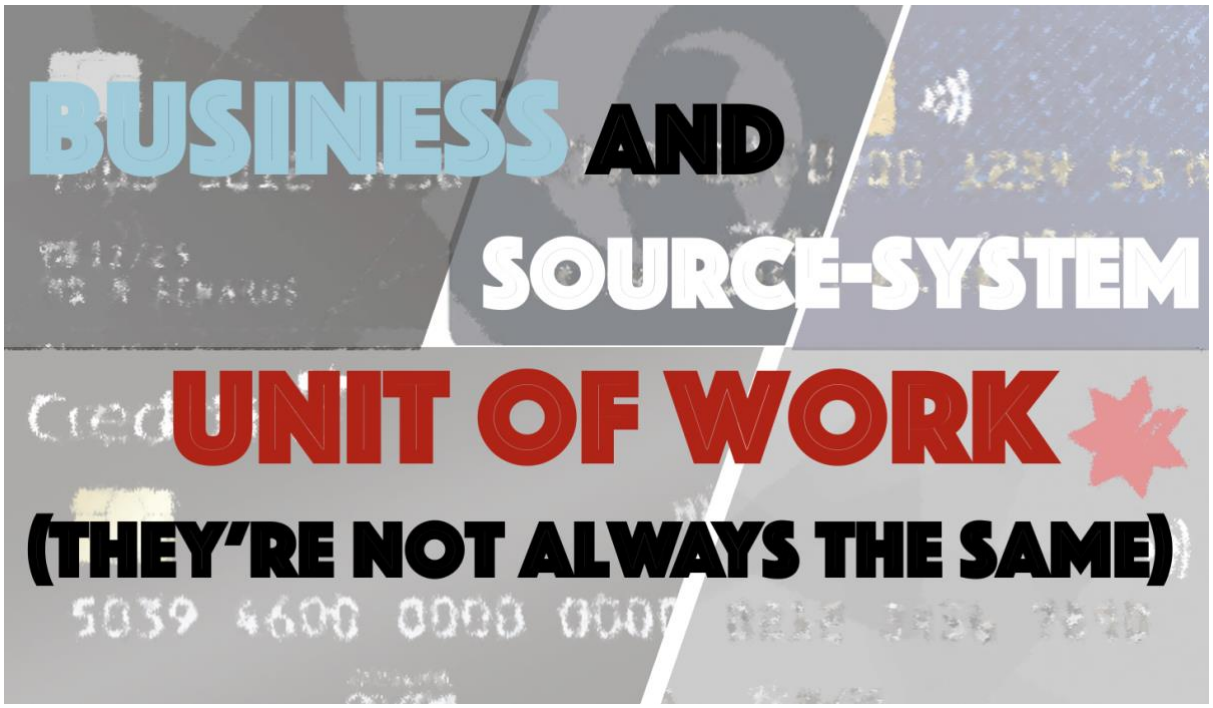
Business and Source-System Unit of Work (they're not always the same)

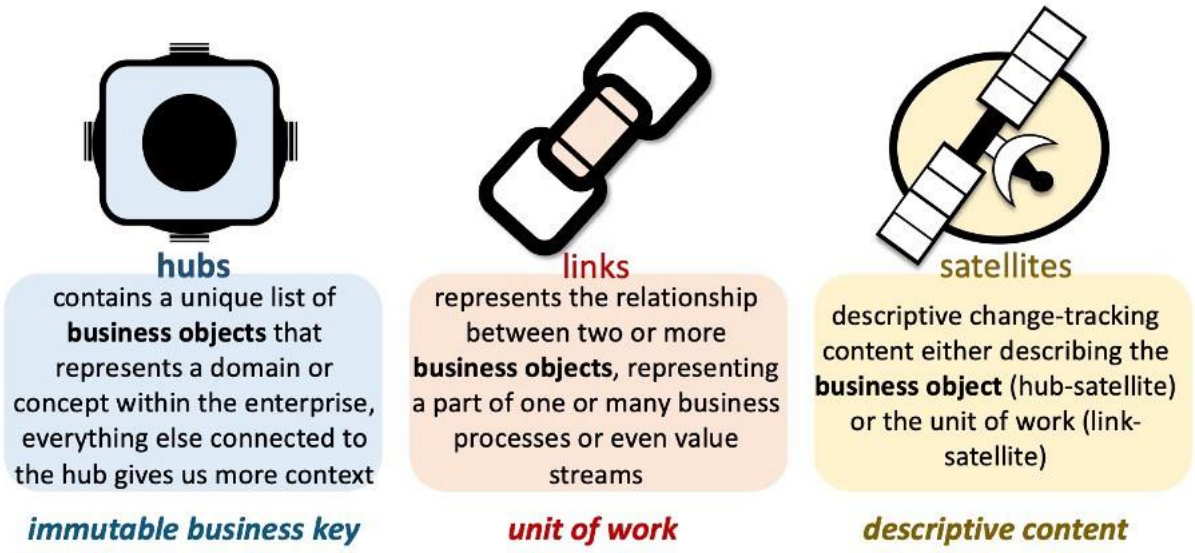


How a software system depicts the **unit of work**, **relationship** or participating business entities in a **transaction** may be *different* to how the organization's own business rules see these business interactions. No one purchases commercial off-the-shelf (COTS) software that *doesn't* automate parts or *all* their business processes, but you *may* find that how the software vendor depicts those rules and processes do not ***exactly*** fit how you would like to depict or report on them. There may be a myriad of reasons for this such as inadequate testing of the source platform before purchase, lack of support for changes or simply that the budget dictates you implement a less than optimal software package to automate your business processes! *This is the reality!*

And so, if the source cannot match your business processes and rules *exactly* and are not able to mould their processes to mirror yours, you may need to solve these business process gaps for yourself. Most of the time, this occurs because that vendor needs to look after their own bottom line and thus have a more generic view of that process or rule to cater for a wider range of *their* own customers.

Today's topic is focused on how to depict source-aligned units of work with the business' unit of work (UoW), they *may* differ, and by example we'll discuss why and the rules behind building a **raw vault** link and a **business vault** link.

As a reminder, the data vault table types…



For a definition of the terms discussed in this article, please visit:

- How part of this solution was solved using Apache Spark, "Apache Spark GraphX and the Seven Bridges of Königsberg", bit.ly/3ezZ6Wh
- Defining Business Vault, "Data Vault Mysteries… Business Vault", bit.ly/3rfV7V3
- The use of Zero Keys, "Data Vault Mysteries… Zero Keys & Ghost Records…", bit.ly/3vjTXdg
- What is a Bridge Table, "The Lost Art of Building Bridges", bit.ly/3MFnXZr

The Business Case

For this use case, a credit card source system is designed to manage credit cards and credit card movement *only*. Cards can be issued in one of two configurations, either they are issued as:

- as a standalone card (as a standalone card product), or
- as primary and secondary/supplementary cards (as a "family" product)

The supplementary and primary cardholder activity is tracked by a "controlling" card number that the customer does not see but is used internally by the source system to track purchases, repayments and rewards made by the members of these cards in the "family" product. The standalone card does not need a controlling card number because it is standalone and we only need to track activity against a single card, not an umbrella of cards.

CARD_NUMBER	CONTROL_CARD	CARD_TYPE
SA-1-ABC		
CA-4-321		A
CP-4-123	CA-4-321	P
CS-4-456	CA-4-321	S

Figure 1 Two configurations of credit cards

For simplification, I have depicted credit card numbers as:

- 'SA' for **S**tand**A**lone, card type is null
- 'CP' for **C**onsolidated **P**rimary Cardholder, card type is 'P'
- 'CS' for **C**onsolidated **S**upplementary/secondary card, card type is 'S', and

- ‘CA’ for hidden **C**onsolidated Card, card type is ‘A’
- The middle number is the scenario number, and
- The last three digits is a unique arbitrary value

The source system can track the movement of credit cards in terms of:

a. **Lost / stolen credit cards** – when this happens a new card is issued, and the old card is marked as lost or stolen by changing the card type to ‘X’. For the standalone card, a new standalone card is issued, for the primary card a new primary card is issued and for the supplementary card a new supplementary card is issued. When there is a product change it *could* change the consolidated card value, but the old, consolidated card does not get marked with an ‘X’ under card type as it cannot be lost/stolen.

b. **Changes product** – a standalone cardholder can choose to change product and convert into the family/consolidated product of cards. When this happens, the standalone card is terminated (marked with an ‘X’) and a primary card is issued in its place, a supplementary card is optionally issued and a designated “controlling” card number is issued too. On an extremely rare occasion the opposite could occur, a family/consolidated product of cards can convert to a standalone card and when this happens the primary card is terminated (marked with an ‘X’) and a standalone card is issued in its place, the supplementary card is terminated but no card issued in its place and the controlling card ceases to be active but **not** terminated with an ‘X’.

Note that the controlling card is **not** a card **account** number, but rather a way of tracking activities for these cards *as a family*.

The data is provided from the source as a growing snapshot of the card movement history and the source supplies the **active card data** in a single row, a column with the transferred-to field is set to null, if the transferred-to column populated it indicates that either the card was lost/stolen, or the cardholder has opted in to a different product of cards. The value in the transferred-to column is the value of the next credit card the card in that table row has transferred to.

To find the **total movement** of a card you must recursively traverse the lineage of cards.

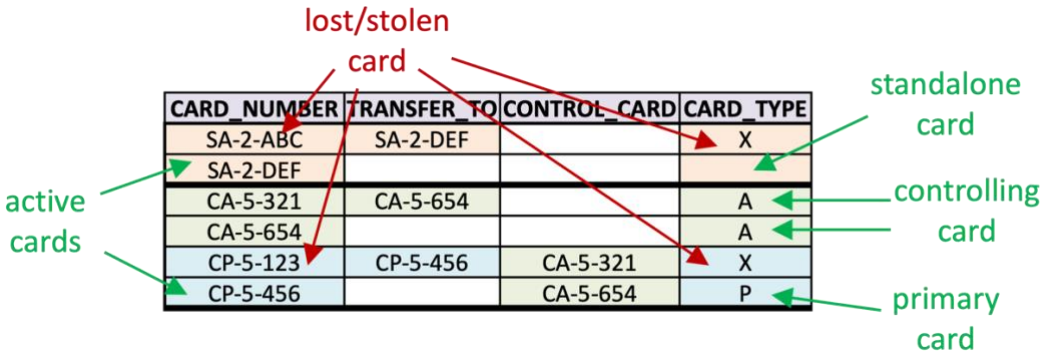


Figure 2 Lost/Stolen event for each card type

‘SA-2’DEF’ is the active standalone card in scenario 2, ‘SA-2-ABC’ was terminated
‘CP-5-456’ is the active primary card for scenario 5, ‘CP-5-123’ was terminated **and** ‘CA-5-321’ converted to ‘CA-5-654’ consolidated “hidden” card number.

Raw Vault Link …Rules

Now that we have the business case how and what do we model in the data vault? First, we need to determine that we need to define what the **business keys** are.
For the above source depiction, we have three business key columns and through data profiling these with the assigned **subject-matter expert** we determine that all three can collapse into one business key column in the card account hub table, each representing credit card transactions, payments and rewards attached to that business key when those individual cards were active (hasn't been transferred).

What are the units of work?

We determine that there are potentially two distinctive UoWs,

1. active-card and "transferred to" depicts the movement that occurs when a card is lost/stolen. The same two columns are used when a product conversion occurs, either from standalone to family, or family to standalone. The movement relationship resembles a **same-as link** and is depicted as such with the credit card type column set to 'X' (as noted above). In this instance because the card type is about the card itself and not the movement, this column is added to a hub-satellite table and **not** a link-satellite table.
2. *What about the relationship between primary and supplementary card with the parent controlling card?* This is a **hierarchical relationship** and *could* be represented in the **hierarchical link table**.

By splitting these two relationships out, *have we broken the unit of work?* i.e., the participating cards in the relationship at the point in time are active at *that* point in time, would we better off keeping them together as a regular link table or splitting them into a same-as link and hierarchical link respectively? A tough choice, but with the SME's help we can solve this!
The case for keeping them together is that we can easily recreate the source at any point in time.
The case for splitting them into two link tables is that we will have these relationships *fit* into what we know as "same-as" and "hierarchical" relationships and as physical link tables. We can still recreate the source, but we would effectively be combining the link tables in a link-to-link relationship to recreate the source *if we need to*.

*Pro tip: same-as and hierarchical links are **logical** representations of relationships between business entities, they do not need to be conformed into the same physical structures. If you had both in a unit of work as we have depicted above, then what type of link table is it? In all cases, all physical variations of link tables are still simply just link tables, the name chosen depicting these relationships are just **logical names** for these relationships themselves. In a single unit of work, you can have as few as two participating business entities and as many as you like. What makes the same-as and hierarchical relationship unique is that the two or more business entities come from the **same** hub table. **No** this is not a **peg-legged link**, these do not exist and is poor data vault modelling practice if you are even considering it!*

What if another source system is used that depicts that same relationship, do we load to the same link table or do we load it into a separate link table?
Single raw vault link table per source is preferred but multiple sources to a single link table *can* be supported. A few things to consider if you attempt either approach:

- The link table is a unique list of relationships, by combining multiple sources into one link table then you will not be able to tell *which* source supplied *that* relationship.
- If combined into a single link table, then which is the trusted source of truth? Why would you have two or more business rule automation tools depicting the same relationship? Do we have redundant source systems?!?
- To circumvent the issue of not knowing which source supplied the unit of work you *could* introduce a dependent child key into the link table to denote where those records came from, but…
    - The dependent-child key in the link table is an arbitrary value that you must now maintain
    - Every query needing the unique relationship must now also execute a SELECT DISTINCT when retrieving that record, an unnecessary cost for you.
    - You could instead keep this link table unique and instead of using a dependent-child key in the link table, rely on the link satellite to determine which source sent that record (if you even want to do that), but then this just means you need to execute a join to yet another satellite table to determine *this* fact. Something that could still be an optimal operation when executing with Snowflake's RELY syntax.
- A valid use case for multiple sources feeding into a single link table could be in the case of a migration from one source system to another.
- Note however, if the link structures are combined then to recreate the source you must rely on either the dependent-child key or the link-satellite to accurately re-create that source.

Weigh these considerations, do we have less tables and the UoW is combined? Do we care which source sent that record? Or do we have separate link tables depicting the same UoW?

For this example, we will break up our sample into separate same-as and hierarchical link tables, *onto the next challenge…*

Is this how the business needs the unit of work?
There is one **glaring shortcoming** of **this unit of work** and it relates to the requirement, "how do we now report on all this movement for a cardholder to the regulator?" You may have noticed that we need to *traverse* by either movement (lost/stolen or product conversion) every time we need to report on the activities of the cardholder. For each cardholder (standalone, primary, and supplementary) there are activities such as purchases, payments and rewards earned. To the regulator they do not want to see multiple card numbers for a single cardholder, they need a single representation of that cardholder. They need to see an **account number**.
In our situation we do not have one!

The business needs an account number/id for several reasons, regulatory reporting, fraud analysis and much more! When looking at the options for combining these cards together into an account number we could either:

- Issue an account number to the cardholder by assigning it a **random value**. The problem with this approach is that we do not have attributes for a cardholder that is *deterministic* to base that algorithm on. The card number changes, and it could change to anything depending on the business events affecting that customer.

- Invent a clever algorithm to come up with a deterministic account number for the cardholders. Is this worth the effort? What variables can we use that are not in itself identifying and creates a value that we now must treat as **personally identifiable** that incurs a **cost to maintain** (maybe not now but certainly in the future).

No, the answer is much simpler: use the **first card issued** as the **account number** for all related cards (lost/stolen and conversion).
The first card issued in the card lineage will ***never*** change!

How a Business Vault Link solves it

One of the core jobs of how you model data into a data vault is you solve a lot of the query-time complexities upfront for the user. That is,

- we perform satellite splitting to **split** data according to what they are describing whether they contain personally identifiable data and the speed at which the attributes update.
- we simplify and optimise join conditions for satellite tables surrounding a hub or link table using point-in-time (PIT) and bridge structures;
- we solve unit-of-work complexity once for all users of the data vault.

The last bullet is the relevant point to this story, and we will do it through profiling card movement scenarios and produce a single business vault link that will act as the single version of the facts, so business users and analysts have a single approved method (query) to retrieve the above information, **the account number**.

To profile and test this, we have profiled the data through various scenarios, for this story we will depict 16 such unique scenarios for card movement, and our business rule implemented as an SQL statement will solve it.

Scenarios

From the most common to the least, each diagram below is a table followed by a graph of the card relationships depicted from those tables.

Scenario 1: the standalone card

Not complex, single card issued, single record

CARD_NUMBER	TRANSFER_TO	CONTROL_CARD	CARD_TYPE
SA-1-ABC			

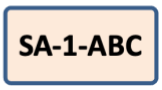


Figure 3 Standalone Card

Account number will be: ‘SA-1-ABC’.

Scenario 2: the standalone card that was lost/stolen

Not complex, two records, the active card, and the lost/stolen card it came from.

CARD_NUMBER	TRANSFER_TO	CONTROL_CARD	CARD_TYPE
SA-2-ABC	SA-2-DEF		X
SA-2-DEF			

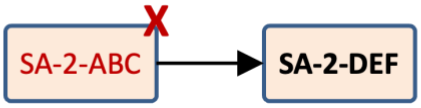


Figure 4 Standalone Card, lost/stolen

Account number will be: 'SA-2-ABC'. It is the first card issued in the card lineage.

Scenario 3: primary card without a supplementary card

The family card has a controlling "hidden" card with no supplementary card issued.

CARD_NUMBER	TRANSFER_TO	CONTROL_CARD	CARD_TYPE
CA-3-321			A
CP-3-123		CA-3-321	P

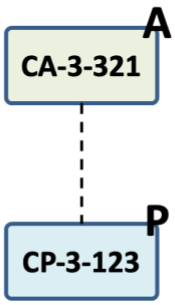


Figure 5 Family card, primary card issued

Account number will be: 'CP-3-123'. It is the card that was issued to the customer, *we will see why this is a better candidate than the "hidden" consolidated card number.*

Scenario 4: primary and supplementary card

A normal family card product scenario.

CARD_NUMBER	TRANSFER_TO	CONTROL_CARD	CARD_TYPE
CA-4-321			A
CP-4-123		CA-4-321	P
CS-4-456		CA-4-321	S

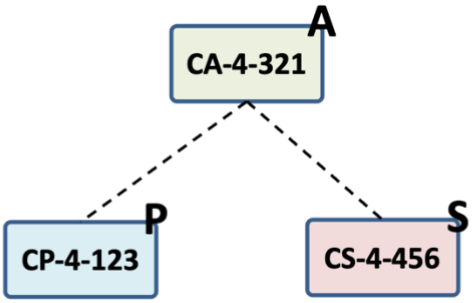


Figure 6 Family of cards, primary + supplementary

Account number will be: 'CP-4-123'. As with the previous scenario, we will show why primary card number is a better candidate than the other cards in the consolidated "family" of cards.

Scenario 5: primary card that was lost/stolen

Showing a simplistic scenario for lost/stolen, notice how the hidden controlling card number changed when the primary card changed.

CARD_NUMBER	TRANSFER_TO	CONTROL_CARD	CARD_TYPE
CA-5-321	CA-5-654		A
CA-5-654			A
CP-5-123	CP-5-456	CA-5-321	X
CP-5-456		CA-5-654	P

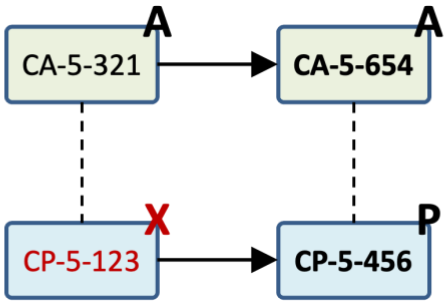


Figure 7 A lost/stolen event has occurred which also migrated the controlling card number

Account number will be: 'CP-5-123'.

Scenario 6: supplementary card was lost/stolen

Like previous scenario but just showing what occurs when the supplementary card is lost/stolen. Notice how the controlling card **did not change** when the supplementary card is lost/stolen.

CARD_NUMBER	TRANSFER_TO	CONTROL_CARD	CARD_TYPE
CA-6-321			A
CP-6-123		CA-6-321	P
CS-6-456	CS-6-789	CA-6-321	X
CS-6-789		CA-6-321	S

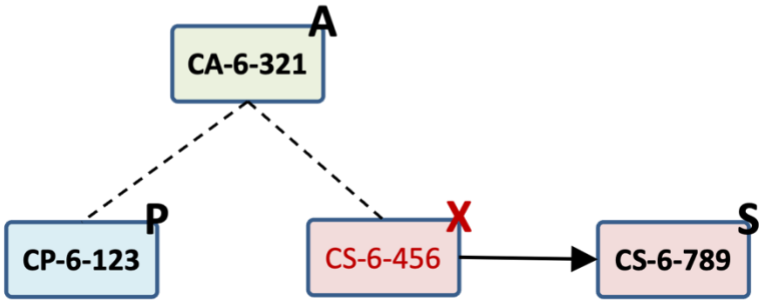


Figure 8 Lost/stolen event for the supplementary card does not migrate the hidden card number

Account number will be: 'CP-6-123'. We need a consistent lineage across card movements, this will become evident when we depict product conversion in the next scenario.

Scenario 7: standalone card converts to a primary card

A product conversion, normal.

CARD_NUMBER	TRANSFER_TO	CONTROL_CARD	CARD_TYPE
SA-7-123	CP-7-123		X
CA-7-321			A
CP-7-123		CA-7-321	P

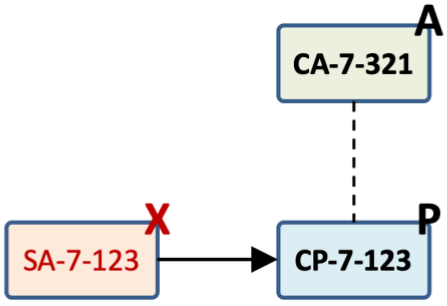


Figure 9 a product change, from Standalone to Primary card

Account number will be: 'SA-7-123'. The lineage according to our business rules in a product conversion scenario is to convert a standalone card to a primary card.

Scenario 8: standalone card is lost/stolen and later converts to primary card, and it too is lost/stolen

Combines scenarios 2, 7 and 5 above.

CARD_NUMBER	TRANSFER_TO	CONTROL_CARD	CARD_TYPE
SA-8-123	SA-8-456		X
SA-8-456	CP-8-123		X
CA-8-321	**CA-8-654**		A
CA-8-654			**A**
CP-8-123	**CP-8-456**	CA-8-321	X
CP-8-456		**CA-8-654**	P

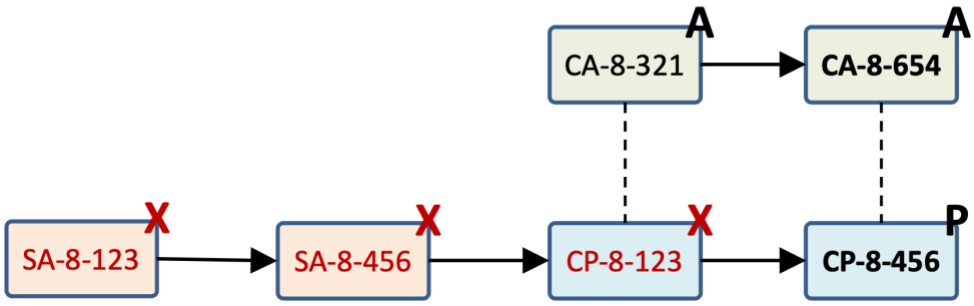


Figure 10 Expanded form the previous, standalone is lost/stolen and then converted

Account number will be: 'SA-8-123'. The first card issued.

Scenario 9: primary card converts to a standalone card

Ultra-rare scenario but supported, the reverse of scenario 7.

CARD_NUMBER	TRANSFER_TO	CONTROL_CARD	CARD_TYPE
CA-9-321			A
CP-9-123	**SA-9-123**	CA-9-321	X
SA-9-123			

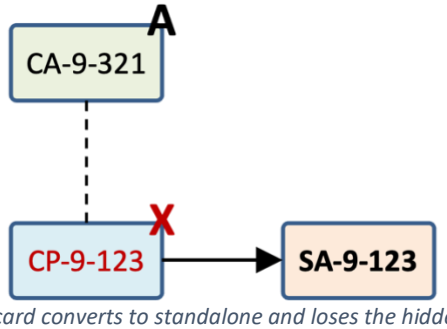


Figure 11 Primary card converts to standalone and loses the hidden card number

Account number will be: 'CP-9-123'. This is the first card issued, *you begin to see a pattern…*

Scenario 10: primary card is lost/stolen and later converts to a standalone card, and it too is lost/stolen

Combines scenarios 5, 9 and 2

CARD_NUMBER	TRANSFER_TO	CONTROL_CARD	CARD_TYPE
CA-10-321	CA-10-654		A
CA-10-654			A
CP-10-123	CP-10-456	CA-10-321	X
CP-10-456	SA-10-123	CA-10-654	X
SA-10-123	**SA-10-456**		X
SA-10-456			

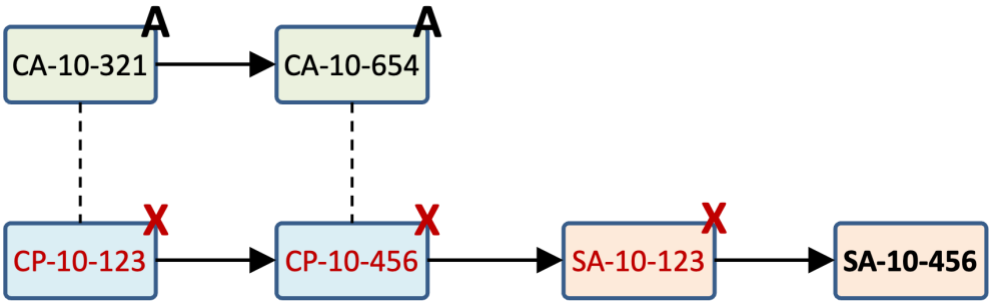


Figure 12 Expanded scenario

Account number will be: 'CP-10-123'. The rule is consistent.

Scenario 11: standalone card is lost/stolen and later converts to a primary card, and it too is lost/stolen, and the supplementary card is lost/stolen as well
Combines scenarios 2, 7, 5 and 6

CARD_NUMBER	TRANSFER_TO	CONTROL_CARD	CARD_TYPE
SA-11-123	SA-11-456		X
SA-11-456	CP-11-123		X
CP-11-123	CP-11-456	CA-11-123	X
CP-11-456		**CA-11-456**	P
CS-11-123	CS-11-456	CA-11-123	X
CS-11-456		**CA-11-456**	S
CA-11-123	CA-11-456		A
CA-11-456			A

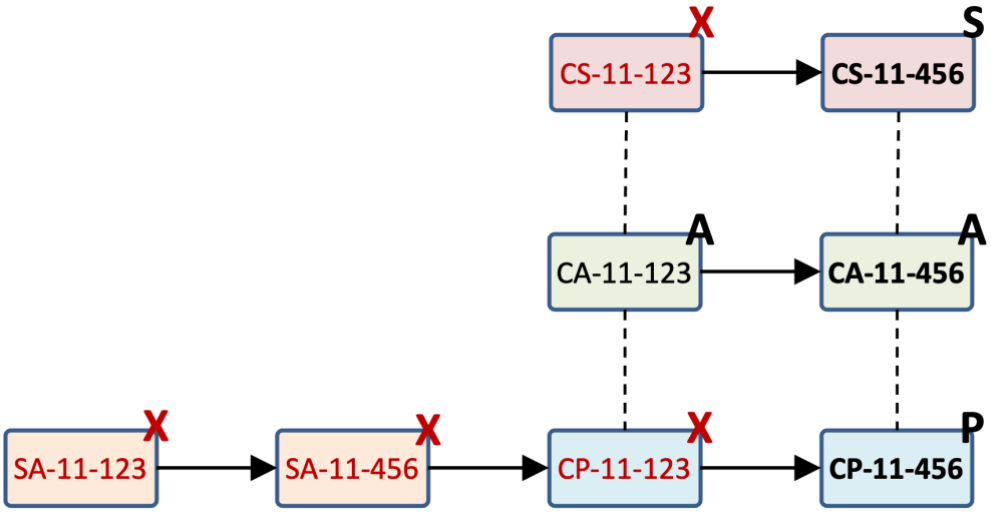


Figure 13 A lot of movement to show this scenario, maybe fraud?

Account number will be: 'SA-11-123'. *The supplementary card is not a reliable source of card lineage.*

Scenario 12: standalone card is lost/stolen, converts to a primary card that is lost/stolen too but converts back to a standalone card

Combines scenarios 2, 7, 5 and 9

CARD_NUMBER	TRANSFER_TO	CONTROL_CARD	CARD_TYPE
SA-12-123	SA-12-456		X
SA-12-456	CP-12-123		X
CP-12-123	CP-12-456	CA-12-123	X
CP-12-456	**SA-12-789**	CA-12-123	X
SA-12-789			
CA-12-123			A

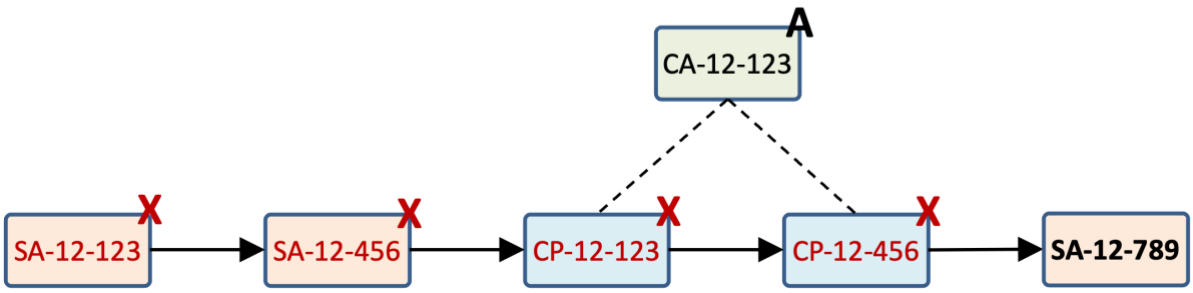


Figure 14 a reversal of converting to a family card

Account number will be: 'SA-12-123'. Ah yes, *still the same rule!*

Scenario 13: standalone, primary, and supplementary cards issued with a controlling card.

An error scenario that breaks our business rules, a standalone card cannot be issued a "hidden" controlling card number.

CARD_NUMBER	TRANSFER_TO	CONTROL_CARD	CARD_TYPE
SA-13-123		CA-13-123	
CP-13-123		CA-13-123	P
CS-13-123		CA-13-123	S
CA-13-123			A

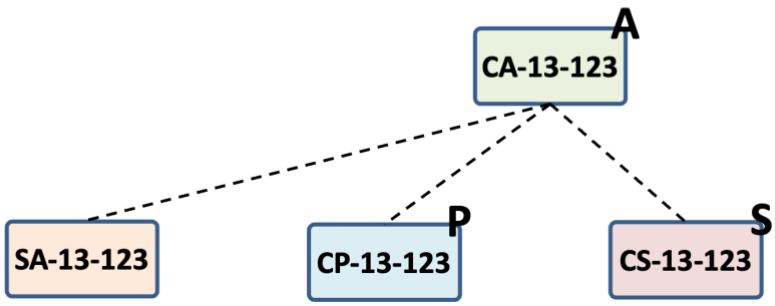


Figure 15 The source-system has a bug

Account number will be: 'SA-13-123'. If we face this **outlier**, *the standalone card takes precedence!*

Scenario 14: standalone becomes primary, primary becomes standalone and then the standalone becomes a primary again

Combines scenarios 7, 9 and 7

CARD_NUMBER	TRANSFER_TO	CONTROL_CARD	CARD_TYPE
SA-14-123	CP-14-123		X
CP-14-123	SA-14-456	CA-14-456	X
SA-14-456	CP-14-456		X
CP-14-456	**CP-14-789**	CA-14-123	X
CP-14-789			P
CA-14-123	**CA-14-456**		A
CA-14-456			A

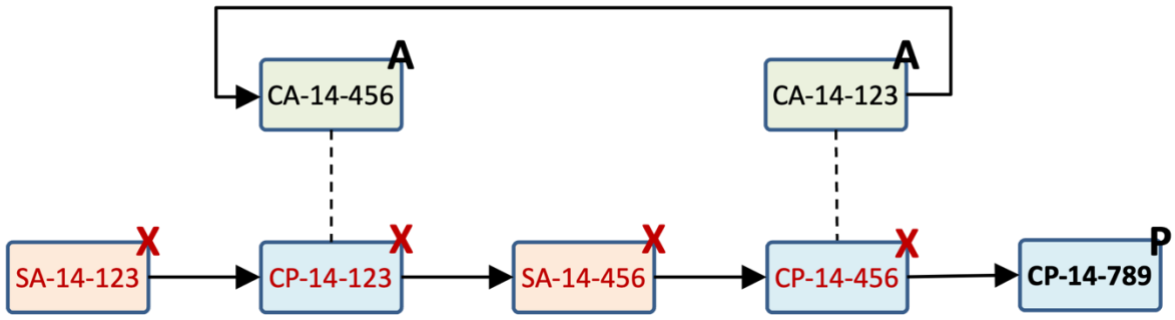


Figure 16 Movement that shouldn't occur!

Account number will be: 'SA-14-123'. Somehow, the source system has represented the controlling card in reverse! This situation is **irrelevant** to the goal of the business rule *however*, thus this outlier has **no impact** to the rule outcome!

Scenario 15: primary is lost/stolen, becomes standalone, becomes primary again, which is then lost/stolen along with the supplementary card

Combines scenarios 5, 9, 7, 5 and 6

CARD_NUMBER	TRANSFER_TO	CONTROL_CARD	CARD_TYPE
CP-15-456	SA-15-123	CA-15-456	X
SA-15-123	CP-15-789		X
CP-15-789	**CP-15-012**	CA-15-456	X
CP-15-012		**CA-15-789**	P
CS-15-456		**CA-15-789**	S
CA-15-123	CA-15-456		A
CA-15-789			**A**

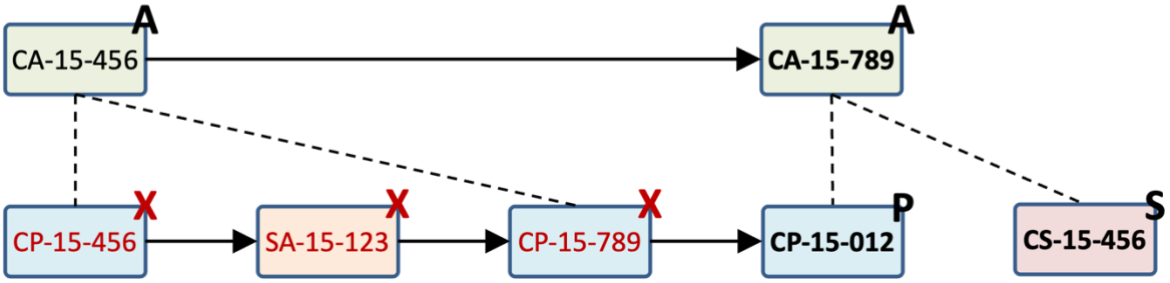


Figure 17 Expands on a previous 'plausible' scenario

Account number will be: 'CP-15-456'. This now highlights that although the first consolidated card "family" had no supplementary card assigned, it does not impact the overall business rule!

Scenario 16: standalone card becomes a supplementary cardholder

Error scenario, a standalone card should only ever convert into a primary

CARD_NUMBER	TRANSFER_TO	CONTROL_CARD	CARD_TYPE
SA-16-123	SA-16-456		X
SA-16-456	**CS-16-123**		X
CS-16-456		**CA-16-456**	S
CP-16-123	**CP-16-456**	CA-16-123	X
CP-16-456		**CA-16-456**	P
CA-16-123	**CA-16-456**		A
CA-16-456			**A**

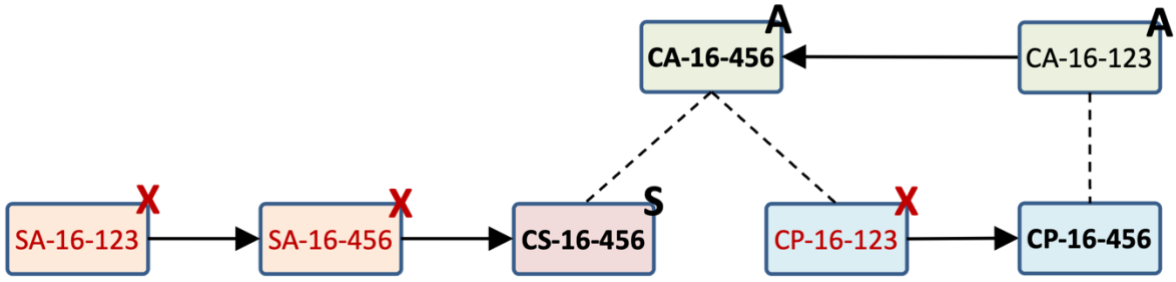


Figure 18 Can we fix this?

Account number will be: 'SA-16-123'. This should never occur and is in fact an error in the source system, *can we solve for this?*

Solving tech debt with SQL

As complex as these scenarios appear we can in fact solve this complexity in a single SQL statement, albeit a large SQL statement broken up into **common table expressions** (CTE). The main block of this code must support variable card lineage in depth, find a starting point by defining the **recursive query** **anchor** and traverse back to the first card issued to identify that card as the **grandparent** card and the best account number candidate.
The anchor is easy to determine, it's the card that has **not** been transferred.

STEP 1: Unpack the Data Vault tables

Using our raw vault tables, we unpack the needed business keys into a flat table-view that resembles the source. Notice (below) how we can do this using an equi-join, the **zero-key** makes this possible just like the **ghost record** is used in the construction of an equi-join between satellite tables and a hub table or satellite tables and a link table for point-in-time (PIT) table efficacy. Again, this is why equi-joins matter!

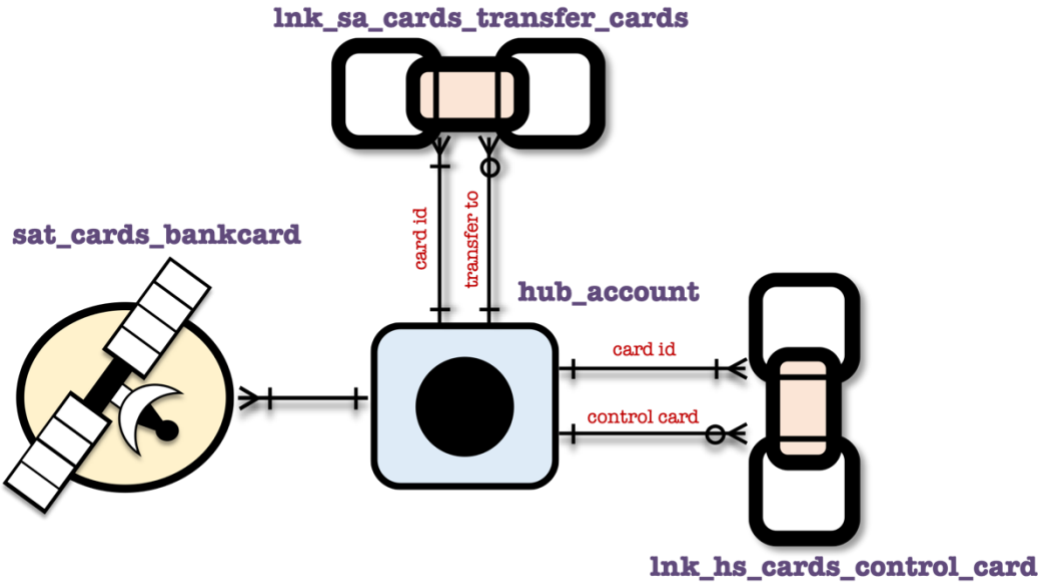


Figure 19 Raw Vault Hub, Link and Satellite tables

```
-- 1. unpack DV
with base_account as (
select hub.account_id as CARD_NUMBER
, case when hub2.account_id ='-1' then null else hub2.account_id end as TRANSFER_TO
, case when hub3.account_id ='-1' then null else hub3.account_id end as CONTROL_CARD
, sat.CARD_TYPE
from hub_account hub
inner join lnk_sa_cards_transfer_cards lnk
on hub.dv_hkey_hub_account=lnk.dv_hkey_hub_account
inner join hub_account hub2
on lnk.dv_hkey_hub_account_transfer_Card=hub2.dv_hkey_hub_account
inner join lnk_hs_cards_control_card lnk2
on hub.dv_hkey_hub_account=lnk2.dv_hkey_hub_account
inner join hub_account hub3
on lnk2.dv_hkey_hub_account_control_Card=hub3.dv_hkey_hub_account
inner join (select s.dv_hkey_hub_account, s.CARD_TYPE
        from sat_cards_bankcard s
```

```
        inner join (select dv_hkey_hub_account, max(dv_rtts) as dv_rtts, max(dv_ldts) as dv_ldts
                from sat_cards_bankcard
                    group by dv_hkey_hub_account) sm
        on s.dv_hkey_hub_account=sm.dv_hkey_hub_account
        and s.dv_ldts=sm.dv_ldts
        and s.dv_rtts=sm.dv_rtts) sat
on hub.dv_hkey_hub_account=sat.dv_hkey_hub_account
)
```

STEP 2: Track Card Lineage – Recursive CTE

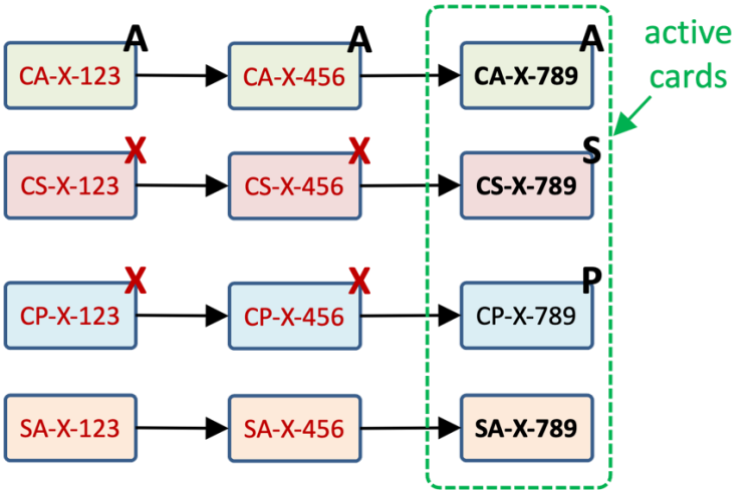


Figure 20 Card lineage

The anchor is the **latest card**, it has not been transferred to anything and therefore we can use a recursive CTE with the latest card as the starting point in Snowflake to build this lineage graph.

For "family" cards this step establishes the lineage for primary, supplementary and the hidden controlling card **independently**. We must apply business rules to bring these independent card lineages together and establish the grandparent of all cards within a family.

*Did you know, at the time of writing, **recursive** CTEs are still **not** possible in Apache Spark or HiveSQL?*

- *Apache Spark SQL - https://issues.apache.org/jira/browse/SPARK-24497*
- *Apache Hive SQL - https://issues.apache.org/jira/browse/HIVE-16725*

```
-- 2. Trace card lineage
, account_map as
(-- anchor
  select CARD_NUMBER, TRANSFER_TO, CONTROL_CARD, CARD_TYPE, 1 as account_level, CARD_NUMBER as
latest_card
  from base_account
  where TRANSFER_TO is null
  -- loop through to get to the lowest level
  union all
  select r.CARD_NUMBER, r.TRANSFER_TO, r.CONTROL_CARD, r.CARD_TYPE, l.account_level + 1 as account_level,
l.latest_card
  from base_account r
  inner join account_map l
  on r.TRANSFER_TO=l.CARD_NUMBER)
```

This gives us two columns for the business rule that is needed for the next step to work,

- **account_level** – this is a count, starting from the anchor card all the way to the end of the recursive operation
- **latest_card** – this is the active card; it is the only value we can use to tie all cards in a lineage together

STEP 3: Categorize grandparent and latest cards

Using SQL Window functions, we're able to categorize what that first card issued as either

- standalone,
- control,
- primary,
- supplementary

Why was this necessary? Well, if you noticed when a card is lost/stolen its card type changes to 'X', terminated due to lost/stolen or product conversion event! Without being able to track the lineage we're not able to determine if the card was one of the above card types!

```
-- 3. categorize grandparent and latest cards
, card_summary as
(-- assign grandparent
select CARD_NUMBER, TRANSFER_TO, CONTROL_CARD, CARD_TYPE, account_level, latest_card
, first_value(CARD_NUMBER) over (partition by latest_card order by account_level desc) as grandparent_id
, case when first_value(CONTROL_CARD) over (partition by latest_card order by account_level desc) is not null then
'consolidated'
     when CARD_TYPE='A' then 'control'
  else 'standalone'
  end as grandparent_card_type
, case when first_value(CARD_TYPE) over (partition by latest_card order by account_level) is null then 'standalone'
     when first_value(CARD_TYPE) over (partition by latest_card order by account_level) = 'A' then 'control'
               when first_value(CARD_TYPE) over (partition by latest_card order by account_level) = 'P' then 'primary'
               when first_value(CARD_TYPE) over (partition by latest_card order by account_level) = 'S' then
'supplementary'
               when first_value(CARD_TYPE) over (partition by latest_card order by account_level) = 'X' then 'transfer'
  else 'error'
  end as latest_card_type
from account_map
)
```

Why not use last_value instead of first_value? Well, last_value will likely be 'X' for a lost/stolen or product conversion event.

For a scenario where the candidate grandparent_id is the same across all rows for a scenario then we can consider the scenario to be **solved** and the grandparent_id will eventually become the **account_id**, although the code does not need a "solved" portion to limit any *additional* logic applied to it, the remainder of the code will not affect the mapped account_id.

The following scenarios have been **solved** in this step:

- Scenario 1: the standalone card
- Scenario 2: the standalone card that was lost/stolen

CARD_NUMBER	TRANSFER_TO	CONTROL_CARD	CARD_TYPE	ACCOUNT_LEVEL	LATEST_CARD	GRANDPARENT_ID
SA-1-ABC				1	SA-1-ABC	SA-1-ABC
SA-2-ABC	SA-2-DEF		X	2	SA-2-DEF	SA-2-ABC
SA-2-DEF				1	SA-2-DEF	SA-2-ABC

STEP 4: Assign grandparent

Issue the grandparent account number ensuring we do not assign the grandparent based on supplementary card lineage, a supplementary card can **never** be a grandparent

```
-- 4. assign grandparent based on rules (do not attempt to assign supplementary card here)
, cards_mapped as (
select distinct
  p.CARD_NUMBER as card_id
, p.TRANSFER_TO
, p.CONTROL_CARD
, p.CARD_TYPE
, p.account_level
, p.latest_card
, p.grandparent_id as orig_card_id
, p.latest_card_type
-- assign account_id
, case when p.grandparent_card_type in ('consolidated', 'standalone') and p.latest_card_type <> 'supplementary' then
p.grandparent_id
      when p.grandparent_card_type = 'control' then c.grandparent_id
      else null
                end as account_id
                , p.grandparent_card_type as p_grandparent_card_type
from card_summary p
left join card_summary c
on p.CARD_NUMBER=c.CONTROL_CARD
and c.latest_card_type <> 'supplementary'
and p.latest_card_type <> 'supplementary'
)
```

For a scenario where the candidate account_id is the same across all rows for a scenario then we can consider the scenario to be **solved** and the **account_id** will eventually become the final account_id, although the code does not need a “solved” portion to limit any *additional* logic applied to it, the remainder of the code will not affect the final mapped account_id.

The following scenarios have been **solved** in this step:

- Scenario 3: primary card without a supplementary card
- Scenario 5: primary card that was lost/stolen
- Scenario 7: standalone card converts to a primary card
- Scenario 8: standalone card is lost/stolen and later converts to primary card, and it too is lost/stolen
- Scenario 9: primary card converts to a standalone card
- Scenario 10: primary card is lost/stolen and later converts to a standalone card, and it too is lost/stolen

- Scenario 12: standalone card is lost/stolen, converts to a primary card that is lost/stolen too but converts back to a standalone card
- Scenario 14: standalone becomes primary, primary becomes standalone and then the standalone becomes a primary again

	CARD_ID	TRANSFER_TO	CONTROL_CARD	CARD_TYPE	ACCOUNT_ID
primary card is the account id →	CA-3-321			A	CP-3-123
	CP-3-123		CA-3-321	P	CP-3-123
	CA-5-321	CA-5-654		A	CP-5-123
primary card is the account id →	CA-5-654			A	CP-5-123
	CP-5-123	CP-5-456	CA-5-321	X	CP-5-123
	CP-5-456		CA-5-654	P	CP-5-123
standalone card is the account id →	CA-7-321			A	SA-7-123
	SA-7-123	CP-7-123		X	SA-7-123
	CP-7-123		CA-7-321	P	SA-7-123
	CA-8-321	CA-8-654		A	SA-8-123
standalone card is the account id →	CA-8-654			A	SA-8-123
	SA-8-123	SA-8-456		X	SA-8-123
	SA-8-456	CP-8-123		X	SA-8-123
	CP-8-123	CP-8-456	CA-8-321	X	SA-8-123
	CP-8-456		CA-8-654	P	SA-8-123
primary card is the account id →	CA-9-321			A	CP-9-123
	CP-9-123	SA-9-123	CA-9-321	X	CP-9-123
	SA-9-123				CP-9-123
	CA-10-321	CA-10-654		A	CP-10-123
primary card is the account id →	CA-10-654			A	CP-10-123
	CP-10-123	CP-10-456	CA-10-321	X	CP-10-123
	CP-10-456	SA-10-123	CA-10-654	X	CP-10-123
	SA-10-123	SA-10-456		X	CP-10-123
	SA-10-456				CP-10-123
standalone card is the account id →	CA-12-123			A	SA-12-123
	SA-12-123	SA-12-456		X	SA-12-123
	SA-12-456	CP-12-123		X	SA-12-123
	CP-12-123	CP-12-456	CA-12-123	X	SA-12-123
	CP-12-456	SA-12-789	CA-12-123	X	SA-12-123
	SA-12-789				SA-12-123
	CA-14-123	CA-14-456		A	SA-14-123
standalone card is the account id →	CA-14-456			A	SA-14-123
	SA-14-123	CP-14-123		X	SA-14-123
	CP-14-123	SA-14-456	CA-14-456	X	SA-14-123
	SA-14-456	CP-14-456		X	SA-14-123
	CP-14-456	CP-14-789	CA-14-123	X	SA-14-123
	CP-14-789			P	SA-14-123

STEP 5: Map the grandparent to the supplementary card

Because we know what the primary card and grandparent is at this stage, we take the assigned grandparent account number and map it to the supplementary card within the card family for a cardholder.

```
-- 5. assign supplementary card based on fellow primary card
, cards_smapped as (
select distinct s.card_id
, s.TRANSFER_TO
, s.CONTROL_CARD
```

```
, coalesce(c.account_id, s.account_id, s.orig_card_id) as account_id
, s.CARD_TYPE
, s.account_level
, s.latest_card
, s.orig_card_id
, s.latest_card_type
from cards_mapped s
left join cards_mapped c
on s.latest_card_type = 'supplementary'
and s.CONTROL_CARD = c.card_id)
```

For a scenario where the candidate account_id is the same across all rows for a scenario then we can consider the scenario to be **solved** and the **account_id** will eventually become the final account_id, although the code does not need a "solved" portion to limit any *additional* logic applied to it, the remainder of the code will **not** affect the final mapped account_id.

The following scenarios have been **solved** in this step:

- Scenario 4: primary and supplementary card
- Scenario 6: supplementary card was lost/stolen
- Scenario 11: standalone card is lost/stolen and later converts to a primary card, and it too is lost/stolen, and the supplementary card is lost/stolen as well
- Scenario 15: primary is lost/stolen, becomes standalone, becomes primary again, which is then lost/stolen along with the supplementary card

	CARD_ID	TRANSFER_TO	CONTROL_CARD	ACCOUNT_ID	CARD_TYPE
primary card is the account id →	CA-4-321			CP-4-123	A
	CP-4-123		CA-4-321	CP-4-123	P
	CS-4-456		CA-4-321	CP-4-123	S
primary card is the account id →	CA-6-321			CP-6-123	A
	CP-6-123		CA-6-321	CP-6-123	P
	CS-6-456	CS-6-789	CA-6-321	CP-6-123	X
	CS-6-789		CA-6-321	CP-6-123	S
	CA-11-123	CA-11-456		SA-11-123	A
	CA-11-456			SA-11-123	A
standalone card is the account id →	SA-11-123	SA-11-456		SA-11-123	X
	SA-11-456	CP-11-123		SA-11-123	X
	CP-11-123	CP-11-456	CA-11-123	SA-11-123	X
	CP-11-456		CA-11-456	SA-11-123	P
	CS-11-123	CS-11-456	CA-11-123	SA-11-123	X
	CS-11-456		CA-11-456	SA-11-123	S
	CA-15-123	CA-15-456		CP-15-123	A
	CA-15-456	CA-15-789		CP-15-123	A
	CA-15-789			CP-15-123	A
primary card is the account id →	CP-15-123	CP-15-456	CA-15-123	CP-15-123	X
	CP-15-456	SA-15-123	CA-15-456	CP-15-123	X
	SA-15-123	CP-15-789		CP-15-123	X
	CP-15-789	CP-15-012	CA-15-456	CP-15-123	X
	CP-15-012		CA-15-789	CP-15-123	P
	CS-15-123	CS-15-456	CA-15-456	CP-15-123	X
	CS-15-456		CA-15-789	CP-15-123	S

STEP 6: Exception handling

Based on the above business rules, if a card ends up with two grandparent account numbers, pick the standalone card number over the primary card number.

```
-- exceptions --- when a card is assigned two account ids=primary
and standalone, pick standalone as the account
, dup_accfix as (
select distinct
 s.card_id as sa_account_id
, p.account_id as cp_account_id
from cards_smapped a
inner join (select card_id
          from cards_smapped
          group by card_id
          having count(account_id) > 1) e
on a.card_id=e.card_id
-- fetch assigned card-account type
inner join cards_smapped s
on a.account_id=s.card_id
and s.latest_card_type='standalone'
inner join cards_smapped p
on e.card_id=p.card_id
inner join cards_smapped p2
on p.account_id=p2.card_id
and p2.latest_card_type='primary'
)

-- 6 deal with exception
, cards_fmapped as (
select distinct s.card_id
, s.TRANSFER_TO
, s.CONTROL_CARD
, coalesce(c.sa_account_id, s.account_id) as account_id
, s.CARD_TYPE
, s.account_level
, s.latest_card
, s.orig_card_id
, s.latest_card_type
from cards_smapped s
left join dup_accfix c
on s.account_id = c.cp_account_id
)
```

This solves for scenario 13: standalone, primary, and supplementary cards issued with a controlling card

CARD_ID	TRANSFER_TO	CONTROL_CARD	ACCOUNT_ID	CARD_TYPE
CA-13-123			SA-13-123	A
CP-13-123		CA-13-123	SA-13-123	P
CS-13-123		CA-13-123	SA-13-123	S
SA-13-123		CA-13-123	SA-13-123	

standalone card is the account id →

STEP 7: Bring it all together

```
, final_table as (
```

```
-- 6. map control card
select p.*
, coalesce(map_latest_account.latest_card, p.account_id) as control_card_id
from cards_fmapped p
left join (
select distinct 'A' as c, account_id
, CARD_TYPE
, latest_card
, latest_card_type
from cards_fmapped A
where CARD_TYPE = 'A'
  and not exists (select 1
                  from cards_fmapped c
                  where a.account_id=c.account_id
                  and c.latest_card_type='standalone')
union all
select distinct 'B' as c, account_id
, CARD_TYPE
, latest_card
, latest_card_type
from cards_mapped
where TRANSFER_TO is null and CARD_TYPE is null
) map_latest_account
on p.account_id=map_latest_account.account_id
)
```

The outcome is a staged content that is ready to be ingested using the same link table loader used for raw vault to load to a business vault link table. Because a link table is about the unique list of relationships only two types of relationships are loaded:

- two records when a card is lost/stolen or switches products
    - 1st record is the card being **terminated**
    - 2nd record is the **new card** (unless the account itself is closed)
- a single record when a new standalone card is used or three records when a new family product of cards are issued (primary + supplementary + controlling card).

Superseded records are never deleted because they show the historical movement of cards that at a point in time, they were active. Essentially the business keys are *mapped* to the business vault link and should add nothing to the hub table, because they had already been loaded in the raw vault link table loads.

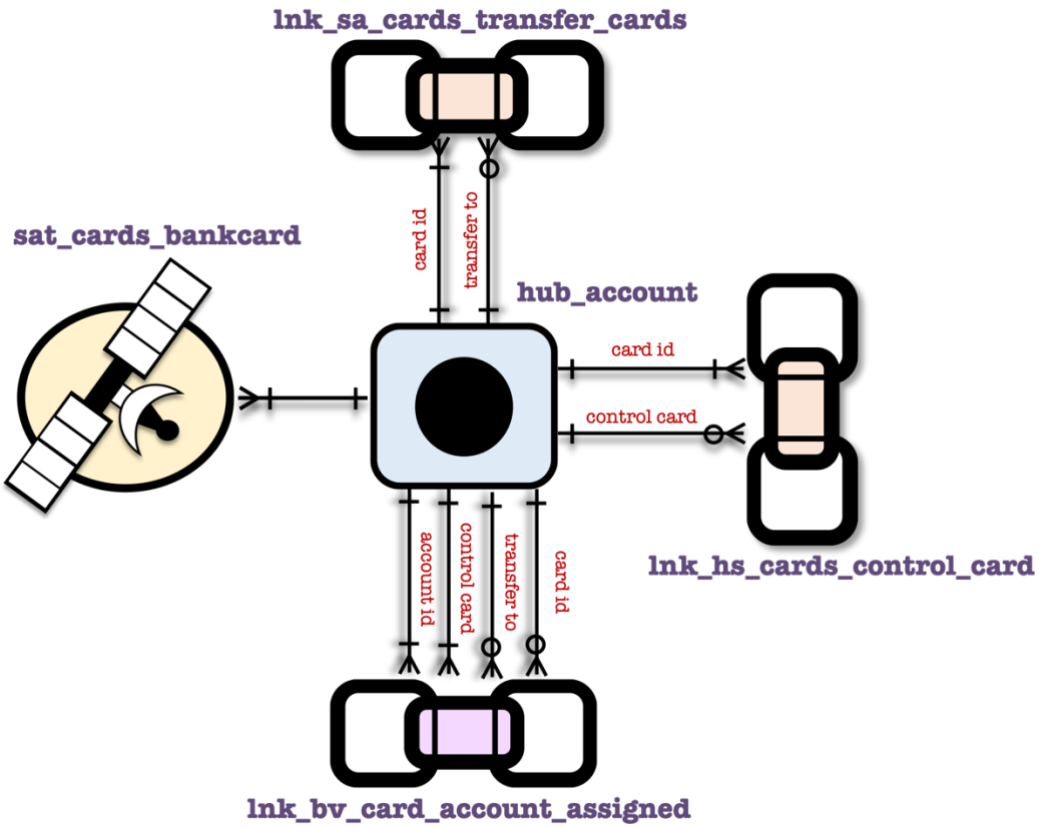


Figure 21 Raw Vault extended with a Business Vault Link table

The information mart(s) grabbing each card from the business vault link must then always retrieve the latest card in the link to get its current state, and there is only two:

- it's a new card.
- The card has been superseded (transferred).

What we have now is a single business vault link that consolidates the raw vault link structures and has an additional column that is used to retrieve what is the assigned account_id that is selected out of the hub table's card numbers. Equi-join is used when retrieving the data and cards that are active have a null transferred_to field which maps to the single zero key record in the hub table. The complexity has been taken care of in the data vault build itself there can be minimal ambiguity on how to retrieve the data you want from the data model.

How to retrieve the account_id… note how we only need one business vault link and equi-join to the same hub table as many times as we need. No recursive query needed, simply use the BV-link for all your card account needs!

```
select hub.account_id as card_id
, case when hub2.account_id ='-1' then null else hub2.account_id end as transferred_to_card_id
, case when hub3.account_id ='-1' then null else hub3.account_id end as control_card
, case when hub4.account_id ='-1' then null else hub4.account_id end as account_id
, sat. card_type
from hub_account hub
inner join lnk_bv_card_account_assigned lnk
on hub.dv_hkey_hub_account=lnk.dv_hkey_hub_account
inner join hub_account hub2
on lnk.dv_hkey_hub_account_transfer_card = hub2.dv_hkey_hub_account
inner join hub_account hub3
```

```
on lnk.dv_hkey_hub_account_control_card = hub3.dv_hkey_hub_account
inner join hub_account hub4
on lnk.dv_hkey_hub_account_assigned_account = hub4.dv_hkey_hub_account
inner join (select s.dv_hkey_hub_account, s.card_type
        from sat_cards_bankcard s
            inner join (select dv_hkey_hub_account, max(dv_rtts) as dv_rtts, max(dv_ldts) as dv_ldts
                    from sat_cards_bankcard
                        group by dv_hkey_hub_account) sm
            on s.dv_hkey_hub_account=sm.dv_hkey_hub_account
            and s.dv_ldts=sm.dv_ldts
            and s.dv_rtts=sm.dv_rtts) sat
on hub.dv_hkey_hub_account=sat.dv_hkey_hub_account
order by scenario, card_id
;
```

Why not build a Bridge table?

Bridge tables are single purpose, designed to shorten model distances and are designed to be **disposable**. Bridge tables are for query assistance whereas a business vault link is designed to support **auditability**, **agility**, and **automation** just like the rest of raw and business vault.

Business vault links are also designed to support complex business rules as the above has shown, if the business rule should evolve then we would consider deprecating the existing business vault link and creating a new one if the change **is significant**, or simply continue to append to the BV link if it is a **minor change**, the change should be captured in the record source column as a version number.

My advice on this complexity is to solve it at the source, but it may turn out that for the reasons mentioned in the first paragraph of this article, solving this at the source system vendor is neither feasible nor desirable and hence will not be solved at the source. And hence we are left to solve this **technical debt** in the business vault. We must limit these use cases in the business vault, in the **ideal** data vault there would be **no business vault**, but alas we are left to deal with **operational reality**.

But here is the point of the article, and what you **must** think about when you're building your data vault and should lead to you building successful data vaults, solve this complexity for your users upfront. Imagine if every business user needed to understand the complexity presented in this article. They *shouldn't*. Like every scalable, single source of fact system should provide, data vault solves this complexity for the business user.

I am an advocate for *less* tables in a data vault, and the way we do it is by solving the complexity for the user by reusing repeatable patterns. If I didn't solve this in an auditable business vault, we would otherwise expect the business users to execute this code every time they need an account number!

Solve it once and your solution will scale!

Until next time…

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.