In today's episode of Data Vault Mysteries we demystify Zero Keys and Ghost Records!

The two have been mentioned in literature and often their purpose has defined interchangeably; however they are in Data Vault 2.0 two very different concepts serving two very distinct purposes! Join me in the classroom as we unveil what each are and how they should be thought of in the context of data modelling! First up…

Ghost Records

A ghost record has no value, no meaning, it is a ghost! A single ghost record is inserted into a satellite table right after the table is created and forgotten about. When running a query between a hub table and its satellite or a link table and its satellite no ghost record is returned; no ghost will exist in the hub or link. So why then, do we insert a ghost record into a satellite table and simply forget about it? The answer is simple if you understand database table indexes and SQL join conditions. There are several ways you can join two relational tables together,

- LEFT JOIN returns all records from the left table (table1), and the matched records from the right table (table2). The result is NULL from the right side, if there is no match
- RIGHT JOIN returns all records from the right table (table2), and the matched records from the left table (table1). The result is NULL from the left side, when there is no match
- EQUI or INNER JOIN selects records that have matching values in both tables

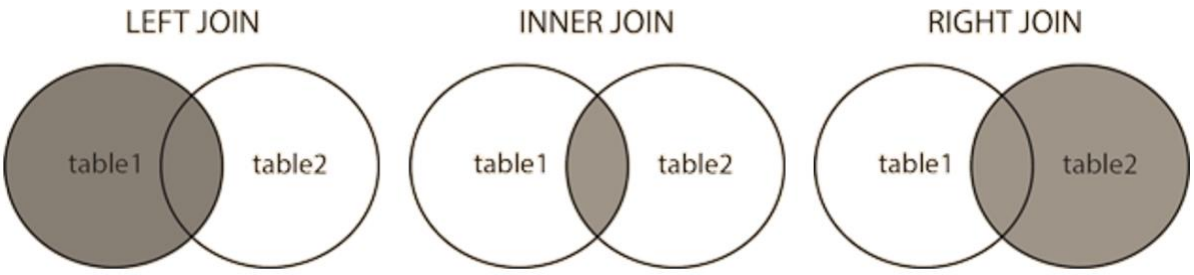


Figure 0-1 Learn more by visiting https://www.w3schools.com/sql/sql_join.asp

Performing joins of two tables together without an index forces the SQL query optimizer to perform a full scan of the contents of each table to return which record meets the join condition you chose. Now if both tables included in the join condition are indexed then the SQL optimizer knows where to find the records to join rapidly! For those non-matched records in a LEFT JOIN, the table on the LEFT retains all the records and the record attributes from the right table that do not match are not returned. A much faster query is if we return all records that match, see bit.ly/3knWleo.
Ok, now that we understand JOIN conditions let's see how it pertains to Data Vault. A hub table will contain the unique list of business keys from whatever source supplied it. It could have come from two different types of source data files:

a) A file with the business key and its descriptive contents
b) A file with many business keys and its descriptive contents

Under both types the descriptive content is subject to satellite splitting, under (a) a hub key can be associated with zero or more hub-satellite tables and under (b) a hub key can be associated with zero or more satellite tables and one or more link tables. Under (b) a link can also be associated with one or more hub tables and zero or more link-satellite tables. It all depends on the content and what the descriptors in the landed and staged file describes!
Note: that we also split by confidential data, rate of change, maximum platform table width, etc.

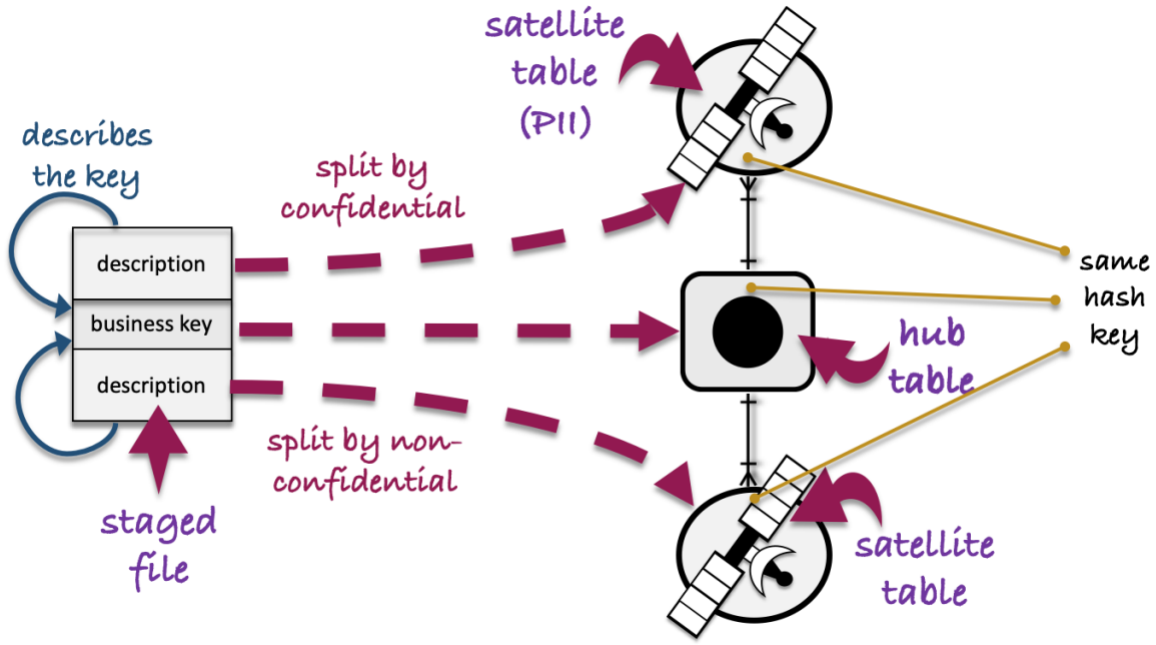


Figure 0-2 File type (a) with satellite splitting

So if you wanted to return the appropriate record descriptor records, relationship and keys for a particular business entity or relationship an EQUI-JOIN between all the populated data vault artefact returns everything we know about that business entity or relationship, and nothing else. So far so good, no?

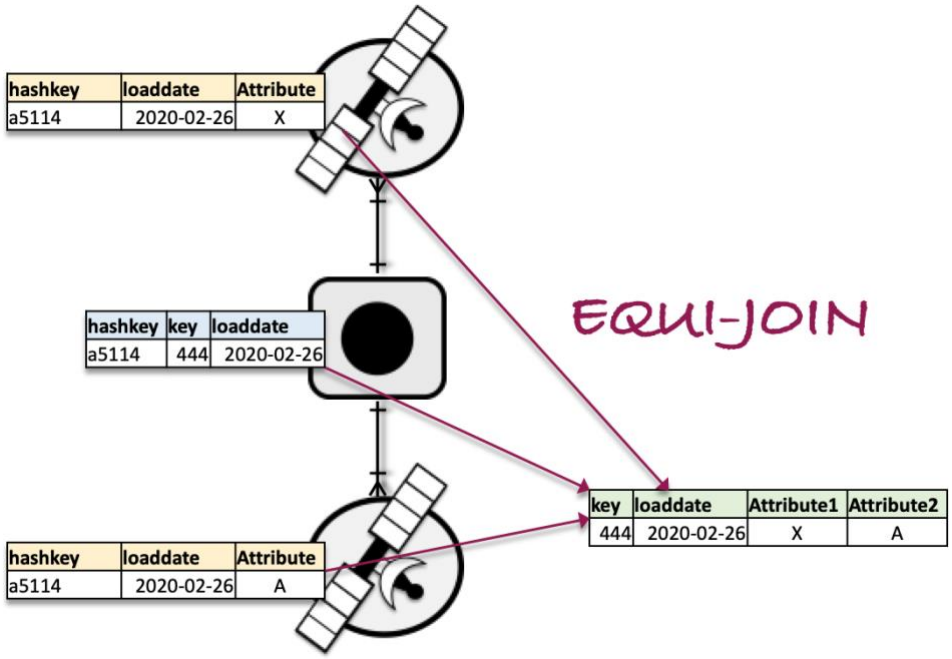


Figure 0-3 EQUI-JOIN from a single source

What if we get data about a business key from multiple source files and in some instances we don't get a record for a business key just yet?

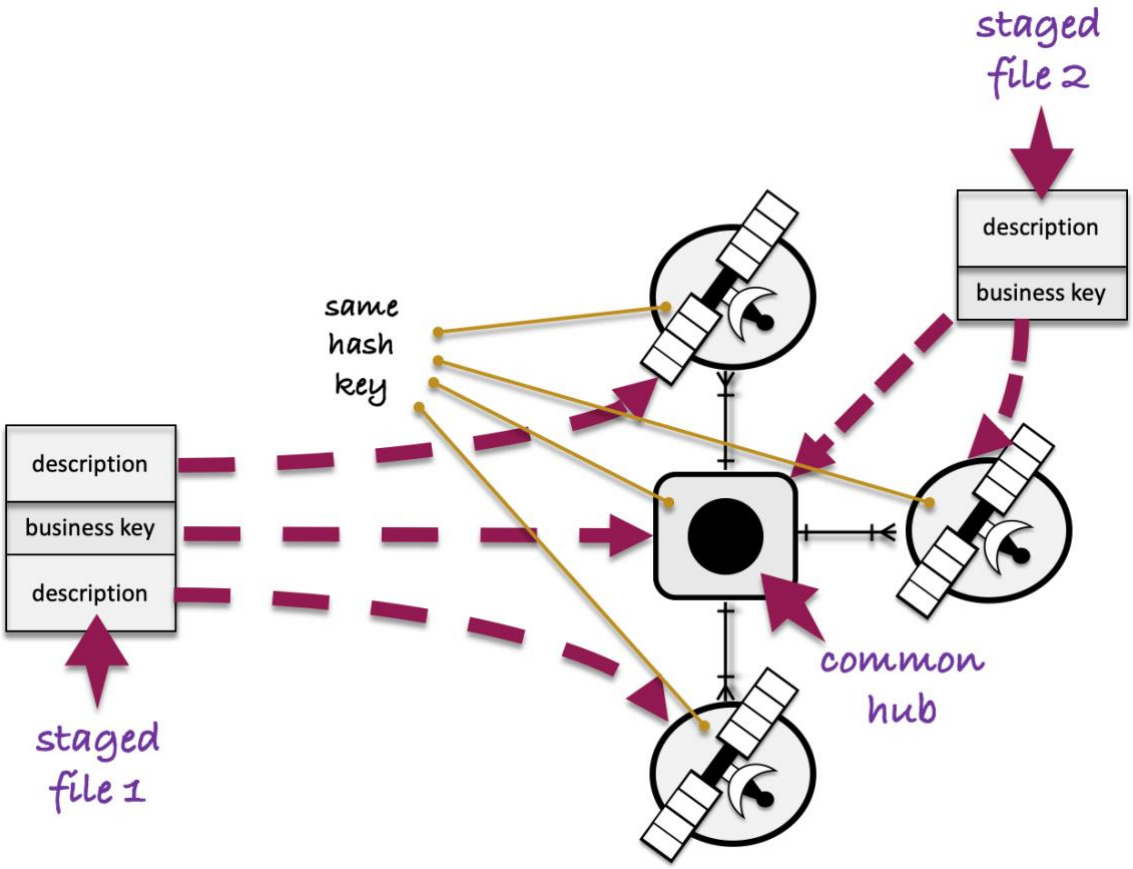


Figure 0-4 Common hub, with multiple sources

Now if we were to perform an EQUI-join from multiple source files with the same keys but the data arrives at different times then the following would occur

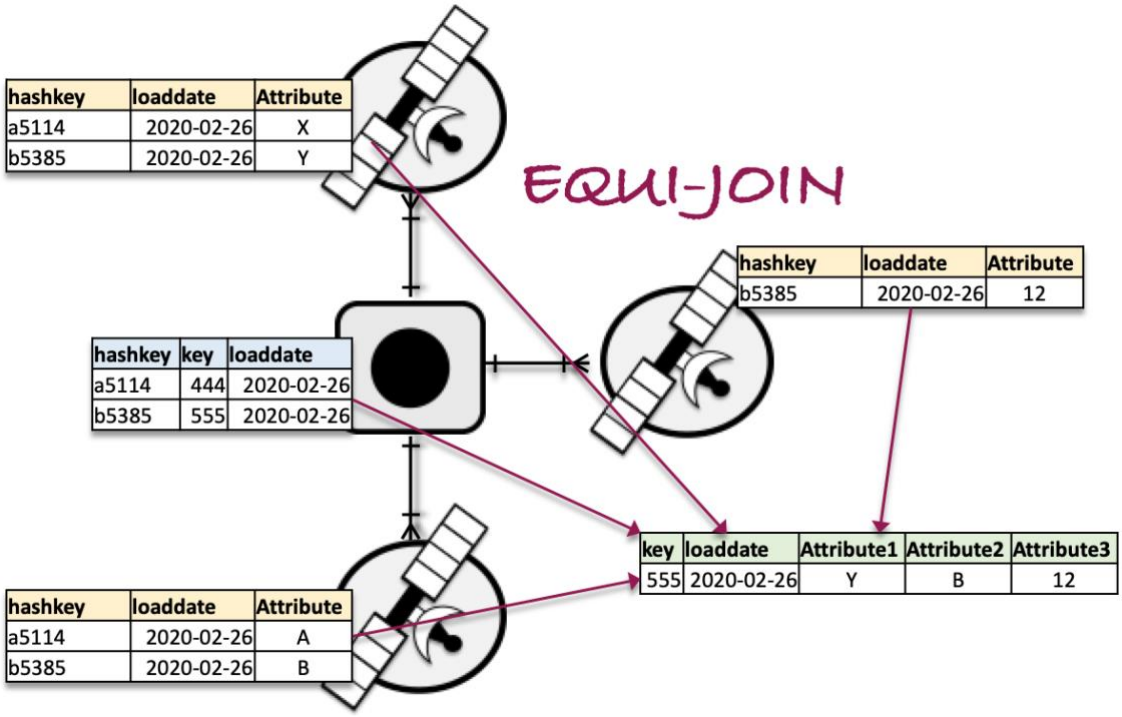
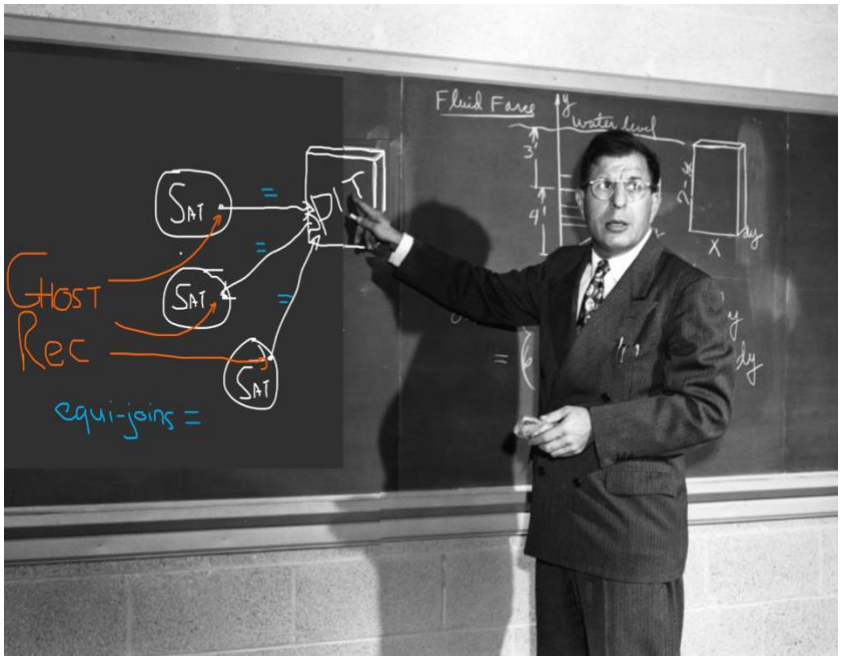


Figure 0-5 Now you'd need an EQUI and LEFT join so you do not discard records!

We haven't forgotten about the *GHOST!* By relying on EQUI-JOINs the business key or relationship without a record in an adjacent satellite then those particular business keys or relationships will not return a record until the adjacent content does arrive. This might not be appropriate for your reporting requirement and thus we must rely on LEFT JOINs for particular source-table conditions! A switch architecture! This will not do! Let's talk about point-in-time (PIT) tables…



A PIT table is not a Data Vault construct per se… it is a method of pulling data from the data vault in an efficient and speedy manner and I'll explain how. As the data vault model flexes and grows as your organization's needs and business processes grow, the volume, velocity, veracity and volatility of your data grows as well. This growth comes at a price, how can we efficiently take snapshots of the data as it grows and still provide the analytics the business users crave? This is the design behind a PIT table. Taking a templated snapshot about everything we know about a business entity or relationship. To do this we construct a PIT by leveraging business reporting rules in accordance with the required outcome, in other words we take a picture of the active satellite keys for a parent key (hub or link), if there are no active records for a snapshot for a parent key then we return the ghost record in the satellite. You may ask when does the snapshot refer to the ghost record? When we know nothing about the last change for the satellite for that parent key. If no record arrived then it will always be the ghost record it refers to, when it does arrive then that is the latest active record we know about that parent key until it changes, and hence the PIT will point to the active record for that snapshot and not the ghost record. The construction of the PIT will use LEFT and EQUI joins where appropriate, but your reports will not. It will leverage the indexes as a foundation for the information mart view!

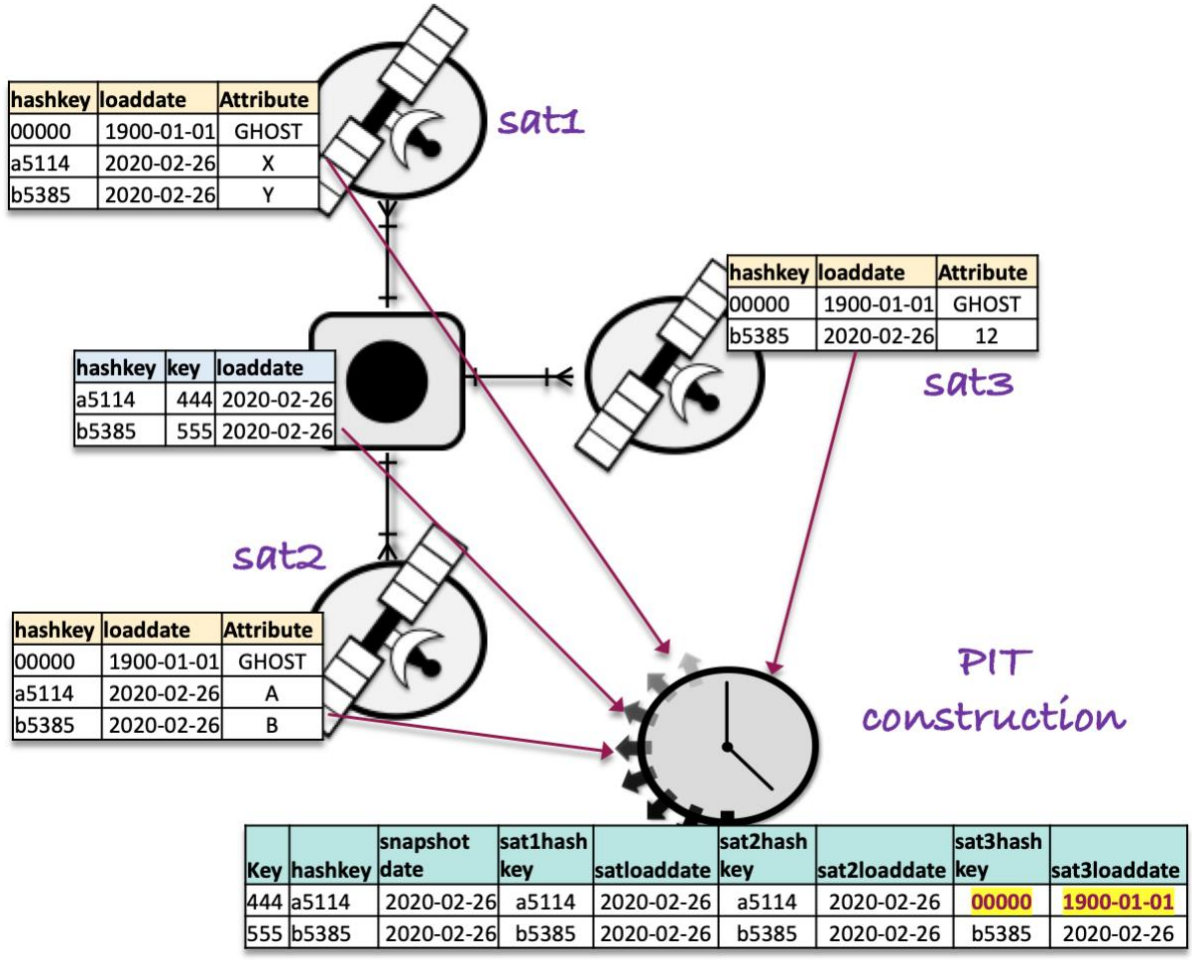


Figure 0-6 Building a PIT

Now the frequency of the snapshots is where the performance is gained; you may have intra-day, daily, weekly, monthly, quarterly or even yearly snapshot requirements. PITs are constructed as frequently as your business requirements need them to be. All they're doing is pointing to the indexed location of where to find the records you need for a business key or relationship according to the snapshot you need. Without a ghost record in each satellite table your PIT tables will not be performant, you must have both ingredients to make the PIT table efficiency work, a single ghost record is inserted in each satellite table to make EQUI-JOINS to the PIT tables work!
In essence what we have done with the model is this:

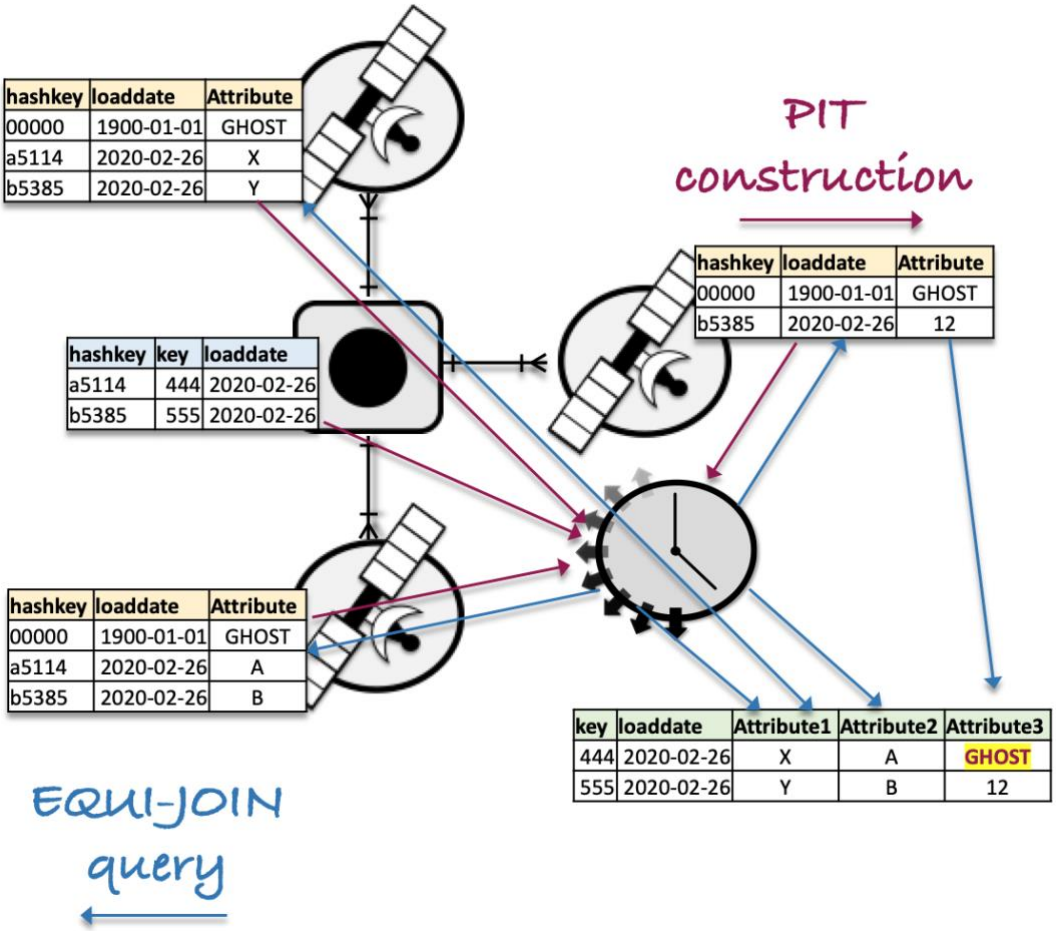


Figure 0-7 PIT query

Zero Keys

We have seen how Ghost records complete EQUI-JOINS between a PIT table and adjacent satellites tables. These techniques take advantage of the platform capabilities, it's why we use surrogate hash keys on an MPP platform as well for even hash distribution and on-disk locality (see: bit.ly/31LxYPU).

Zero keys solve a different problem, let's say data arrives and the business key is null. Is this a problem? Well that depends, first of all do indexes or primary keys accept null values? The answer is no. (see: bit.ly/3bHkzvW) A null value included in hashing can also break the hashing outcome, here's a quick demo on how this occurs:

```
62    select hashbytes('md5', null) as test
```

Messages

6:37:42 PM Started executing query at Line 63
Msg 8116, Level 16, State 1, Line 1
Argument data type NULL is invalid for argument 2 of hashbytes function.
Total execution time: 00:00:00.016

Figure 0-1 Breaking the hash function in SQL Server

In order to circumvent both issues a Zero Key is coalesced in the staging of the landed file for business keys, we turn a null into '-1'. Why '-1'? There is little chance that a business key will ever be a negative value, the business key column is populated and when the hub-loader loads the staged content the hub table will then contain the zero key, no pre-loading of zero keys required and no functional relationship to PIT tables!
But defining a zero key enables other functionality as well! Let's say a landed file has multiple business keys depicting a unit of work, participants in the unit of work can be optional, what do we do then? The same zero-key treatment is applied! Thus the link table containing this unit of work includes the mandatory and optional portions of the relationship!

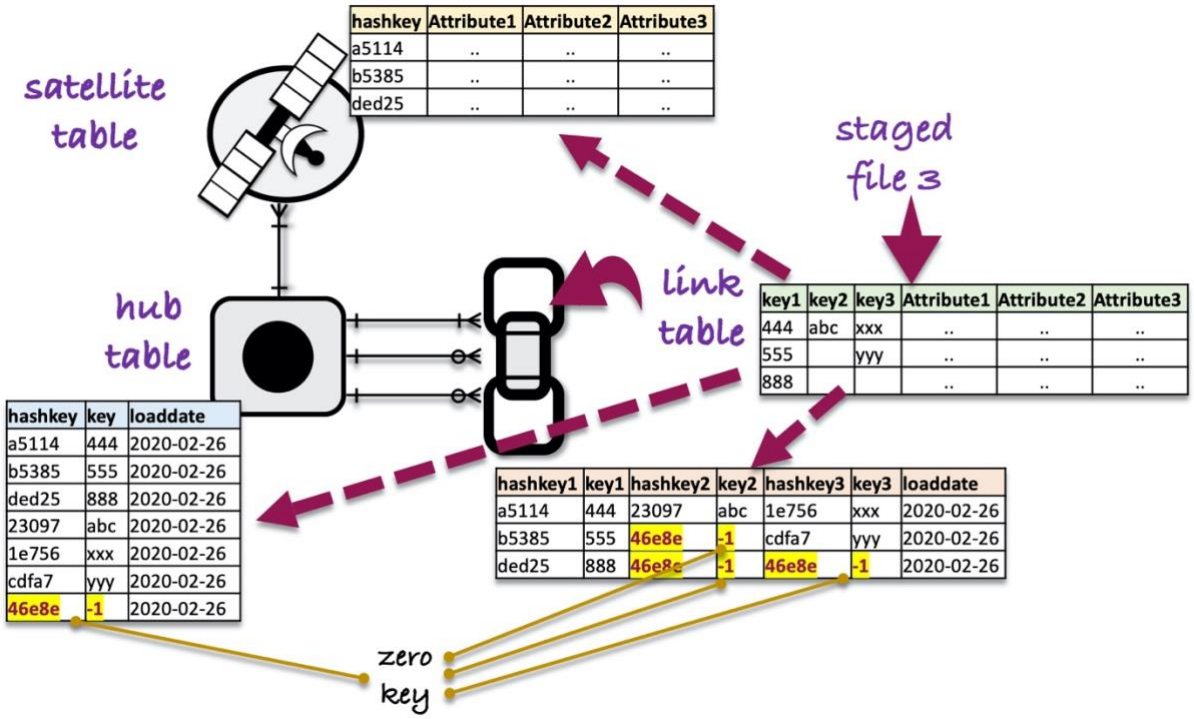
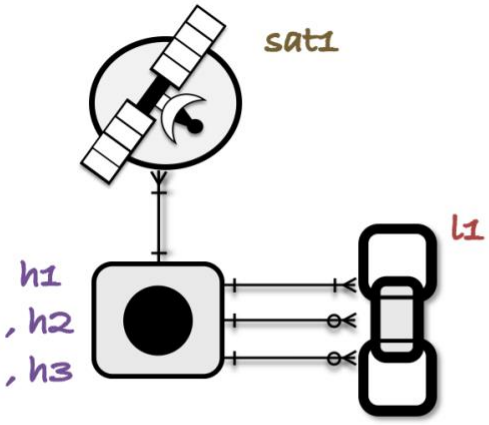


Figure 0-2 Zero key occurs naturally in hub

When designing a query around the hub, link and satellite tables (even though there is an optional portion of the relationship); the query will again be an EQUI-join because regardless of the optional portion being populated or not the surrounding link table record will find something to join to in the hub table, the zero-key! Similar to what Ghost records do for PIT table joins, the zero-key does for optional link table joins!

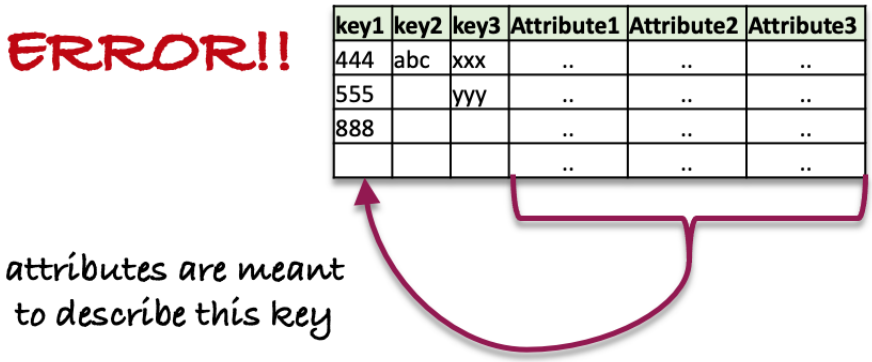



```
select h1.key as businesskey1
, case when h2.key = '-1' then null
  else h2.key end as businesskey2
, case when h3.key = '-1' then null
  else h3.key end as businesskey3
, sat1.*
from link l1
inner join hub h1
on l1.hashkey1 = h1.hashkey
inner join hub h2
on l1.hashkey2 = h2.hashkey
inner join hub h3
on l1.hashkey3 = h3.hashkey
inner join sat1
on h1.hashkey = sat1.hashkey
```

businesskey1	businesskey2	businesskey3	Attribute1	Attribute2	Attribute3
444	abc	xxx	..	..	..
555		yyy	..	..	..
888			..	..	..

Figure 0-3 Zero key enables EQUI-join

There is an exception however, what if there is descriptive information describing only that business key, the relationship participant. How can we have descriptive content tied to what essentially is a null value? You can't! That is an error in either the data vault model designed or the loading of the content to the data vault model. In either circumstance the load must be stopped and investigated!



key1	key2	key3	Attribute1	Attribute2	Attribute3
444	abc	xxx	..	..	..
555		yyy	..	..	..
888			..	..	..
			..	..	..

Figure 0-4 Populated attributes must have a parent key, through our modelling Attri1-3 describes key1

You cannot have descriptive content describing nothing!
Now you might be saying, "that's great Patrick, but Snowflake has no indexes so I don't need Zero-keys". On the contrary, a null business key loaded into the data vault might be for host of reasons, if however you control this behaviour upfront, you can rule out that the modelling or the tooling let a null business key through. It simplifies trouble shooting if you see this in your data. Then again you can also apply the 'not null' constraint on the column in Snowflake and because we have substituted nulls with Zero keys we still keep the functionality we get that is illustrated above.

Default Keys

Strictly in the Kimball modelling space (facts & dimensions) default keys are pre-loaded to a dimension table for a very different purpose. According to Kimball design tip #128 (see: bit.ly/2Nkzn7W) a slowly changing dimension is pre-loaded with negative keys.

Key	Description
-1	Missing Value
-2	Not Happened Yet
-3	Bad Value
-4	Not Applicable

In construction of your fact table, if the measure falls in any of those categories of data listed above then the foreign key loaded to the fact table will be one of those pre-loaded keys. What that does is when a star-join optimized query is run against that star schema it will return the dimension and fact content selected and one of those default value as determined by your reporting rules. Unlike data vault, the default keys here are strictly used for reporting! Still the same EQUI-join query pattern is used and database platforms with OLAP functionality recognizes this type of query and optimally returns the data from this dimension – fact table configuration. See: bit.ly/2O0pUXh

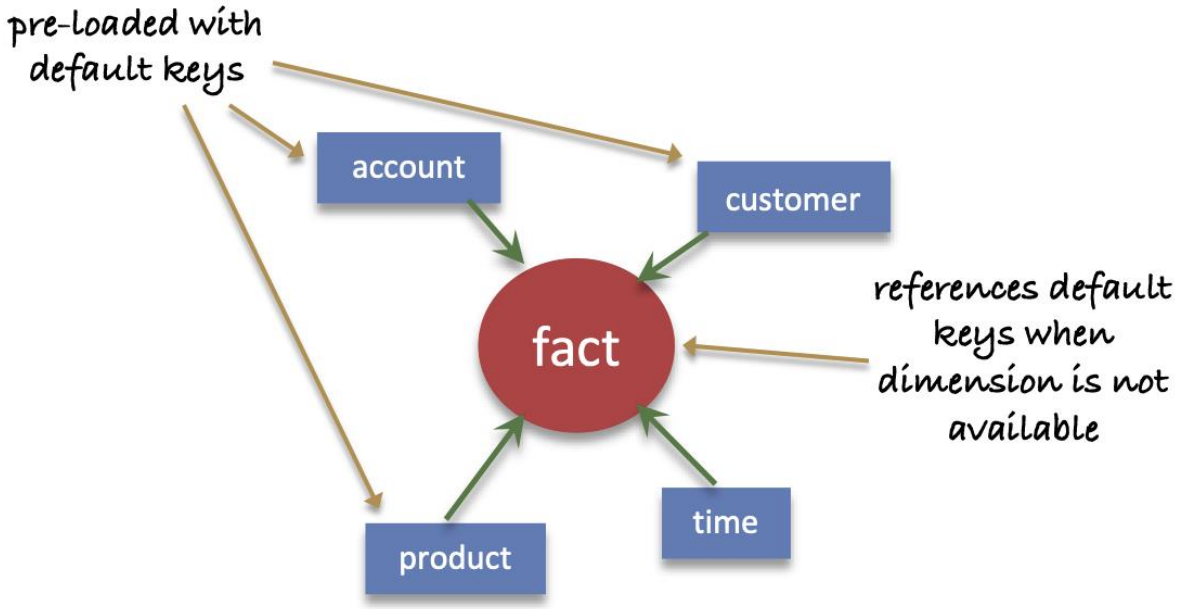


Figure 0-1 Default keys in dimensional modelling

This concept is not useful to data vault because it is not intended to be an information layer business intelligence (BI) tools are accustomed to using. These BI tools will use the information marts built over a data vault and not directly query it!

Summary

The difference in mindset between dimensional modelling and data vault is never so clear; dimensional modelling focuses on the business reporting outcomes whereas data vault moulds to the platform environment, technology and business processes to build a data warehouse. The focus of either is so different including its implementation that it has never been so plain to see in its definitions of default keys; what I mean is: the data vault is designed to build a data warehouse and Kimball modelling is designed around an information mart, let that sink in, data warehouse versus and information mart. An information mart conforms data to suit the business need right now, a data vault integrates data sources through agility, automation and auditability as a base to deliver conformed information marts; the two solve two different purposes in a data warehouse. Data vault does not replace the dimensional

mart, but rather it complements it, and as the methodology is used as the base it is the audit history, it is non-destructive to change and thus it implies that the information mart layer has become disposable.