Paper 1898-2018

Integrating SAS and Data Vault

Patrick Cuba, Cuba BI Consulting Pty Ltd

ABSTRACT

Data Vault (DV) modelling technique is fast gaining popularity around the world as an easy to learn, easy to manage, easy to maintain and easy to adapt (to business change) data modelling technique. In this paper we will briefly explore what DV is; DV artifacts and we will explore how SAS can be used to **automate** its data loading patterns.

INTRODUCTION

DV was described by Dan Linstedt in 1990 in a series of articles dubbed *"Common Foundational Integration Modelling Architecture"* – the original name for DV. The term he coined for the modelling approach was *"all the data, all the time"* and *"single version of facts"*. Essentially DV is intended to capture the data in a raw format and load the data as quickly as possible from multiple data sources. DV is a **temporal** data store and it is designed to be flexible to change and be able to accept batch or real time data. The data can be structured or semi-structured and the technique is designed to **scale**. DV is an amalgamation of two common **Data Warehouse** modelling techniques –**Dimensional (Star Schema) Modelling (Kimball)** and **Relational Modelling (3NF)**. Bill Inmon has endorsed DV; stating "The Data Vault is the optimal choice for modeling the EDW in the DW 2.0 framework."

WHAT IS DATA VAULT MADE UP OF?

DV captures business concepts into **hubs** represented by a unique **business key (BK)** and assigned a **hub key (HK)**. **Satellites** expand the hubs with **temporal** data attributes from source data relating to that BK. **Links** are used to join hubs together – they denote the relationships between BKs**.**

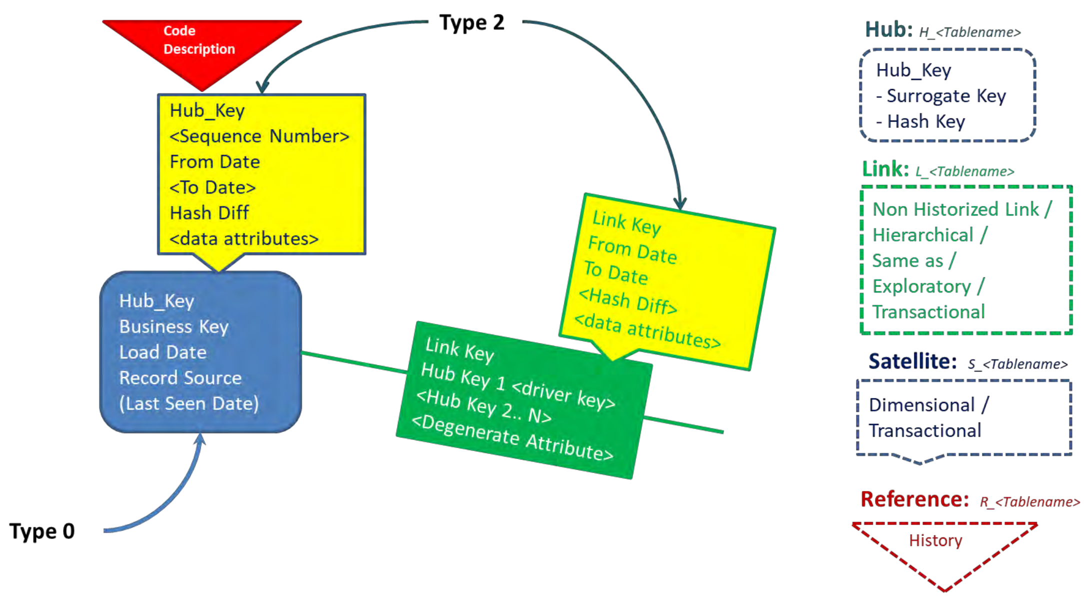


Figure 1 Data Vault entity attributes, bracketed columns are optional

The tables have a default set of columns and a uniform structure. In the next figure we have prepopulated the above DV structures with sample data; HK (H_PERSON_K) has a 1-to-1 relationship to the BK (SOCIAL_SEC_NUM); HKs link to multiple versions of the HK recorded in the satellite table. Changes in details relating to HKs are loaded as new versions by end-dating the existing version and inserting a new version with a high date (31-Dec-9999). A link records the relationship of H_PERSON to the H_ADDRESS by HKs and that relationship's status is recorded in a link satellite table.

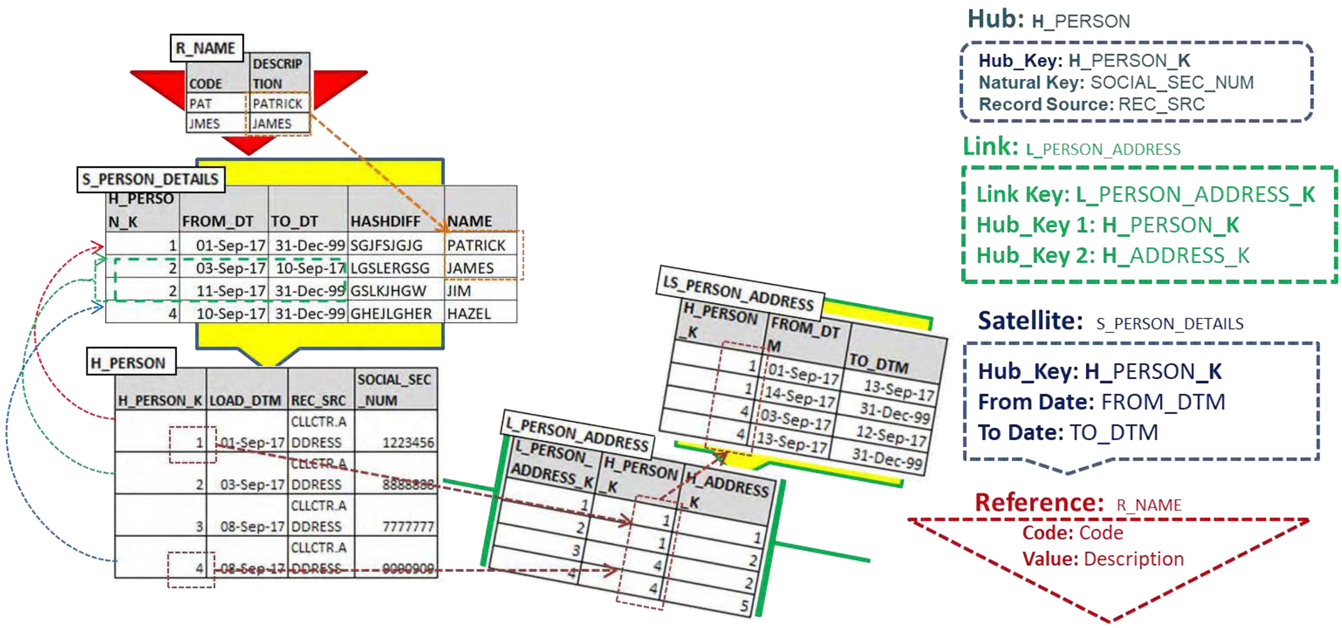


Figure 2 Data Vault example using surrogate keys (SK)

AUTOMATING DATA VAULT WITH SAS

With the limited types of DV tables we can generate automated code to populate those tables. Here we will introduce a 4-step process to automate the population of DV tables using SAS that scales.

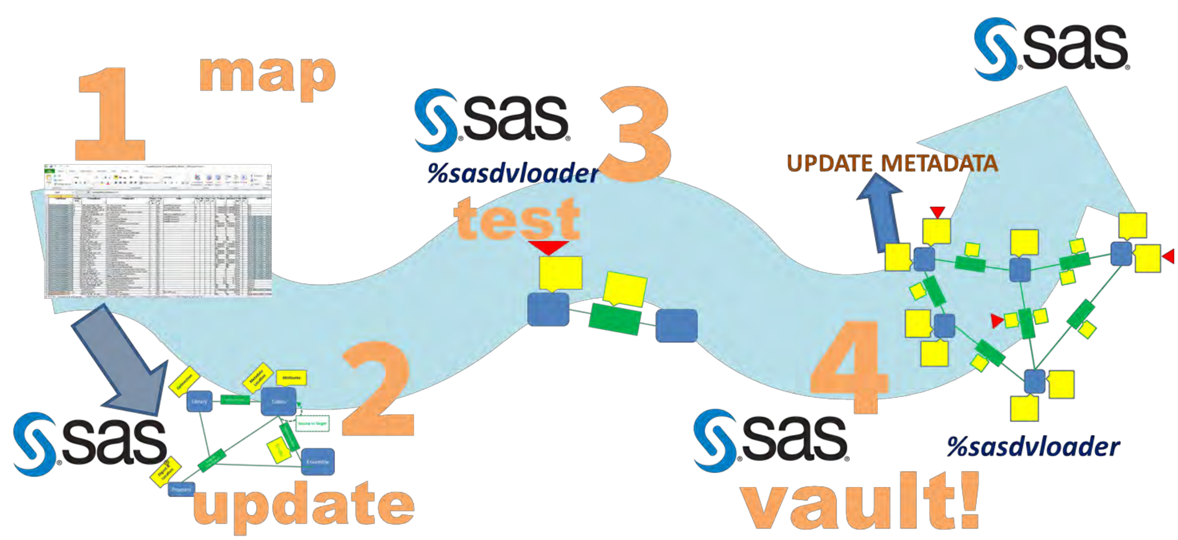


Figure 3 4-Step process to automate Data Vault with SAS

1 - MAP

We use spreadsheets to map what our DV will look like. We can think of mapping DV in terms of **ensembles** that can be made up of a collection of hubs, links and satellites. This will allow us to define multiple sources to populate a particular DV entity; we can see what I mean in the next few sections.

Hubs and Satellites – the business entities and their history

mapping hubs

SOURCE LIBRARY	SOURCE TABLE	TARGET LIBRARY	TARGET HUB	TARGET HUB KEY
CLLCTR	ADDRESS	DV_CIS	H_PERSON	H_PERSON_K
CLLCTR	PERSON	DV_CIS	H_PERSON	H_PERSON_K
CLLCTR	ADDRESS	DV_CIS	H_COMPANY	H_COMPANY_K
CLLCTR	COMPANY	DV_CIS	H_COMPANY	H_COMPANY_K
CLLCTR	ADDRESS	DV_CIS	H_ADDRESS	H_ADDRESS_K
ONLWB	ACCOUNT	DV_CIS	H_ACCOUNT	H_ACCOUNT_K
WIRE1	TXN	DV_CIS	H_ACCOUNT	H_ACCOUNT_K
WIRE1	TXN	DV_CIS	H_ACCOUNT	H_ACCOUNT_K
SALES	PRODUCTS	DV_CIS	H_PRODUCTS	H_PRODUCT_K
CORE	TRANS	DV_CIS	H_TXNS	H_TXN_K

nsembles Libraries Hubs Satellites Links

mapping satellites

SOURCE TABLE	FILTER	TARGET LIBRARY	TARGET HUB	SATELLITE TARGET TABLE	FROM DATE	TO DATE
ADDRESS	ADDRESS_TYPE='POSTAL'	DV_CIS	H_PERSON	S_ADDRESS_POSTAL	DATETIME	DATETIME
ADDRESS	ADDRESS_TYPE='PHYSICAL'	DV_CIS	H_PERSON	S_ADDRESS_PHYSICAL	DATETIME	DATETIME
PERSON		DV_CIS	H_PERSON	S_PERSON_DETAILS	DATETIME	DATETIME
ADDRESS	ADDRESS_TYPE='HEAD OFFICE'	DV_CIS	H_COMPANY	S_ADDRESS_HOFFICE	DATETIME	DATETIME
ADDRESS	ADDRESS_TYPE='BRANCH'	DV_CIS	H_COMPANY	S_ADDRESS_BRANCH	DATETIME	DATETIME
COMPANY		DV_CIS	H_COMPANY	S_COMPANY_DETALS	DATETIME	DATETIME
ADDRESS		DV_CIS	H_ADDRESS			
ACCOUNT		DV_CIS	H_ACCOUNT	S_ACCOUNT_ONLWB_APPDETAILS	DATETIME	DATETIME
ACCOUNT		DV_CIS	H_ACCOUNT	S_ACCOUNT_ONLWB_BALANCES	DATETIME	DATETIME
TXN		DV_CIS	H_ACCOUNT	S_ACCOUNT_WIRE1_DETAILS	DATETIME	DATETIME
TXN		DV_CIS	H_ACCOUNT	S_ACCOUNT_WIRE1_TXN	DATETIME	
PRODUCTS		DV_CIS	H_PRODUCTS	S_PRODUCT_DETAILS	DATETIME	DATETIME
TRANS		DV_CIS	H_TXNS	S_BANK_CORE_TXNS	DATETIME	

bles Libraries Hubs Satellites Links Attributes DV Column Typ

Figure 4 Mapping hubs & satellites using spreadsheets

In **red** we have mapped the ADDRESS source table to 3 hubs and 4 satellites (each satellite is filtered by ADDRESS_TYPE) and we have stipulated that we track changes as **datetime**. H_PERSON will have physical addresses and/or postal addresses and H_COMPANY will have a head office address and/or branch addresses. The 3rd hub H_ADDRESS is mapped without any satellites.

In **green** we have mapped multiple sources to a single H_ACCOUNT hub because each source provides the same account number unique business key. We get the account number from the ACCOUNT and TXN (transactions) source tables. H_ACCOUNT is mapped with four satellite tables. Three of the satellites are **Type 2** tables as we have mapped values for both 'FROM_DATE' and 'TO_DATE', the last satellite called 'S_Account_Wires_Txn' does not have 'TO_DATE' mapped and it will be treated as a **transactional** satellite. It is used to record transactions as transactions are point in time facts and do not persist over time.

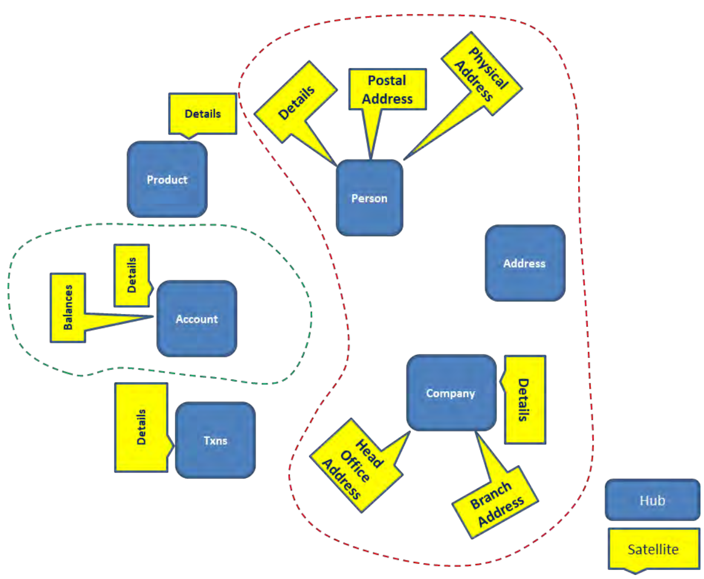
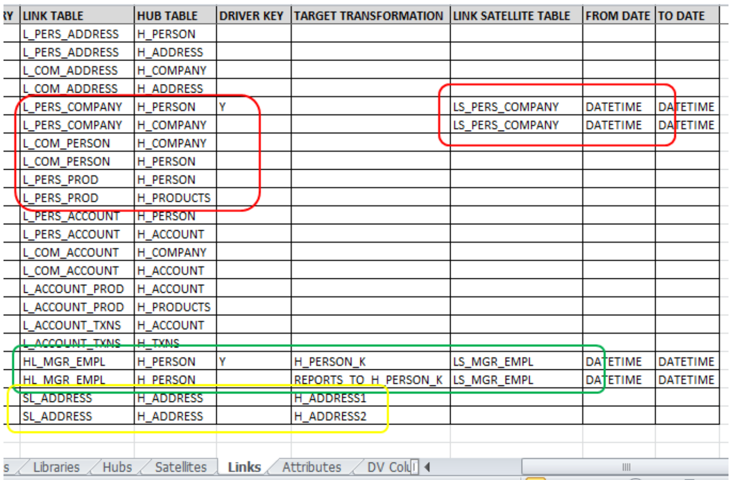


Figure 5 Hubs & satellites and their ensembles

Links – the relationships between business entities



RY	LINK TABLE	HUB TABLE	DRIVER KEY	TARGET TRANSFORMATION	LINK SATELLITE TABLE	FROM DATE	TO DATE
	L_PERS_ADDRESS	H_PERSON					
	L_PERS_ADDRESS	H_ADDRESS					
	L_COM_ADDRESS	H_COMPANY					
	L_COM_ADDRESS	H_ADDRESS					
	L_PERS_COMPANY	H_PERSON	Y		LS_PERS_COMPANY	DATETIME	DATETIME
	L_PERS_COMPANY	H_COMPANY			LS_PERS_COMPANY	DATETIME	DATETIME
	L_COM_PERSON	H_COMPANY					
	L_COM_PERSON	H_PERSON					
	L_PERS_PROD	H_PERSON					
	L_PERS_PROD	H_PRODUCTS					
	L_PERS_ACCOUNT	H_PERSON					
	L_PERS_ACCOUNT	H_ACCOUNT					
	L_COM_ACCOUNT	H_COMPANY					
	L_COM_ACCOUNT	H_ACCOUNT					
	L_ACCOUNT_PROD	H_ACCOUNT					
	L_ACCOUNT_PROD	H_PRODUCTS					
	L_ACCOUNT_TXNS	H_ACCOUNT					
	L_ACCOUNT_TXNS	H_TXNS					
	HL_MGR_EMPL	H_PERSON	Y	H_PERSON_K	LS_MGR_EMPL	DATETIME	DATETIME
	HL_MGR_EMPL	H_PERSON		REPORTS_TO_H_PERSON_K	LS_MGR_EMPL	DATETIME	DATETIME
	SL_ADDRESS	H_ADDRESS		H_ADDRESS1			
	SL_ADDRESS	H_ADDRESS		H_ADDRESS2			

Figure 6 Mapping links

In **red** we have mapped our hubs to links and notice we have a two-way link recorded in separate link tables – L_PERS_COMPANY and L_COM_PERSON. We assumed that a company can have many employees but an employee can only belong to one company. And for that reason we have a **driver key** in the "Person to Company" link table. If a person changes jobs the Person entity drives the change in the relationship. We record the change in the link-satellite (LS_PERS_COMPANY) table by tracking the change in the FROM_DATE and TO_DATE columns.

In **green** we have mapped a **hierarchical link (HLINK)** between manager and employee. The hierarchical link (HL_MGR_EMPL) is between the same hub – *person to person* and we cannot have the same hub key name appear in the same table and therefore one of the hub keys will need to be renamed. HLINK will contain the columns "H_Person_K" and "Report_to_H_Person_K".

In yellow we have mapped a **"same-as"** link (SLINK - SL_ADDRESS); this is the result of **data quality (DQ)** rules (such as *fuzzy matching*) that have mapped similar addresses together. Data entry errors can be the result of sticky fingers or lack of clearly defined domains for such things as differently spelt street names or different address lines used differently. For example, the word "Street" can be depicted at "St" or "Str". Let's say we are capturing addresses in South Africa where in English you capture the address as "23 George Street" but in Afrikaans you capture that same address as "George Straat 23". We would need to standardize our address representations. This type of link requires a DQ tool to match the BK; this DV automation framework would take the result of that exercise and populate the SLINK accordingly.

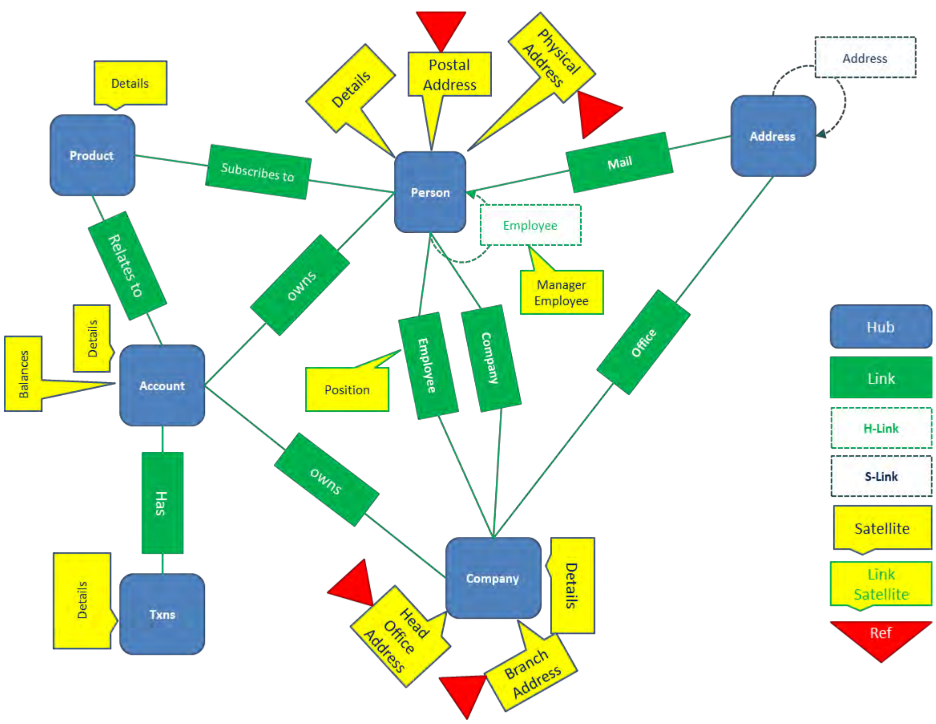


Figure 7 Hubs, links and satellites depicted

Attributes – the details recorded about each business entity

We now map the business context data from source to the satellite tables; DV will track changes as history.

SOURCE TABLE	SOURCE COLUMN	TARGET LIBRARY	TARGET TABLE	TARGET TRANSFORMATION	TARGET COLUMN NAME	TARGET COLUMN DATA TYPE	TARGET DATA BYTES	DV COLUMN TYPE
ADDRESS	ENTITY_ID	DV_CIS	S_ADDRESS_POSTAL		SOCIAL_SEC_NUM	CHAR	20	NK
ADDRESS	ADDRESS1	DV_CIS	S_ADDRESS_POSTAL		ADDRESS_LINE_1	CHAR	50	
ADDRESS	ADDRESS2	DV_CIS	S_ADDRESS_POSTAL		ADDRESS_LINE_2	CHAR	50	
ADDRESS	SUBURB	DV_CIS	S_ADDRESS_POSTAL		SUBURB	CHAR	20	
ADDRESS	PCODE	DV_CIS	S_ADDRESS_POSTAL		POSTAL_CODE	CHAR	4	
ADDRESS	STATE	DV_CIS	S_ADDRESS_POSTAL	LKP: DV_CIS.R_SCODE.STATE_NAM	STATE_CODE	CHAR	3	
ADDRESS	ENTITY_ID	DV_CIS	S_ADDRESS_POSTAL		SOCIAL_SEC_NUM	CHAR	20	NK
ADDRESS		DV_CIS	S_ADDRESS_PHYSICAL		ADDR_SEQ	NUM	8	SQ
ADDRESS	ADDRESS1	DV_CIS	S_ADDRESS_PHYSICAL		ADDRESS_LINE_1	CHAR	50	SQ
ADDRESS	ADDRESS2	DV_CIS	S_ADDRESS_PHYSICAL		ADDRESS_LINE_2	CHAR	50	SQ
ADDRESS	SUBURB	DV_CIS	S_ADDRESS_PHYSICAL		SUBURB	CHAR	20	SQ
ADDRESS	PCODE	DV_CIS	S_ADDRESS_PHYSICAL		POSTAL_CODE	CHAR	4	SQ
ADDRESS	STATE	DV_CIS	S_ADDRESS_PHYSICAL	LKP: DV_CIS.R_SCODE.STATE_NAM	STATE_CODE	CHAR	3	
ADDRESS	OWN	DV_CIS	S_ADDRESS_PHYSICAL		OCCUPANCY	CHAR	1	
PERSON	SOCIAL_SEC_NUM	DV_CIS	S_PERSON_DETAILS		SOCIAL_SEC_NUM	CHAR		NK
PERSON	NAME1	DV_CIS	S_PERSON_DETAILS		FIRST_NAME	CHAR	50	
PERSON	NAME2	DV_CIS	S_PERSON_DETAILS		MIDDLE_NAME	CHAR	50	
PERSON	NAME3	DV_CIS	S_PERSON_DETAILS		LAST_NAME	CHAR	50	
PERSON	DOB	DV_CIS	S_PERSON_DETAILS		DATE_OF_BIRTH	DATE	YYYYMMDD	
COMPANY	NAME	DV_CIS	S_COMPANY_DETALS		BUSINESS_NUMER	CHAR	40	NK
COMPANY	NAME	DV_CIS	S_COMPANY_DETALS		TRADING_NAME	CHAR	50	
ADDRESS	ADDRESS1	DV_CIS	H_ADDRESS		ADDRESS_LINE_1	CHAR	50	NK
ADDRESS	ADDRESS2	DV_CIS	H_ADDRESS		ADDRESS_LINE_2	CHAR	50	NK
ADDRESS	SUBURB	DV_CIS	H_ADDRESS		SUBURB	CHAR	20	NK
ADDRESS	PCODE	DV_CIS	H_ADDRESS		POSTAL_CODE	CHAR	4	NK
ADDRESS	STATE	DV_CIS	H_ADDRESS	LKP: DV_CIS.R_SCODE.STATE_NAM	STATE_CODE	CHAR	3	NK
ACCOUNT	ACCNO	DV_CIS	S_ACCOUNT_ONLWB_APPDETAILS		ACCOUNT_NUMBER	CHAR	10	NK

raries / Hubs / Satellites / Links / **Attributes** / DV Column Types

Figure 8 Mapping vault attributes

In **red** we have recorded the column types, lengths and a **DV column type**. Column types can either be *numeric or character* – the length of the field can be used to specify the date format we want – for example: 'yyyymmdd', 'ddmmyyyy'. **Table 1** below elaborates on the values of DV column types.

In **green** we have references to lookup tables; in this case the R_SCODE lookup table's STATE_NAME column is used to return the value for STATE_CODE. This can be classified as *standardizing the data* and is a DQ function; for example, if we see multiple spellings for "New South Wales" we only want to record "NSW" in DV.

Notice that in yellow we have the columns ADDRESS_LINE_1, ADDRESS_LINE_2, SUBURB, POSTAL_CODE and STATE_CODE mapped three times. Filtering by ADDRESS_TYPE we either map source data to S_ADDRESS_*POSTAL* or S_ADDRESS_*PHYSICAL* satellite tables and we load all addresses to H_ADDRESS hub table and define the address columns as natural keys (NK) to the hub. As depicted in the links mapping section we have a SLINK linked to H_ADDRESS.

In **blue** we see the use of a sequence number that allows S_ADDRESS_PHYSICAL satellite table to contain **multi active records** by business key. The reason for doing this is because a person can own more than one physical address – we record if they currently own the asset in the OCCUPANCY column from the source table.

Column acronym	Description
NK	Natural Key (this is the *business key*)
FD / FDT	From Date – we can use this if we plan to map DV "From Date" to a source table date or to use your own custom column name
TD / TDT	To Date – we can use this if we plan to map DV "To Date" to a source table date or to use your own custom column name

Column acronym	Description
SD / SDT	System Date – we can record the system date the data was uploaded
BD / BDT	Batch Date; although Hub table will have this for the first time the record was recorded, we can optionally capture this in a satellite table too
HDF	Hash Difference, default column name is HashDiff – algorithm is SHA256
SQ	Sequence – can be used for multi active satellite records
RS	Record Source, default column name is REC_SRC column in the hub table
LS	Last scene date, we can record the last time a record was loaded from source.

Table 1 DV column types

Links can be included in attribute mapping tab with **degenerate dimensions** which would be useful in a construct like a **transactional link (TLINK)**. We prefer to record transactions in a transactional satellite table due to the ease of maintenance and scalability of hub and satellite tables.

2 - UPDATE

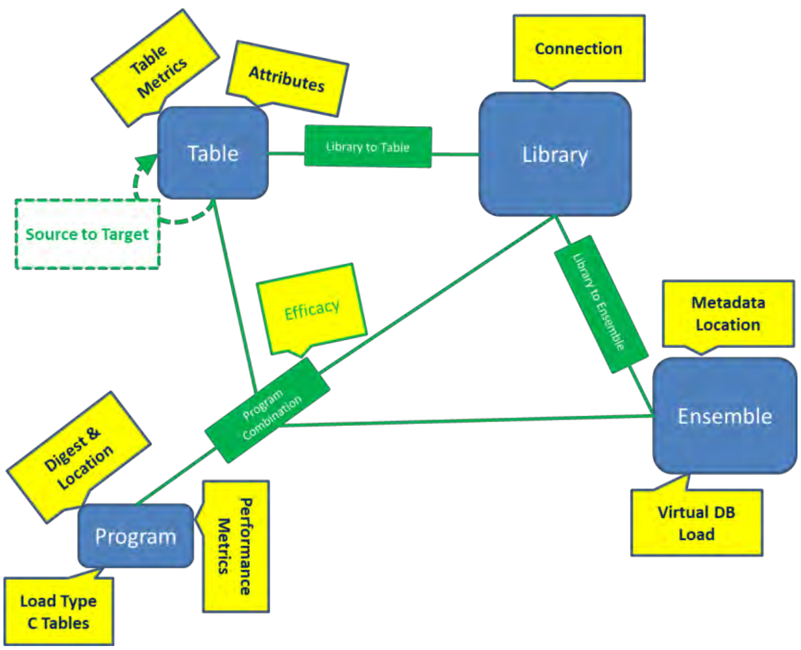


Figure 9 Schema Vault

Once your mapping exercise has completed run the custom utility "***%sasdvmapupdate***" and this will populate the DV **schema** with the changes you have applied. Recording the physical metadata of the DV you have designed into a **schema vault (SV)** allows the utility to have the same flexibility as the data model we are mapping.

One ensemble may have many libraries and a library will link to many tables. Together they will be used to generate **SAS programs** to load DV. Depending on the library type we can also populate the programs with pass-through code to execute **in database (inDB)** code.

Metric satellites have been added to the SV to record DV load performance and target table sizes; however these tables are optional.

The flexibility of this design allows us to:

- Have the same DV table being populated by differing ensembles.
- Create a single program (or many programs) to populate each table.
- Have a central place to query for DV performance.
- Build a **Business Data Vault (BDV)** that can be populated by the **RAW Data Vault (RDV)**.
- Provide an audit trail of the changes to the schema.
- Allow us to run multiple versions of DV depending on a date.
- Track program versions by way of *digest values*.
- Create **control tables** (C tables) for extract control.

3 - TEST

We can test the permutations of our mapping exercises by running "***%sasdvload***(*Mode*=DDL)" that will only create the tables we have defined in our mappings. The macro will query SV to autogenerate the tables. By running "***%sasdvload***(*Mode*=PGM)" the utility will also create the **load code** but it will <u>not</u> be executed. The programs will be saved to a location you specify in the format "***SASDVLoad_<Wave>_<Ensemble>_<SourceLibrary>_<SourceTable>_to_<TargetLibrary>_<TargetTable>.sas***". Run order (Wave) is determined by the type of HKs we have decided to use which can either be surrogate or hash keys. Let's breakdown what they are.

- **Surrogate Key (SK)** – these are ***dumb keys*** in the sense that there are no smarts behind it. As we load a new BK into the hub table we simply add the value of "1" to the previous maximum SK value. That means that as new BKs come in the surrogate key will increment by +1.
- **Hash Key (HK)** – we combine the identified natural keys (delimited with sanding values) and apply cryptography to create a unique digest value of the combination of those keys. The cryptography algorithm we use is **SHA256** as it has a *lowest hash collision risk*.

How does this determine the run order of the programs? As with **star schemas** you cannot populate **fact** tables without creating the SK in the **dimension tables** first. Therefore, dimension table loads must happen <u>before</u> loading fact tables. The same applies to DV; you need to populate hub tables before you can populate the satellite and link tables.

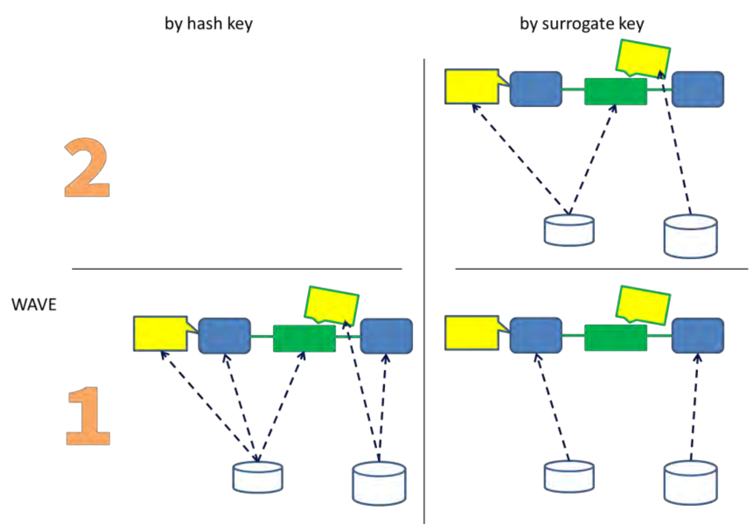


Figure 10 Data Vault by hash and surrogate key loading

With HKs this is no longer the paradigm. A **hash value** calculated using the same combination of BKs and hashing algorithm from whatever source will always generate the same HK and therefore satellites, hubs and links can be populated at the <u>same time</u>. This also means that a HK can be used across independent systems as long as they produce the same hash value. The only reason we do not use NKs instead of HKs is because we may not want to divulge the original NK value across independent systems for privacy or security reasons; for example an account number. Being able to populate all our tables at the same time flattens our run stream as depicted in **Figure 10**; the left side of the diagram is depicting updates using HKs while the right side is depicting the same updates but with SKs.

You can choose to recreate only parts of the DV by referencing only certain subsets of the ensemble and even by certain dates recorded in SV – yes, we can load multiple configurations into the SV and create the SAS programs or DDL or both. Here is an example where we are depicting the creation of DDL and SAS programs for a particular date by calling the macro utility

%sasdvload*(Mode=PGM DDL, Ensemble=CISDATA, Source=CLLCTR.ADDRESS, Date=25SEP2017 23:39:00)*. If Mode includes 'X' then the generated code will be executed, e.g. *Mode=PGM DDL X*.

All target tables and load code that uses the source table CLLCTR.ADDRESS will be interrogated by ***%sasdvload***. The target tables will not be replaced unless there has been a change in its structure <u>and</u> it is empty – a safety measure in case you have run this option accidently.

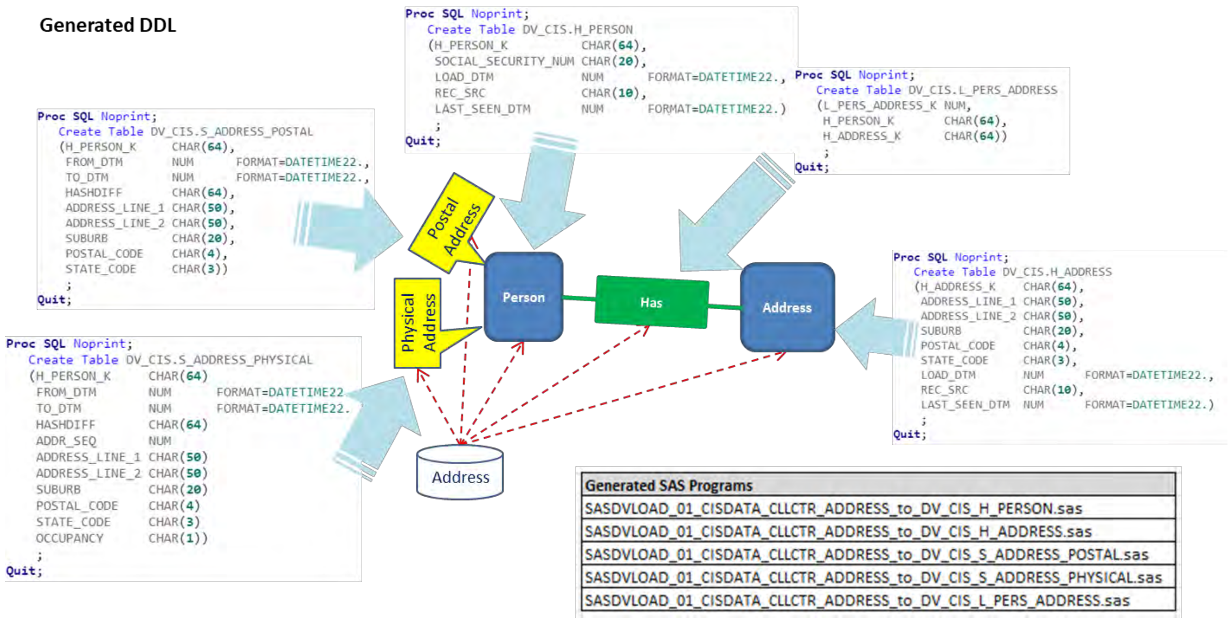


Figure 11 MODE=DDL (showing person and address artifacts only)

Code generated using this utility as depicted in **Figure 11** will be split by ensemble, source library, source table, target library and target table. Depending on how you parameterize the utility you could be generating a few SAS programs or thousands. The utility will rely on SAS connectivity through ACCESS products if the tables referenced are residing in a database other than SAS.

The SAS programs created by the utility will contain a **header block** with a version digest value **(Figure 12)**. Updates to existing programs will perform a check between the digest value in SV and those recorded in the SAS program's header block. The macro "***%sasdvload***" macro will replace the program if the digest values do not match. The updated program will be saved with the new digest value generated from SV. For example, if we have expanded the link table or added an attribute to a satellite table.

```
************************************************ ;
* PGM: SASDVLoad_01_CISDATA_DV_CIS_H_PERSON.sas  ;
* Generated on 17SEP17:10:46:53                  ;
* Digest: FD00D877E5568FDE22DBA1212974817D20     ;
************************************************ ;
```

Figure 12 Sample generated program header & digest value

As for the content of the generated program it will include the table DDL and load code. You may execute the utility repeatedly to check the content of the program the utility generates. In a later section we will discuss what the loading code will look like.

SAS Metadata

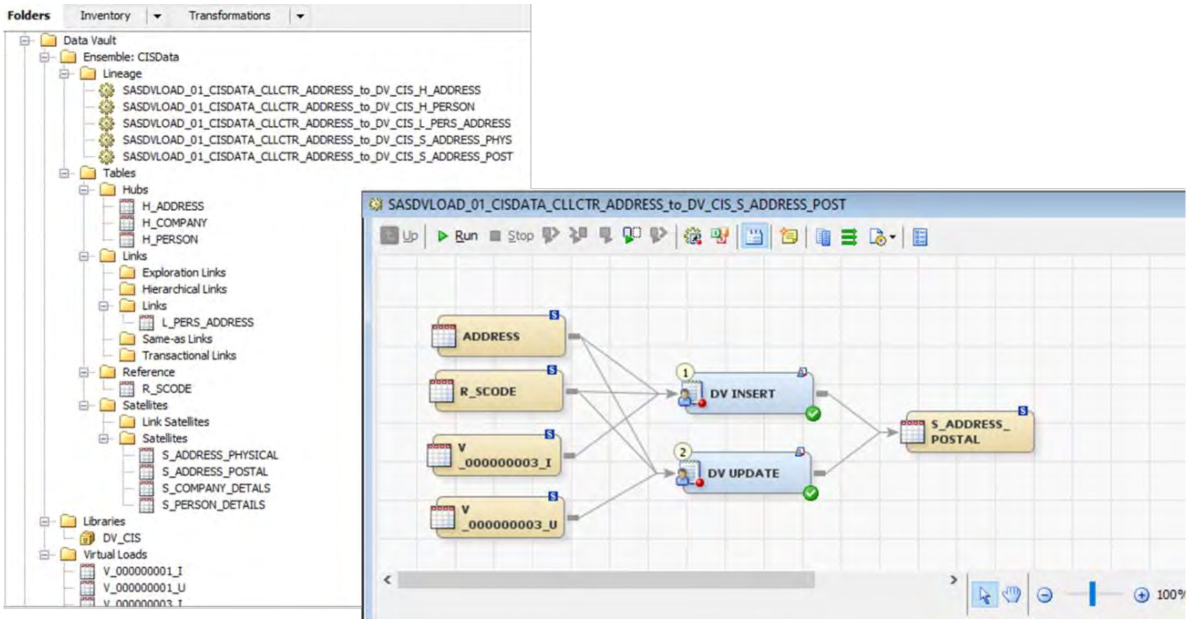


Figure 13 Sample metadata update (address sample only)

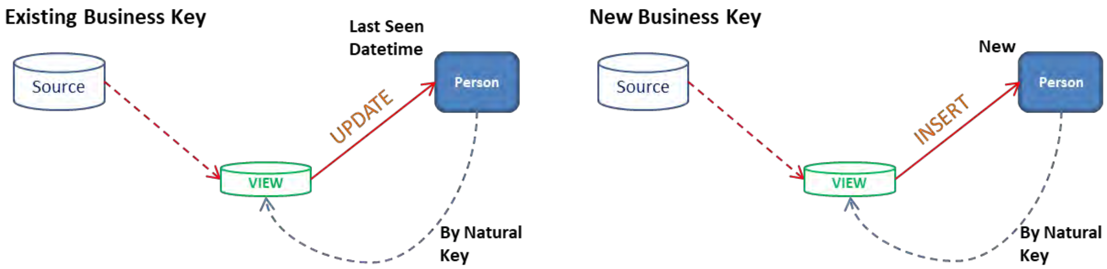
%sasdvload can be utilized to update SAS **metadata** as long as access to the SAS metadata has been granted. Issue the macro call with the same combinations as specified under TEST section except this time you need to issue the macro call with this switch: "***%sasdvload***(*Mode*=METADATA)". Once the metadata has been created you can then move the metadata to any *metadata tree* you wish. The step of registering metadata is important when you want your DV artefacts to be available to other SAS-based solutions to easily access it; products that use metadata such as *Information Maps* and *SAS DI Studio.* We record the source to target **data lineage** as **jobs** and you can use SAS DI Studio to **schedule** or create your SAS code execution string customized to your local environment. Of course you may choose to manage your metadata manually by registering the artefacts yourself.

%sasdvload will organize the DV artifacts into a tree as depicted by **Figure 13** grouping similar type tables together. The exceptions you can see are the reference and control tables; these you will need to be created yourself. The major advantage of using SAS metadata is the depiction of data lineage; that is being able to trace where the columns came from and how it was populated. Again you will need to include MODE=X to actually execute the utility programs or you may choose to run the generated programs yourself!

Virtualize the load (Vloads)

Moving physical data between environments is <u>costly</u> in terms of IO therefore the number of steps to update DV tables should be kept to a <u>minimum</u>. ***%sasdvload*** will create VIEWs that hold **delta** (∆) records used to update and/or insert records into DV. The VIEWs will be created once and will contain the ∆ between the target and source tables. If the DV tables are in a database and not a SAS dataset then the data movement will be kept within the database (inDB). Much like DV tables having limited number of table types there are a limited number of data load types and views. Once the VIEWs are created the only actual movement of data occurs when the results of these VIEWs are either **INSERT**ed to the target DV table or used to **UPDATE** the DV table. After the data has moved the views will be empty as the ∆ will be NULL – no difference will exist between source and target until the next iteration of loading data to DV.

In the following diagrams we only depict Vloads as SAS datasets for simplicity. SASDVBatchDatetime macro variable (in **Figure 14**) is the batch run's datetime which must be consistent for the entire batch run. We use the batch date time instead of the system date because we may be running the batch over midnight; a system date will be different for jobs that run before midnight and for those jobs that run after midnight.



VIEW Code

```
Proc SQL Noprint;
  Create View DV_CIS.V_000000001_u As
  Select SHA256HEX(Src.ENTITY_ID)) As H_PERSON_K
       , &SASDVBatchDatetime.   As LAST_SEEN_DTM Length=8
  From CLLCTR.Address Src
  Inner Join DV_CIS.H_PERSON Tgt
  On Src.ENTITY_ID=Tgt.SOCIAL_SECURITY_NUM
  Where (ADDRESS_TYPE='POSTAL'
  Or ADDRESS_TYPE='PHYSICAL')
  And &SASDVBatchDatetime. ne LAST_SEEN_DTM
  ;
Quit;
```

VIEW Code

```
Proc SQL Noprint;
  Create View DV_CIS.V_000000001_i As
  Select Distinct
         SHA256HEX(Src.ENTITY_ID)) As H_PERSON_K
       , ENTITY_ID                 As SOCIAL_SECURITY_NUM Length=20
       , &SASDVBatchDatetime.      As LOAD_DTM            Length=8
       , 'CLLCTR.ADDRESS'          As REC_SRC             Length=10
       , &SASDVBatchDatetime.      As LAST_SEEN_DTM       Length=8
  From CLLCTR.Address Src
  Where Not Exists (Select H_PERSON_K
                    From DV_CIS.H_PERSON Tgt
                    Where SHA256HEX(Src.ENTITY_ID))=Tgt.H_PERSON_K)
  And (ADDRESS_TYPE='POSTAL'
  Or ADDRESS_TYPE='PHYSICAL')
  ;
Quit;
```

UPDATE Code

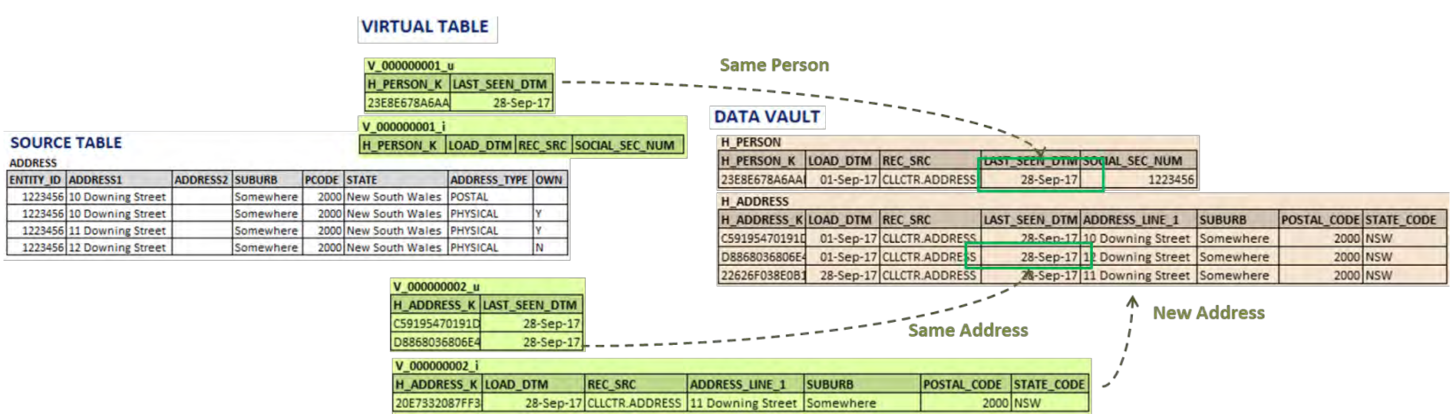
```
Proc SQL Noprint;
  Update DV_CIS.H_PERSON As Tgt
  Set LAST_SEEN_DTM = (Select LAST_SEEN_DTM
                         From DV_CIS.V_000000001_u Src
                         Where Tgt.H_PERSON_K=Src.H_PERSON_K)
                         Where Tgt.H_PERSON_K in
                         (Select H_PERSON_K From DV_CIS.V_000000001_u);
Quit;
```

INSERT Code

```
Proc SQL Noprint;
  Insert Into DV_CIS.H_PERSON
  Select *
  From DV_CIS.V_000000001_i;
Quit;
```

Figure 14 Updating hubs, ***pgm: SASDVLOAD_01_CISDATA_CLLCTR_ADDRESS_to_DV_CIS_H_PERSON.sas***

The **VIEW code** is generated once and the VIEW name is derived by HK value used to uniquely identify the load program in SV's *program hub*. Only if there is a change to the generated program version will the view be replaced by ***%sasdvload***. The **UPDATE** and **INSERT** code will be the only code that executes any data movement. 'LAST_SEEN_DTM' is an optional field and can be excluded from the DV load in the interest of load speed. We do updates before inserts as this will limit the number of records needed to be updated.



SOURCE TABLE – ADDRESS

ENTITY_ID	ADDRESS1	ADDRESS2	SUBURB	PCODE	STATE	ADDRESS_TYPE	OWN
1223456	10 Downing Street		Somewhere	2000	New South Wales	POSTAL	
1223456	10 Downing Street		Somewhere	2000	New South Wales	PHYSICAL	Y
1223456	11 Downing Street		Somewhere	2000	New South Wales	PHYSICAL	Y
1223456	12 Downing Street		Somewhere	2000	New South Wales	PHYSICAL	N

V_000000001_u

H_PERSON_K	LAST_SEEN_DTM
23E8E678A6AA	28-Sep-17

V_000000001_i

H_PERSON_K	LOAD_DTM	REC_SRC	SOCIAL_SEC_NUM

V_000000002_u

H_ADDRESS_K	LAST_SEEN_DTM
C59195470191D	28-Sep-17
D8868036806E4	28-Sep-17

V_000000002_i

H_ADDRESS_K	LOAD_DTM	REC_SRC	ADDRESS_LINE_1	SUBURB	POSTAL_CODE	STATE_CODE
20E7332087FF3	28-Sep-17	CLLCTR.ADDRESS	11 Downing Street	Somewhere	2000	NSW

DATA VAULT – H_PERSON

H_PERSON_K	LOAD_DTM	REC_SRC	LAST_SEEN_DTM	SOCIAL_SEC_NUM
23E8E678A6AA	01-Sep-17	CLLCTR.ADDRESS	28-Sep-17	1223456

H_ADDRESS

H_ADDRESS_K	LOAD_DTM	REC_SRC	LAST_SEEN_DTM	ADDRESS_LINE_1	SUBURB	POSTAL_CODE	STATE_CODE
C59195470191D	01-Sep-17	CLLCTR.ADDRESS	28-Sep-17	10 Downing Street	Somewhere	2000	NSW
D8868036806E4	01-Sep-17	CLLCTR.ADDRESS	28-Sep-17	12 Downing Street	Somewhere	2000	NSW
22626F038E0B1	28-Sep-17	CLLCTR.ADDRESS	28-Sep-17	11 Downing Street	Somewhere	2000	NSW

Figure 15 loading sample data to the hubs

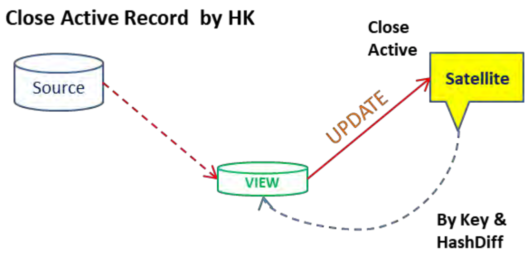
V_000000001_i will **not** INSERT new records from ADDRESS to H_PERSON – Person already exists.

V_000000001_u will UPDATE H_PERSON.LAST_SEEN_DTM

V_000000002_i will INSERT a new record for the new address

V_000000002_u will UPDATE H_ADDRESS.LAST_SEEN_DATE

Close Active Record by HK



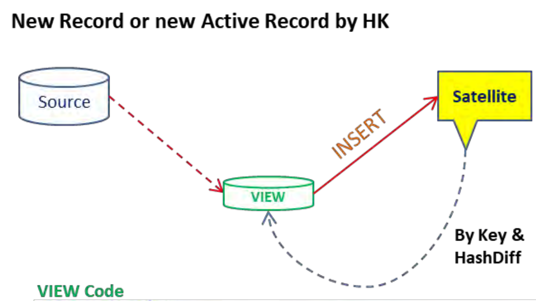
VIEW Code

```
Proc SQL Noprint;
  Create View DV_CIS.V_000000003_u As
  Select SHA256HEX(Src.ENTITY_ID) As H_PERSON_K
       , &SASDVBatchDatetime.-1  As TO_DTM
       , Tgt.HASHDIFF
  From (Select In01.*
             , Ref_001.STATE_CODE
        From CLLCTR.Address In01
        Left Join DV_CIS.SCODE Ref_001
        On In01.STATE = Ref_001.STATE_NAME) Src
  Inner Join DV_CIS.S_ADDRESS_POSTAL Tgt
  On  SHA256HEX(Src.ENTITY_ID) = Tgt.H_PERSON_K
  Where Src.ADDRESS_TYPE='POSTAL'
    And Tgt.TO_DTM = &SASDVHighDatetime.
    And SHA256HEX(Cat(Src.ADDRESS1
             ,'|',Src.ADDRESS2
             ,'|',Src.SUBURB
             ,'|',Src.PCODE
             ,'|',Src.STATE_CODE)) ^= Tgt.HASHDIFF
  ;
Quit;
```

UPDATE Code

```
 Proc SQL Noprint;
  Update DV_CIS.S_ADDRESS_POSTAL As Tgt
  Set TO_DTM = (Select TO_DTM
                From DV_CIS.V_000000003_u Src
                Where Tgt.H_PERSON_K=Src.H_PERSON_K
                  And Tgt.HASHDIFF  =Src.HASHDIFF)
                Where Tgt.H_PERSON_K in (Select H_PERSON_K From DV_CIS.V_000000003_u)
                  And Tgt.TO_DTM = &SASDVHighDatetime.;
Quit;
```

New Record or new Active Record by HK



VIEW Code

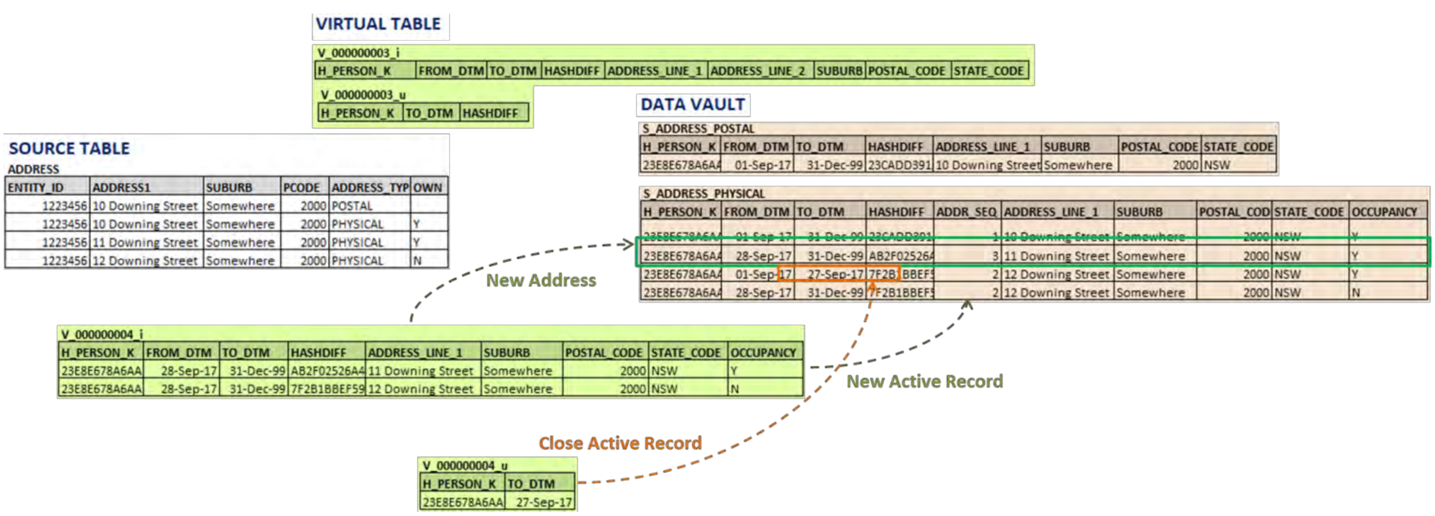
```
Proc SQL Noprint;
   Create View DV_CIS.V_000000003_i As
   Select SHA256HEX(Src.ENTITY_ID) As H_PERSON_K Length=64
        , &SASDVBatchDatetime. As FROM_DTM
        , &SASDVHighDatetime.  As TO_DTM
        , SHA256HEX(Cat(Src.ADDRESS1
         ,'|',Src.ADDRESS2
         ,'|',Src.SUBURB
         ,'|',Src.PCODE
         ,'|',Ref_001.STATE_CODE)) As HASHDIFF
        , Src.ADDRESS1            As ADDRESS_LINE_1 Length=50
        , Src.ADDRESS2            As ADDRESS_LINE_2 Length=50
        , Src.SUBURB                                Length=20
        , Src.PCODE               As POSTAL_CODE    Length=4
        , Ref_001.STATE_CODE
   From CLLCTR.Address Src
   Left Join (Select *
              From DV_CIS.S_ADDRESS_POSTAL
              Where TO_DTM=&SASDVHighDatetime.) Tgt
   On  SHA256HEX(Src.ENTITY_ID) = Tgt.H_PERSON_K
   Left Join DV_CIS.SCODE Ref_001
   On Src.STATE = Ref_001.STATE_NAME
   Where Src.ADDRESS_TYPE='POSTAL'
     And (Tgt.H_PERSON_K is Null)
   ;
Quit;
```

INSERT Code

```
Proc SQL Noprint;
   Insert Into DV_CIS.S_ADDRESS_POSTAL
   Select *
   From DV_CIS.V_000000003_i;
Quit;
```

Figure 16 Update satellites, ***pgm: SASDVLOAD_01_CISDATA_CLLCTR_ADDRESS_to_DV_CIS_S_ADDRESS_POSTAL.sas***

Satellite tables involve a slightly more complex load pattern. We will UPDATE to close off records before we INSERT new records. This is to avoid complications if a load fails midway; we can simply rerun the .sas code after whatever issue came up is resolved. Changed records in the INSERT VIEW are determined by comparing the source against the active target table records by HK and by HashDiff columns. The UPDATE code will close off the active record if new records are available by the same HK.



VIRTUAL TABLE

V_000000003_i

H_PERSON_K	FROM_DTM	TO_DTM	HASHDIFF	ADDRESS_LINE_1	ADDRESS_LINE_2	SUBURB	POSTAL_CODE	STATE_CODE

V_000000003_u

H_PERSON_K	TO_DTM	HASHDIFF

SOURCE TABLE

ADDRESS

ENTITY_ID	ADDRESS1	SUBURB	PCODE	ADDRESS_TYP	OWN
1223456	10 Downing Street	Somewhere	2000	POSTAL	
1223456	10 Downing Street	Somewhere	2000	PHYSICAL	Y
1223456	11 Downing Street	Somewhere	2000	PHYSICAL	Y
1223456	12 Downing Street	Somewhere	2000	PHYSICAL	N

DATA VAULT

S_ADDRESS_POSTAL

H_PERSON_K	FROM_DTM	TO_DTM	HASHDIFF	ADDRESS_LINE_1	SUBURB	POSTAL_CODE	STATE_CODE
23E8E678A6AA	01-Sep-17	31-Dec-99	23CADD391	10 Downing Street	Somewhere	2000	NSW

S_ADDRESS_PHYSICAL

H_PERSON_K	FROM_DTM	TO_DTM	HASHDIFF	ADDR_SEQ	ADDRESS_LINE_1	SUBURB	POSTAL_COD	STATE_CODE	OCCUPANCY
~~23E8E678A6AA~~	~~01-Sep-17~~	~~31-Dec-99~~	~~23CADD391~~	~~1~~	~~10 Downing Street~~	~~Somewhere~~	~~2000~~	~~NSW~~	~~Y~~
23E8E678A6AA	28-Sep-17	31-Dec-99	AB2F02526A	3	11 Downing Street	Somewhere	2000	NSW	Y
23E8E678A6AA	01-Sep-17	27-Sep-17	7F2B1BBEF5	2	12 Downing Street	Somewhere	2000	NSW	Y
23E8E678A6AA	28-Sep-17	31-Dec-99	7F2B1BBEF5	2	12 Downing Street	Somewhere	2000	NSW	N

V_000000004_i

H_PERSON_K	FROM_DTM	TO_DTM	HASHDIFF	ADDRESS_LINE_1	SUBURB	POSTAL_CODE	STATE_CODE	OCCUPANCY
23E8E678A6AA	28-Sep-17	31-Dec-99	AB2F02526A4	11 Downing Street	Somewhere	2000	NSW	Y
23E8E678A6AA	28-Sep-17	31-Dec-99	7F2B1BBEF59	12 Downing Street	Somewhere	2000	NSW	N

V_000000004_u

H_PERSON_K	TO_DTM
23E8E678A6AA	27-Sep-17

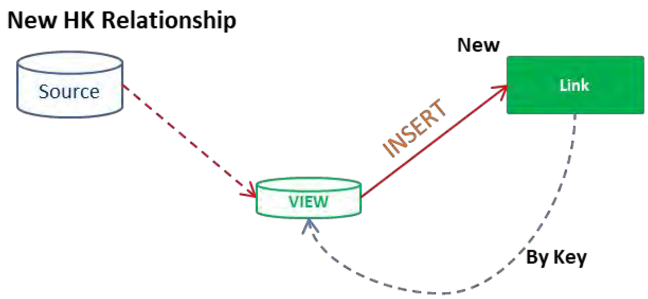
Figure 17 loading sample data to the satellites

V_000000003_i will not contain any new records to INSERT to S_ADDRESS_POSTAL

V_000000003_u will have no records to UPDATE

V_000000004_i will contain the changed record and new record to INSERT to S_ADDRESS_PHYSICAL

V_000000004_u will contain the changed record to close in S_ADDRESS_PHYSICAL



INSERT Code

```
Proc SQL Noprint;
   Insert Into DV_CIS.L_PERS_ADDRESS
   Select *
   From DV_CIS.V_000000005_i;
Quit;
```

VIEW Code

```
Proc SQL Noprint;
  Create Table DV_CIS.V_000000005_i As
  Select SHA256HEX(Src.ENTITY_ID) As H_PERSON_K
        , SHA256HEX(Cat(Src.ADDRESS1
        ,'|',Src.ADDRESS2
        ,'|',Src.SUBURB
        ,'|',Src.PCODE
        ,'|',Ref_001.STATE_CODE)) As H_Address_K
        , Coalesce((Select Max(L_PERS_ADDRESS_K)
          From DV_CIS.L_PERS_ADDRESS), 0) + Monotonic() As L_PERS_ADDRESS_K
  From CLLCTR.Address Src
  Left Join DV_CIS.SCODE Ref_001
  On Src.STATE = Ref_001.STATE_NAME
  Where Not Exists (Select H_PERSON_K, H_Address_K
                    From DV_CIS.L_PERS_ADDRESS Tgt
                    Where SHA256HEX(Src.ENTITY_ID), $Hex32.)=H_PERSON_K
                      And SHA256HEX(Cat(Src.ADDRESS1
                                  ,'|',Src.ADDRESS2
                                  ,'|',Src.SUBURB
                                  ,'|',Src.PCODE
                                  ,'|',Ref_001.STATE_CODE)), $Hex32.))
  And (ADDRESS_TYPE='POSTAL'
  Or ADDRESS_TYPE='PHYSICAL')
  ;
Quit;
```

Figure 48 Update links, ***pgm: SASDVLOAD_01_CISDATA_CLLCTR_ADDRESS_to_DV_CIS_L_PERS_ADDRESS.sas***

In our example, we will not have any new link records to load as the relationships are already recorded.

Staging

DV can be loaded directly from the data sources or it can be **staged** before loading. If you are prototyping new ensembles to load into DV you may consider one of these types of staging:

- **Persistent Staging** – history is kept in staging and you can **replay** loading to DV
- **Transient Staging** – the data in staging is a *pit stop* and is only kept in this area until the next load iteration.

Finally, you need to consider the type of data sources you are loading whether it is a **PULL** or a **PUSH**. PUSH files are files dumped in a **landing zone** for ETL to pick up and PULL is the act of reaching into the data source and pulling the data we need. For this the ***%sasdvload*** does come with template **control tables** and they can be generated for each generated .sas program file. This is the **C Table** portion you saw in **Figure 9** and once the templates are generated you would be expected to populate it externally to ***%sasdvload*** utility.

Run Num	Extract Date	Extract Time	Number of Records Extracted	Last Value
0	1JAN1960	00:00:01	0	.
1	27SEP2017	19:06:56	32131124	27SEP2017: 01:09:53
2	27SEP2017	20:22:15	1221341	27SEP2017: 19:09:22

increments +1 at each run

Represents the Max Value from Extract; the next extract will extract everything newer than the last max value

Date and time of extract

Figure 19 C table template

Referring to **Figure 19** “Last Value” can be anything you decide to use to extract from the source contents either by date time or identity column id or you can track the suffix derived from a source file name; for example, a file source table or file named “ADDRESS_*20170927*”. This means that you code your solution to fetch the suffix of the filename in the "yyyymmdd" format and you may need to loop the load in the order as the files come in and update the C table accordingly.

When designing your solution with ***%sasdvload*** consider if the data you retrieve is a **snapshot** or a Δ, how often the data is retrieved and ensure that the data arrives in a consistent format – if it doesn't then how do we trap that data for loading? Do we allow the DV processes to continue if there is an exception or stop the loads altogether? Consider that if you receive a snapshot what happens if there are records missing in the source, *do we close the record in the target table?* The data must remain unique by business key before loading into DV. Data with duplicates will create ***Gaps and Islands*** and may be difficult to take apart and reload – unless you can restore DV to a previous state. If a single data source has multiple BKs in them then consider creating VIEWs over that data source to query the data uniquely and register those views as data sources in SV.

4 - VAULT!

Now that we have successfully tested the vault changes we can schedule the DV loads. The design of this platform is so that complete flexibility is in mind. We can reproduce the whole DV or parts of DV as depicted in the "***%sasdvload***" macro calls mentioned earlier in this paper. Care needs to be taken when making changes to existing structures as you need to decide how you to rebuild the table with its history. Changes to existing structures are not recommended as that is in part the flexibility of DV; should you need additional attributes from a source table then you can add a new satellite to that hub. Different sources to a common hub should have different satellite tables as you will rarely see two or more data sources with common data columns, granularity and latency. Separate these satellite table names by including an acronym for the source in the satellite table name. You should be grabbing as much as you can from the source tables in your initial build because potentially the business has not decided what to do with the data yet. Satellite tables should also be populated according to their rate of change – much like designing dimensional tables – try to avoid having too few columns in a satellite such as 6NF models (for example an **anchor model** - a link to what this is in the reference section). If you really need to reload your satellite table then I would recommend that you rebuild the satellite table in isolation, load it in parallel and then retire the old satellite table after you are happy with it.

If developing DV on premise I would also discuss the need to optimize the storage of DV tables with IT by ensuring that the physical disk location of the tables is decentralized; such as what is possible on a **Massively Paralleled Platforms (MPP)**. Architecturally this would remove the bottleneck of having all data loading forced through a single bus. It will also enhance the table querying experience as the DV table partitions are spread across multiple disks and thus parallel querying and multithreading is made possible.

Query Performance Tables – PIT & Bridge, Supernova

A concern of DV is the number of tables you would need to join in your query. DV provides two types of table structures in order to assist in query performance and both are snapshots of the data.

- **Point-in-Time (PIT)** tables that are designed around a hub and its satellite tables.
- **Bridge tables** that are used to join hubs and links together; like a **super-link table.**

Both are derived from existing DV tables and the frequency of updating them is entirely up to the designer. They are snapshots and can be designed with transactional dates or with a **date range** by utilizing "From" and "To" dates respectively. Instead of creating them as tables you could create them as VIEWs so you only have to ever create them once. Depending on the inDB technology used to store DV a query performance VIEW may be further enhanced by creating them as **materialized** or **indexed views** – the database will incrementally **cache** with frequently queried data.

In our example, we have taken daily snapshots of the satellite tables surrounding the Person hub table and saved them in a PIT table P_Person. One of the tables has a sequence number so we need to capture all the addresses per day coming from the satellite by hub key + sequence number. Notice that on 28 September we loaded a new sequence. The query performance is enhanced because by using PIT table you do not need to write a BETWEEN clause in your SQL query – you can use the PIT date value – this will use the satellite table's indexes more efficiently.

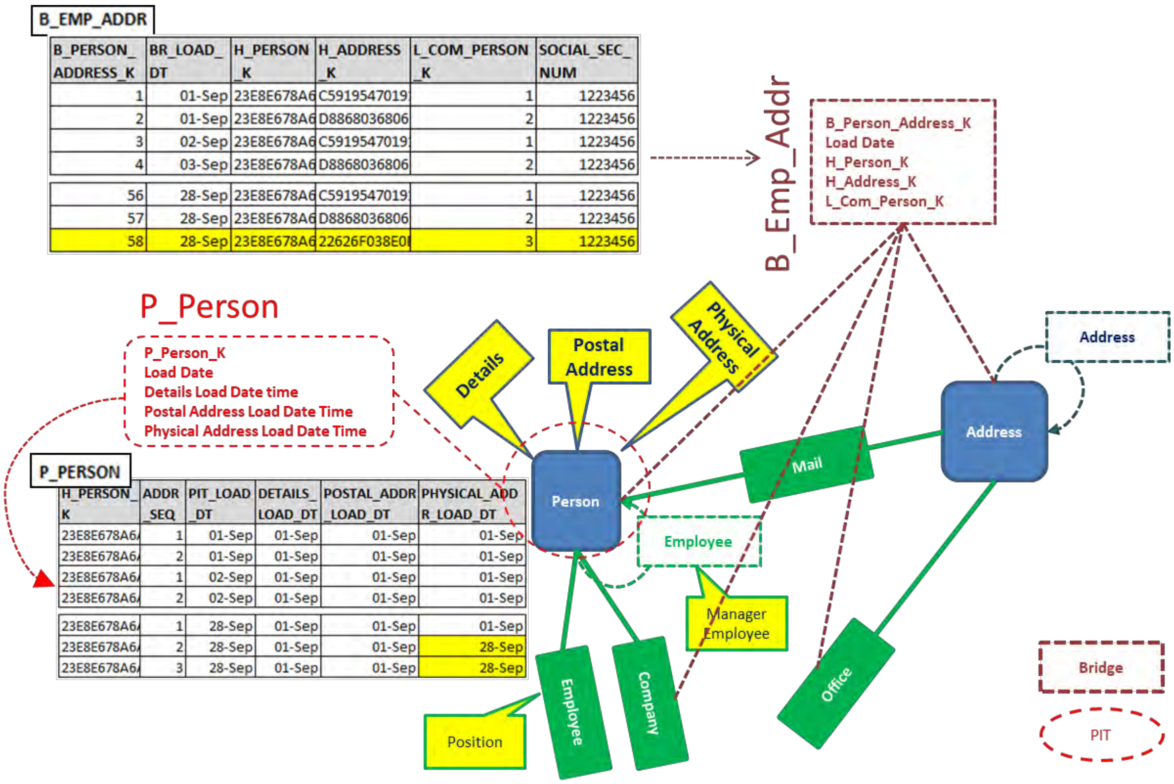


Figure 20 Query assistance tables, PIT and bridge

Don't forget that if you are joining satellites tables together that are linked by separate hubs but you do not need the BK returned in your query then it is ok to not include the hub in your query! You can also use virtual table technology outside of DV to create pre-canned virtual tables as depicted in the article "*Data Vault and Data Virtualization: Double Agility*" (a link to this paper is in the reference section below). The author has dubbed his VIEW structures as **supernova** tables. These are outside the scope of this paper.

Data Quality

Earlier in this paper we depicted the use of same-as links for matching two or more addresses. The idea behind DQ is to clean "dirty" data and standardize the data into a format that results in better quality analytics. You can register the results of DQ exercises and load those results (into a BDV) by registering the Slink entries in SV. In **Figure 21** we have expanded the model with a new hub and slink for standardized addresses.

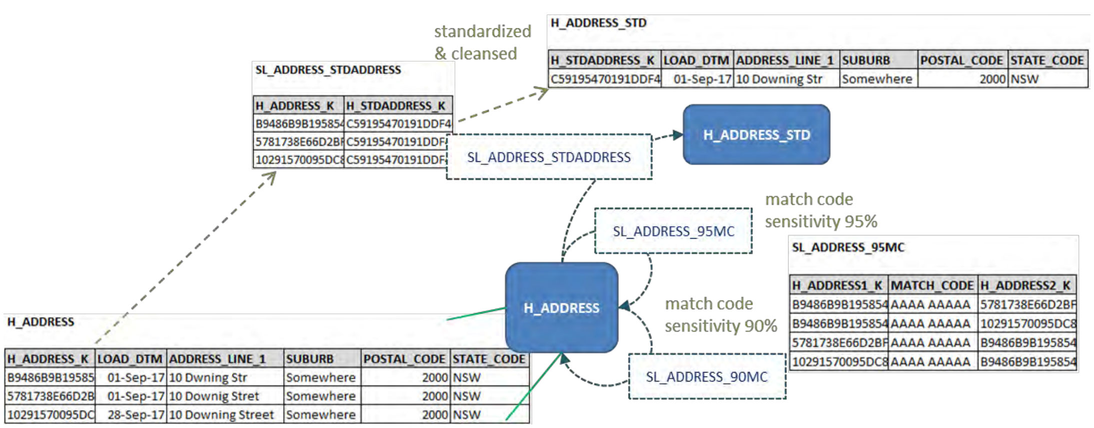


Figure 21 DQ address example

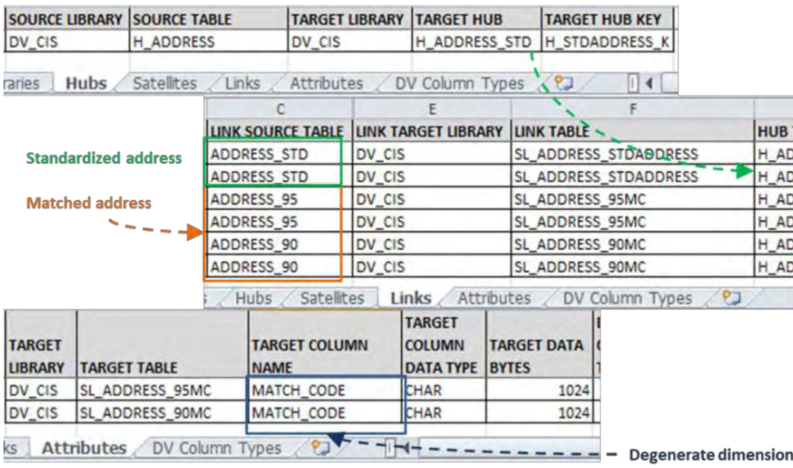


Figure 22 DQ SV registration

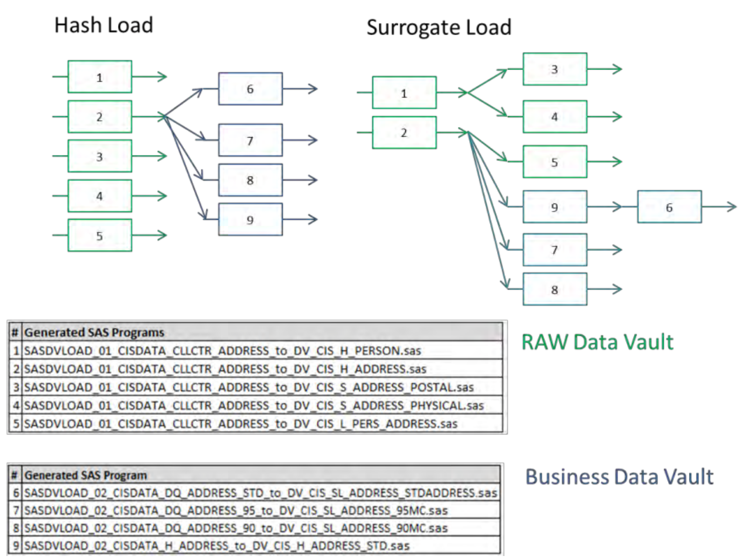
A holding table is needed between loading RDV and BDV; the loading patterns for hub tables remain the same. We can use DataFlux in Base SAS to match and standardize our addresses.

```
STD_Address1=dqStandardize(Address1, 'Address');
STD_Address2=dqStandardize(Address2, 'Address');

ADDRESS95=dqMatch(CAT(ADDRESS1, '|', ADDRESS2, '|', SUBURB, '|', PCODE, '|', STATE)
          , 'Address', 95, 'ENAUS');
```

Figure 23 DataFlux code sample

Scheduling



If RDV is based on HK*s* then all the hub, links and satellite tables can be loaded simultaneously whereas a DV based on SK*s* will need the hub tables loaded <u>before</u> all the other tables. Each program generated will need to be scheduled.

Of course if you have BDV components that are based on RDV components then the BDV load programs will need to be scheduled after the RDV programs because RDV updates occur <u>before</u> loads to BDV. You can have BDV populated immediately from source; the risk occurs if you need to change the derived columns or column values retrospectively and you can no longer retrieve the source.

Figure 24 Scheduling Hash or Surrogate Key based DV

Dynamic job run

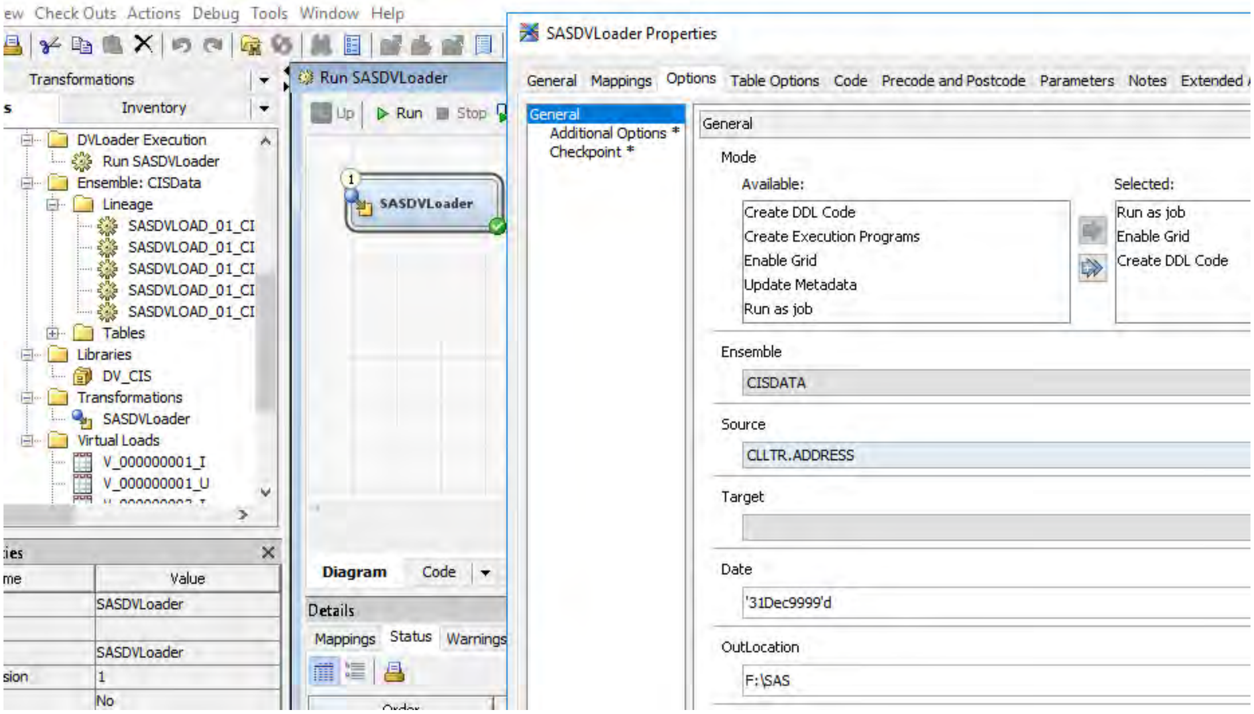


Figure 25 SASDVLoader as a job

SASDVLoader transformation can be used to automate the loading of DV and the modes are the same as specified parameter for ***%sasdvload*** macro in the screenshot you can see one additional option, "MODE=Run as job". Instead of publishing .sas code to disk to be scheduled later the DI job will also execute the job. Therefore you could use the transformation in multiple DI Studio jobs with different parameters as specified in the screenshot; they are all optional. If you do not include these options then the scheduled DI job will run everything defined in Schema Vault. As you can imagine you could load the same table from the same source multiple times a day and it is up to you how often you schedule the job. Additionally if you do not specify an "Outlocation" to save the generated SAS program the program will be loaded temporarily into the SAS Work directory.

RE-VAULT!

The reverse of the 4-step automation process is possible; that is to scan an existing DV and automatically create a mapping spreadsheet with associated hub, link, satellite and attribute tabs. I call this process "*RE-Vault!*" or "Reverse Engineer Data Vault".

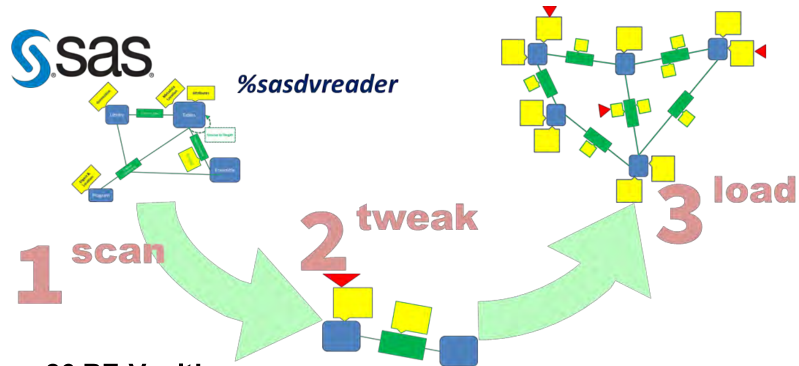


Figure 26 RE-Vault!

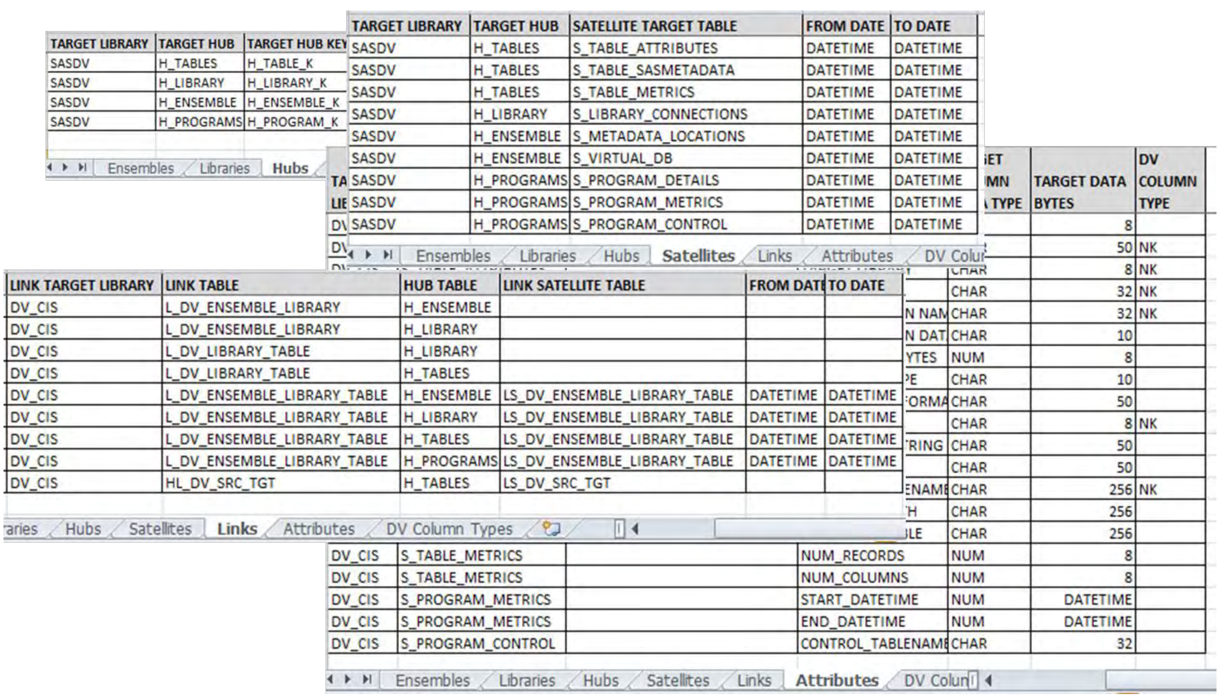
This could be necessary if you are looking to replace a manually created DV ETL framework with that of ***%sasdvload***. This is achieved in three easy steps.

1. Scan - Run ***%sasdvreader(****read=, outputfile=****)*** where the read parameter can either be just the libraries or library.tablenames (delimited by blank spaces) that you want the utility to read. Outputfile is the name and location of where you want to save the Excel mapping sheet.

2. Tweak - Open the Excel sheet and tweak any changes you need to do to the mapping. Typically, things like "FILTER", "TARGET_FORMATION" and "ENSEMBLE" that cannot be determined by reading the DICTIONARY.TABLES entries of the DV; these elements would be custom code specific. For access to the underlying sys.tables (as in SQL Server) the authenticated user that reads the sys tables will need to be granted access to read it. These tables are usually invisible to the SAS interface but can still be reached by running SAS PROC SQL code.

3. Load – Vault through the 4-step automation process.

After running the command "***%sasdvreader(****read=SASDV, outputfile=c:\SAS\SASDVLoader.xlsx****)***" our spreadsheets will look like **Figure 27**. Here we have mapped SV into mapping sheets



TARGET LIBRARY	TARGET HUB	TARGET HUB KEY
SASDV	H_TABLES	H_TABLE_K
SASDV	H_LIBRARY	H_LIBRARY_K
SASDV	H_ENSEMBLE	H_ENSEMBLE_K
SASDV	H_PROGRAMS	H_PROGRAM_K

TARGET LIBRARY	TARGET HUB	SATELLITE TARGET TABLE	FROM DATE	TO DATE
SASDV	H_TABLES	S_TABLE_ATTRIBUTES	DATETIME	DATETIME
SASDV	H_TABLES	S_TABLE_SASMETADATA	DATETIME	DATETIME
SASDV	H_TABLES	S_TABLE_METRICS	DATETIME	DATETIME
SASDV	H_LIBRARY	S_LIBRARY_CONNECTIONS	DATETIME	DATETIME
SASDV	H_ENSEMBLE	S_METADATA_LOCATIONS	DATETIME	DATETIME
SASDV	H_ENSEMBLE	S_VIRTUAL_DB	DATETIME	DATETIME
SASDV	H_PROGRAMS	S_PROGRAM_DETAILS	DATETIME	DATETIME
SASDV	H_PROGRAMS	S_PROGRAM_METRICS	DATETIME	DATETIME
SASDV	H_PROGRAMS	S_PROGRAM_CONTROL	DATETIME	DATETIME

LINK TARGET LIBRARY	LINK TABLE	HUB TABLE	LINK SATELLITE TABLE	FROM DATE	TO DATE
DV_CIS	L_DV_ENSEMBLE_LIBRARY	H_ENSEMBLE			
DV_CIS	L_DV_ENSEMBLE_LIBRARY	H_LIBRARY			
DV_CIS	L_DV_LIBRARY_TABLE	H_LIBRARY			
DV_CIS	L_DV_LIBRARY_TABLE	H_TABLES			
DV_CIS	L_DV_ENSEMBLE_LIBRARY_TABLE	H_ENSEMBLE	LS_DV_ENSEMBLE_LIBRARY_TABLE	DATETIME	DATETIME
DV_CIS	L_DV_ENSEMBLE_LIBRARY_TABLE	H_LIBRARY	LS_DV_ENSEMBLE_LIBRARY_TABLE	DATETIME	DATETIME
DV_CIS	L_DV_ENSEMBLE_LIBRARY_TABLE	H_TABLES	LS_DV_ENSEMBLE_LIBRARY_TABLE	DATETIME	DATETIME
DV_CIS	L_DV_ENSEMBLE_LIBRARY_TABLE	H_PROGRAMS	LS_DV_ENSEMBLE_LIBRARY_TABLE	DATETIME	DATETIME
DV_CIS	HL_DV_SRC_TGT	H_TABLES	LS_DV_SRC_TGT		

					TARGET DATA BYTES	DV COLUMN TYPE
					8	
					50	NK
				CHAR	8	NK
				CHAR	32	NK
			N NAM	CHAR	32	NK
			N DAT	CHAR	10	
			YTES	NUM	8	
			PE	CHAR	10	
			ORMA	CHAR	50	
				CHAR	8	NK
			RING	CHAR	50	
				CHAR	50	
			ENAMI	CHAR	256	NK
			H	CHAR	256	
			LE	CHAR	256	
DV_CIS	S_TABLE_METRICS		NUM_RECORDS	NUM	8	
DV_CIS	S_TABLE_METRICS		NUM_COLUMNS	NUM	8	
DV_CIS	S_PROGRAM_METRICS		START_DATETIME	NUM	DATETIME	
DV_CIS	S_PROGRAM_METRICS		END_DATETIME	NUM	DATETIME	
DV_CIS	S_PROGRAM_CONTROL		CONTROL_TABLENAME	CHAR	32	

Figure 27 RE-Vault Schema Vault into mapping worksheets and tabs

CONCLUSION

DV is a very flexible way to capture all the data to your warehouse. DV can grow to a constellation of tables but the complexity is simplified due to the familiarity of the table structures we know how to query. SAS can be used to automate the data loads and together with SAS/ACCESS products the loads can be customized to be agnostic to the storage technology underneath. DV is all about capturing all the facts of the business and the representation of the data captured allows for more exploration of not only the contents but the relationships as depicted in the model. We can utilize other SAS products to mine the data, build dimensional models from the data, develop a self-serve BI environment for exploring the data and port other SAS solutions onto DV. Importantly DV can be used as a step towards **Master Data Management** and the "Golden Record".

As DV grows a **business glossary** will be needed to track where the data is and how it is stored and to *socialize* the data.

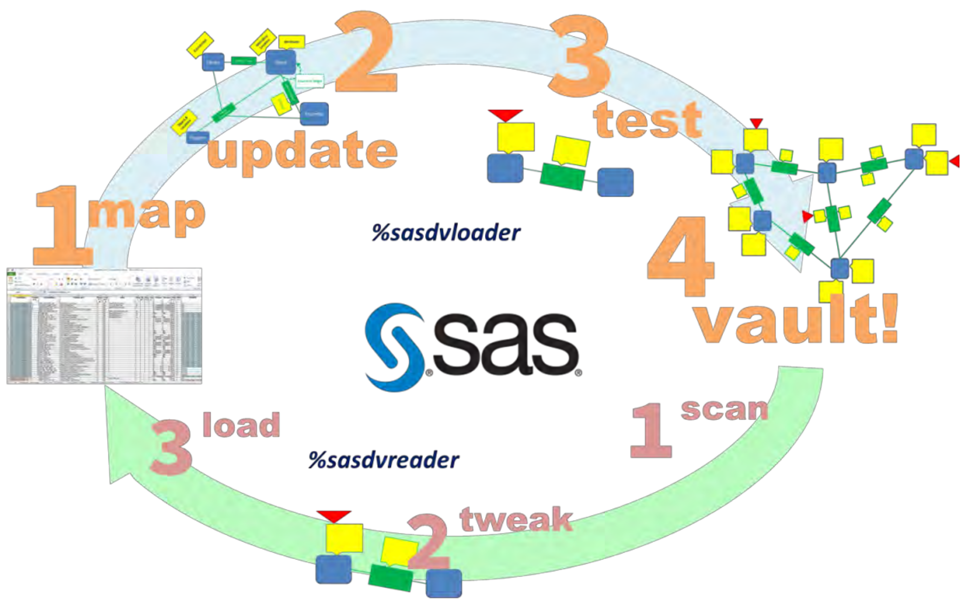


Figure 28 Vault & RE-Vault!

The framework discussed in this paper can be used to develop a new DV or inherit an existing DV and automate the loading of DV through SAS.

Thus this framework can be used to:

- reduce development costs;
- improve analytical value;
- easily test and implement change;
- integrate to the platform; and
- treat data as a corporate asset

Also DV in practice what is modeled <u>conceptually</u> is usually what is <u>implemented</u>.

Special thanks to **Selerity** for access to their environment for developing screenshots and testing code; you can find them here: https://seleritysas.com/ for your SAS **cloud** solutions.

REFERENCES

The Data Administrator, Linstedt, D. “Data Vault Series 1 – Data Vault Overview”. 1 July 2002. Available at http://tdan.com/data-vault-series-1-data-vault-overview/5054.

Linstedt, D. 2008. *Supercharge your Data Warehouse*: Dan Linstedt.

Linstedt, D; Olschimke, M. 2016. *Building a Scalable Data Warehouse with Data Vault 2.0*: Elsevier.

Corr, L. 2011. *Agile Date Warehouse Design*: Decision Press.

Hultgren, H. 2012. *Modeling the Agile Data Warehouse with Data Vault*: New Hamilton.

The Data Warrior, Graziano, K. “Data Vault vs.The World (3 of 3)”. January 2013. Available at https://kentgraziano.com/2013/01/27/data-vault-vs-the-world-3-of-3/.

What’s New in Encryption in SAS 9.4. Available at http://support.sas.com/documentation/cdl/en/secref/69831/HTML/default/viewer.htm#secrefwhatsnew94.htm.

Anchor: any time to the time of any, Available at http://www.anchormodeling.com/.

Camps, D. 2013, “The SQL of Gaps and Islands in Sequences”, Available at https://www.red-gate.com/simple-talk/sql/t-sql-programming/the-sql-of-gaps-and-islands-in-sequences/

Van der Lans, R, “Data Vault and Data Virtualization: Double Agility”, March 2015, Available at https://www.cisco.com/c/dam/en_us/services/enterprise-it-services/data-virtualization/documents/whitepaper-cisco-datavaul.pdf

Vos,R 2014, “Driving Keys and relationship history, one or more tables?” Available at http://roelantvos.com/blog/?p=1253

RECOMMENDED READING

- *Grid Computing in SAS® 9.4*
- *SAS® 9.4 Language Interfaces to Metadata*
- *Base SAS® 9.4 Procedures Guide*
- *SAS(R) Data Integration Studio 4.903: User's Guide*
- *SAS® 9.4 Data Quality Server: Reference, Third Edition*

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Patrick Cuba
Principal Consultant & Certified Data Vault Modeler
Cuba BI Consulting
+61 (0) 458 912 634
patrickcuba8@gmail.com
https://au.linkedin.com/in/patrickcuba