Data vault method coverage

This document provides examples for all expected or seen model and loading scenarios. Please note, that some of the syntax is only announced but not yet implemented in the compiler reference implementation.

Also please keep in mind, that json does not use linefeed or indention as syntax elements. **The examples are formatted for better human readability only**.

Credits and license

Data Vault stereo type coverage and examples

The following examples provide field and model DVPD declarations for all official data vault stereotypes and their variations. It includes all examples to the stereotypes, explained in the book "Building A Scalable Data Warehouse With Data Vault 2.0" by Dan Linstedt and Michael Olschimke from 2016. These examples are marked with "{DV-<chapter>.<subchapter>}" in the heading to provide a linkt to the book.

To be easy understandable, the examples use simplified table and column names that don't follow all best practices. For the same reason, not all examples use the busniess cases of the data vault book.

Some properties of the DVPD can be declared on the level of the **model profile**. All examples refer to the following, most common, profile settings:

```
{   "compare_criteria": "key+current",
    "uses_diff_hash_default": true,
    "is_enddated_default": true,
    "has_deletion_flag_default": true
}
```

Hub tables

Hub with a single column business key {DV-4.3}

```
"fields": [
            {   "field_name": "AIRLINE_ID"
```

```
                ,"field_type": "Varchar(20)"
                ,"targets": [{"table_name": "airline_hub"}]
            },
              ...
        ],
"tables": [
            {   "table_name": "airline_hub"
               ,"table_stereotype": "hub"
               ,"hub_key_column_name": "HK_AIRLINE"
            }
              ...
        ]
```

Notes:
Mapping a field to a hub declares it to be a busniess key. No more informations are necessary. The busniess key column in the hub will have the same name and type like the field.

Hub with a composite business key {DV-4.3.1.1}

This example assumes, that customer id's are not unique over the different web shops, the data is collected from. Therfore the web shop id must also be used for identification.

```
"fields": [
            {   "field_name": "WEBSHOP_ID",
                "field_type": "integer",
                "targets": [{"table_name": "customer_hub"}]
            },
            {   "field_name": "CUSTOMER_ID",
                "field_type": "Varchar(20)",
                "targets": [{"table_name": "customer_hub"}]
            },
              ...
        ],
"tables": [
            {   "table_name": "customer_hub",
                "table_stereotype": "hub",
                "hub_key_column_name": "HK_CUSTOMER"
            },
              ...
        ]
```

Notes:
In the example there are no further declarations about the concatenation order of the businesskeys for calculating the hash of the hub key. This implies, that there is some hard coded ordering rules applied, when the loading code is implemneted/generated (e.g. alphabethical order). Keep in mind, that there are properties, that allow control over the concatenation order, but only have an effect, if they are supported by the "code generator".

Hub with a last seen date {DV-4.3.2.5}

```
"fields": [
            {   "field_name": "TAILNUM",
                "field_type": "Varchar(20)",
                "targets": [{"table_name": "airline_hub"}]
            },
            {   "field_name": "LASTSEENDATE",
                "field_type": "TIMESTAMP",
                "field_value":"${CURRENT_TIMESTAMP}",
                "targets": [
                       {"table_name": "airline_hub"
                       , "exclude_from_key_hash":true
                       , "update_on_every_load":true}
                       ]
            },
              ...
        ],
"tables": [
            {   "table_name": "airplane_hub",
                "table_stereotype": "hub",
                "hub_key_column_name": "HK_AIRPLANE"
            }

              ...
        ]
```

Notes:
By setting "exclude_from_key_hash" for LASTSEEDATE, the field will not be classified as business key of the hub.

Multiple hubs with partly the same busiiness key

Here for both hub's the busiiness key must contain the COMPANY ID to be unique. Also, the incoming field ARCHITECT contains an employee identification and must be mapped to the business key column EMPLOYEE_ID

```
"fields": [
            {   "field_name": "COMPANY_ID",
                "field_type": "Varchar(20)",
                "targets":[ {"table_name": "employee_hub"},
                            {"table_name": "building_hub"} ]
            },
            {   "field_name": "ARCHITECT",
                "field_type": "integer",
                "targets":[ {"table_name": "employee_hub",
                             "column_name":"EMPLOYEE_ID"
                            } ]
            },
            {   "field_name": "BUILDING_SIGNATURE",
                "field_type": "Varchar(100)",
                "targets":[ {"table_name": "building_hub"} ]
```

```
            }
              ...
        ],
"tables": [
            {   "table_name": "employee_hub",
                "table_stereotype": "hub",
                "hub_key_column_name": "HK_EMPLOYEE"
            }
            {   "table_name": "building_hub",
                "table_stereotype": "hub",
                "hub_key_column_name": "HK_BUILDING"
            }
              ...
        ]
```

Notes:
The field "ARCHITECT" will be mapped to the business key column "EMPLOYEE_ID" in the employee_hub. Renaming is necessary, when the incoming data uses different field names for foreign keys. There can be only one column name in the hub.

Link tables

Link connecting two hubs {DV-4.3}

An order and its related customer

```
"fields": [
            {   "field_name": "ORDER_ID",
                "field_type": "integer",
                "targets":[ {"table_name": "order_hub"} ]
            },
            {   "field_name": "CUSTOMER_ID",
                "field_type": "Varchar(20)",
                "targets":[ {"table_name": "customer_hub"} ]
            }
        ],
"tables": [
            {   "table_name": "order_customer_link",
                "table_stereotype": "lnk",
                "link_parent_tables": ["order_hub","customer_hub"]
            },
            {   "table_name": "order_hub",
                "table_stereotype": "hub",
                "hub_key_column_name": "HK_ORDER"
            },
            {   "table_name": "customer_hub",
                "table_stereotype": "hub",
                "hub_key_column_name": "HK_CUSTOMER"
            }
        ]
```

Notes:
Just by declaring the two hubs to be the parents of the link, the link will inherit the key hashes of the hub and use the business keys of the hubs for hash calculation.

As with hashes for hubs, there must be a ruleset in the code or code generator, in wich order the business keys must be concatenated. There are also optional keywords in DVDP available, that allow explicit control of that ordering (if supported by the used generator).

Link connecting three hubs {DV-4.4.2}

Example of a car rental relation. A rental references a customer, a car and a rental station.

```
"fields": [
            {   "field_name": "VEHICLE_ID",
                "field_type": "integer",
                "targets": [ {"table_name": "car_hub"} ]
            },
            {   "field_name": "CUSTOMER_ID",
                "field_type": "Varchar(20)",
                "targets": [{"table_name": "customer_hub"}]
            },
            {   "field_name": "STATION_NAME",
                "field_type": "Varchar(20)",
                "targets": [{"table_name": "station_hub"}]
            }
        ],
"tables": [
            {   "table_name": "car_customer_station_link",
                "table_stereotype": "lnk",
                "link_parent_tables": [ "car_hub",
                                        "customer_hub",
                                        "station_hub"]
            },
            {   "table_name": "car_hub",
                "table_stereotype": "hub",
                "hub_key_column_name": "HK_CAR"
            },
            {   "table_name": "customer_hub",
                "table_stereotype": "hub",
                "hub_key_column_name": "HK_CUSTOMER"
            },
            {   "table_name": "station_hub",
                "table_stereotype": "hub",
                "hub_key_column_name": "HK_STATION"
            },
        ]
```

Link with two hubs, but one with two relations {DV-4.4.4}

The link describes a flight, referencing the filght carrier and two airports (1 source and 1 destination).

```
  "fields": [
              {   "field_name": "CARRIER_ID",
                  "field_type": "integer",
                  "targets": [{"table_name": "carrier_hub"}]
              },
              {   "field_name": "SOURCEAIRPORT_ID",
                  "field_type": "Varchar(20)",
                  "targets":[{    "table_name": "airport_hub",
                                  "column_name": "AIRPORT_ID",
                                  "relation_name":"SOURCE"
                              } ]
              },
              {   "field_name": "DESTINATIONAIRPORT_ID",
                  "field_type": "Varchar(20)",
                  "targets":[{    "table_name": "airport_hub",
                                  "column_name": "AIRPORT_ID",
                                  "relation_name":"DEST"
                              } ]
              }
              ...
          ],
  "tables": [
              {   "table_name": "car_customer_station_link",
                  "table_stereotype": "lnk",
                  "link_parent_tables":[  {"table_name":"carrier_hub"},
                                          {"table_name":"airport_hub",
                                           "relation_name":"SOURCE"
                                          }
                                          {"table_name":"airport_hub",
                                           "relation_name":"DEST"
                                          } ]
              },
              {   "table_name": "carrier_hub",
                  "table_stereotype": "hub",
                  "hub_key_column_name": "HK_CARRIER"
              },
              {   "table_name": "airport_hub",
                  "table_stereotype": "hub",
                  "hub_key_column_name": "HK_AIRPORT"
              },
              ...
          ]
```

Notes

This uses the "full" link_parent_tables declaration syntax, that is needed for declaring multiple connection to the same hub and relation names. By declaring two connectons to airport_hub, this will generate a link table, containing the hub key of airport twice. Without any explicit naming, the name of the airport hub key columns in the link will be postfixed with the name of the relation. (HK_AIRPORT_SOURCE, HK_AIRPORT_DEST)

The relation_names used in the link parent declaration also need to be used in the field target mapping. There are two different fields mapped to the AIRPORT_ID Colunm in the airport_hub, one for each relation.

Link with dependent child key columns {DV-4.4.5.2}

In this case the selling month and year are placed as dependent child keys of the webshop sale report link.

```
"fields": [
            {   "field_name": "WEBSHOP_ID",
                "field_type": "integer",
                "targets": [{"table_name": "webshop_hub"}]
            },
            {   "field_name": "PRODUCT_ID",
                "field_type": "integer",
                "targets": [{"table_name": "product_hub"}]
            },
            {   "field_name": "SELLING_MONTH",
                "field_type": "integer",
                "targets": [{"table_name": "webshop_sale_report_link"}]
            },
            {   "field_name": "SELLING_YEAR",
                "field_type": "integer",
                "targets": [{"table_name": "webshop_sale_report_link"}]
            },
            ...
        ],
"tables": [
            {   "table_name": "webshop_sale_report_link",
                "table_stereotype": "lnk",
                "link_parent_tables": ["webshop_hub","product_hub"]
            },
            {   "table_name": "webshop_hub",
                "table_stereotype": "hub",
                "hub_key_column_name": "HK_WEBSHOP"
            },
            {   "table_name": "product_hub",
                "table_stereotype": "hub",
                "hub_key_column_name": "HK_PRODUCT"
            }
        ]
```

Notes:
Dependend child keys are declared just by mapping fields to link tables without any more options.

Link with last seen date {DV-4.4.5}

An order and its related customer. The last seen date is the timestamp of the load process. Its not part of the source, but added by the loading process. It is stored in the link, not relevant for the link hash key and must be updated every time. This behaviour is declared in the field and its mapping.

```
"fields": [
            {   "field_name": "ORDER_ID",
                "field_type": "integer",
```

```
                "targets": [{"table_name": "order_hub"}]
            },
            {   "field_name": "CUSTOMER_ID",
                "field_type": "Varchar(20)",
                "targets": [{"table_name": "customer_hub"}]
            },
            {   "field_name": "LASTSEENDATE",
                "field_type": "TIMESTAMP",
                "field_value":"${CURRENT_TIMESTAMP}",
                "targets": [{   "table_name": "order_customer_link",
                                "exclude_from_key_hash":true,
                                "update_on_every_load":true
                            }]
            },
        ],
"tables": [
            {   "table_name": "order_customer_link",
                "table_stereotype": "lnk",
                "link_parent_tables": ["order_hub","customer_hub"]
            },
            {   "table_name": "order_hub",
                "table_stereotype": "hub",
                "hub_key_column_name": "HK_ORDER"
            },
            {   "table_name": "customer_hub",
                "table_stereotype": "hub",
                "hub_key_column_name": "HK_CUSTOMER"
            }
        ]
```

Same As Link {DV-5.1.1}{DV-5.2.2}

The source for a "same as" Link contains the business keys of the main object and the business key of its duplicate. The following example loads a "Same as Link" that contains one hub key HK_PRODUCT for the master product and HK_PRODUCT_DUPLICATE for the duplicate.

```
"fields": [
            {   "field_name": "PRODUCT_ID",
                "field_type": "integer",
                "targets": [{"table_name": "product_hub"}]},
            {   "field_name": "SAME_PRODUCT_ID",
                "field_type": "integer",
                "targets": [{   "table_name": "product_hub",
                                "column_name":"PRODUCT_ID",
                                "relation_name": "DUPLICATE"
                            }]
            },
        ],
"tables": [
            {   "table_name": "product_duplicate_saslink",
                "table_stereotype": "lnk",
```

```
                "link_parent_tables":[  {   "table_name":"product_hub"},
                                        {   "table_name":"product_hub",
                                            "relation_name":"DUPLICATE"
                                        }]
            },
            {   "table_name": "product_hub",
                "table_stereotype": "hub",
                "hub_key_column_name": "HK_PRODUCT"
            }
        ]
```

Notes
To also load the SAME_PRODUCT_ID id to the product hub, its field mapping needs to declare an explicit relation name, to distinguish it from PRODUCT_ID. It also needs to declare the column name to be PRODUCT_ID. The same relation name must be declared in the mapping of link parent tables. This determines, in which hub key column in the link which hash value of must be placed. The link will generate its own hub key column name for the second column, from the original name and the name of the relation (if not told otherwise).

Link on Link {DV-5.2.1}

A link on a link is **not supported** by the core syntax of DVPD, since it is highly discouraged by Dan Linstedt. The Data Vault methodolgy describes ways to circumvent the need for it.

hierachical link {DV-5.2.3}

This is an example of a part containment hierarchy. The declaration is another variety of multiple parent relations to the same hub.

```
"fields": [
            {   "field_name": "PART_ID",
                "field_type": "integer",
                "targets": [{"table_name": "product_hub"}]
            },
            {   "field_name": "CONTAINING_PART_ID",
                "field_type": "integer",
                "targets": [    {   "table_name": "product_hub",
                                    "column_name":"PART_ID",
                                    "relation_name": "CONTAINED_BY"
                                }]
            },
        ],
"tables": [
            {   "table_name": "product_containment_hlink",
                "table_stereotype": "lnk",
                "link_parent_tables":[  {"table_name":"product_hub"},

{"table_name":"product_hub","relation_name":"CONTAINED_BY"
                                        }]
            },
```

```
            {   "table_name": "product_hub",
                "table_stereotype": "hub",
                "hub_key_column_name": "HK_PRODUCT"
            }
        ]
```

non historized link {DV-5.2.4}

This source delivers bookings, identified by booking_ids. Since bookings will neever be changed, they can be stored in a non historized link.

```
{"fields": [
            {   "field_name": "ACCOUNT_NO",
                "field_type": "integer",
                "targets": [{"table_name": "account_hub"}]
            },
            {   "field_name": "ACCOUNTANT_ID",
                "field_type": "varchar(20)",
                "targets": [{"table_name": "accountant_hub"}]
            },
            {   "field_name": "BOOKING_ID",
                "field_type": "varchar(22)",
                "targets": [{"table_name": "account_booking_tlink"}]
            },
            {   "field_name": "BOOKING_TIME",
                "field_type": "varchar(22)",
                "targets": [{"table_name": "account_booking_tlinksat"}]
            },
            {   "field_name": "AMOUNT",
                "field_type": "decimal(12,2)",
                "targets": [{"table_name": "account_booking_tlinksat"}]
            },

        ],
"tables": [
            {   "table_name": "account_booking_tlink",
                "table_stereotype": "lnk",
                "link_parent_tables": ["account_hub","accountant_hub"]
            },
            {   "table_name": "account_booking_tlinksat",
                "table_stereotype": "sat",
                "satellite_parent_table": "account_booking_tlink",
                "is_enddated": false,
                "has_deletion_flag":false,
                "compare_criteria":"key"
            }

            {   "table_name": "account_hub",
                "table_stereotype": "hub",
                "hub_key_column_name": "HK_ACCOUNT"
```

```
            },
            {   "table_name": "accountant_hub",
                "table_stereotype": "hub",
                "hub_key_column_name": "HK_ACCOUNTANT"
            }
        ]
}
```

Note:
The behaviour of a non historized link is defined by the elements and loading procedure of its satellite. Without historization, the satellited can ommit deletion flag and enddating, and the loading process only must check, if the key is already in the satellite to prevent duplication of data from a repeated data delivery.

nondescriptive link (single) {DV-5.2.5}

Like the simple link example, non descriptive links are expressed by leaving out satellites. To keep only the current relations and remove deprecated relations a deletion rule for the link is declared. The source delivers complete customer relations for all orders in the data increment, so the order is partition criteria.

This example focuses on only one link from the books Figure 5.17. Since (bofore 0.7.0) the DVPD core syntax only supports 1 stage table, loading both links in Figure 5.17 of the book, needs two pipelines.

```
"fields": [
            {   "field_name": "OFFERING_ID",
                "field_type": "integer",
                "targets": [{"table_name": "offering_hub"}]
            },
            {   "field_name": "CUSTOMER_ID",
                "field_type": "integer",
                "targets": [{"table_name": "customer_hub"}]
            }
        ],
"tables": [
            {   "table_name": "customer_offering_interest_ndlink",
                "table_stereotype": "lnk",
                "link_parent_tables": ["offering_hub","customer_hub"]
            },
            {   "table_name": "offering_hub",
                "table_stereotype": "hub",
                "hub_key_column_name": "HK_OFFERING"
            }
            {   "table_name": "customer_hub",
                "table_stereotype": "hub",
                "hub_key_column_name": "HK_CUSTOMER"
            }
        ]
"deletion_detection": {
            "procedure":"stage_comparison",
            "partitioning_fields": ["OFFERING_ID"],
            "deletion_rules":[
                {   "tables_to_cleanup":["customer_offering_interest_ndlink"],
```

```
                    "join_path":["offering_hub"]
                }
            ]

}
```

Notes:
By restricting the deletion detection to the current OFFERING_ID in the stage, incremental data delivery will not remove relations of offerings, that are not in the current data set.

nondescriptive link (multi) {DV-5.2.5}

> needs review

```
"fields": [
            {   "field_name": "OFFERING_ID",
                "field_type": "integer",
                "targets": [{"table_name": "offering_hub"}]
            },
            {   "field_name": "CUSTOMER_ID",
                "field_type": "integer",
                "targets": [{"table_name": "customer_hub"}]
            },
        ],
"tables": [
            {   "table_name": "customer_offering_interest_ndlink",
                "table_stereotype": "lnk",
                "link_parent_tables": ["offering_hub","customer_hub"]
            },
            {   "table_name": "offering_hub",
                "table_stereotype": "hub",
                "hub_key_column_name": "HK_OFFERING"
            }
            {   "table_name": "customer_hub",
                "table_stereotype": "hub",
                "hub_key_column_name": "HK_CUSTOMER"
            }
        ]
"deletion_detection": {
            "procedure":"stage_comparison",
            "partitioning_fields": ["OFFERING_ID"],
            "deletion_rules":[
                {"tables_to_cleanup":["customer_offering_interest_ndlink"]}
            ]
}
```

exploration link {DV-5.2.7}

An exploration link is declared like any other link by declaring the hubs, that are connected by the link and the link itself. The main difference to normal links comes from the sourcing of the business keys, that will be

selected from the raw vault.

> when a directive to take hub key values from the source dataset instead of recalculating it, has beend added in later versions, there will be an example here

Satellite tables

Normal Satellite on a hub {DV-4.5}

```
"fields": [
            {   "field_name": "AIRPORTID",
                "field_type": "integer",
                "targets": [{"table_name": "airport_hub"}]
            },
            {   "field_name": "RUNWAYLENGTH",
                "field_type": "DECIMAL(5,1)",
                "targets": [{"table_name": "airport_sat"}]
            },
            {   "field_name": "RUNWAYELEVATION",
                "field_type": "DECIMAL(10,0)",
                "targets": [{"table_name": "airport_sat"}]
            }
        ],
"tables": [
            {   "table_name": "airport_sat",
                "table_stereotype": "sat",
                "satellite_parent_table": "airport_hub",
                "diff_hash_column_name": "DIFF_AIRPORT_SAT"
            },
            {   "table_name": "airport_hub",
                "table_stereotype": "hub",
                "hub_key_column_name": "HK_AIRPORT"
            }

        ]
```

Notes:
Just by mapping RUNWAYLENGTH and RUNWAYELEVATION to the satellite table, they will be part of the column set, that is concatenated for calculating the diff hash. In the example there are no further declarations about the concatenation order of the diff hash. This implies, that there is some hard coded ordering rules applied, when the loading code is implemneted/generated (e.g. alphabethical order). Keep in mind, that there are properties, that allow control over the concatenation order, but only have an effect, if they are supported by the "code generator".

Multiple satellites on a hub (Splitting by rate of change) {DV-4.5.2.2}

Store the fast changing attributes of a product (price, priority) separate from the slow/never changing attribute (name, class). Just as reminder: A pipline only transforms one source object. So splitting by source system is achieved by using different pipelines serving different satellites.

```
"fields": [
            {   "field_name": "GTIN",
                "field_type": "Varchar(20)",
                "targets": [{"table_name": "product_hub"}]
            },
            {   "field_name": "NAME",
                "field_type": "Varchar(20)",
                "targets": [{"table_name": "product_slow_sat"}]
            },
            {   "field_name": "CLASS",
                "field_type": "Varchar(20)",
                "targets": [{"table_name": "product_slow_sat"}]
            },
            {   "field_name": "PRICE",
                "field_type": "Varchar(20)",
                "targets": [{"table_name": "product_fast_sat"}]
            },
            {   "field_name": "PRIORITY",
                "field_type": "Varchar(20)",
                "targets": [{"table_name": "product_fast_sat"}]}
        ],
"tables": [
            {   "table_name": "product_slow_sat",
                "table_stereotype": "sat",
                "satellite_parent_table": "customer_hub",
                "diff_hash_column_name": "DIFF_PRODUCT_SLOW_SAT"
            },
            {   "table_name": "product_fast_sat",
                "table_stereotype": "sat",
                "satellite_parent_table": "customer_hub","diff_hash_column_name":
"DIFF_PRODUCT_FAST_SAT"
            },
            {   "table_name": "product_hub",
                "table_stereotype": "hub",
                "hub_key_column_name": "HK_PRODUCT"
            }
        ]
```

Satellite with extract date {DV-4.5.3.5}

In case the extract date differs significantly from the loading date, it must be somehow part of the incoming data. To prevent historization of every new extraction, when the content has not changed, the extract date is excluded from the comparison during the satellite loading.

```
"fields": [
            {   "field_name": "AIRPORTID",
                "field_type": "integer",
                "targets": [{"table_name": "airport_hub"}]
            },
            {   "field_name": "RUNWAYLENGTH",
```

```
                "field_type": "DECIMAL(5,1)",
                "targets": [{"table_name": "airport_sat"}]
            },
            {   "field_name": "RUNWAYELEVATION",
                "field_type": "DECIMAL(10,0)",
                "targets": [{"table_name": "airport_sat"}]
            }
            {   "field_name": "EXTRACT_TIMESTAMP",
                "field_type": "TIMESTAMP",
                "targets": [{   "table_name": "airport_sat",
                                "exclude_from_change_detection":true
                            }]
            }
    ],
"tables": [
            {   "table_name": "airport_sat",
                "table_stereotype": "sat",
                "satellite_parent_table": "airport_hub",
                "diff_hash_column_name": "DIFF_AIRPORT_SAT"
            },
            {   "table_name": "airport_hub",
                "table_stereotype": "hub",
                "hub_key_column_name": "HK_AIRPORT"
            }
        ]
```

Satellite on a link, with a driving key declaration {DV-4.5.5}

This data source is a table with the order, referencing the product of the order. Should the order change to another product, the former product of the order must be "unlinked". Data Vault indicates this to the load operation by declaring driving keys, that must be used for ending former relations. Driving keys are the hub key columns of the parent link of the satellite.

```
"fields": [
            {   "field_name": "ORDER_ID",
                "field_type": "integer",
                "targets": [{"table_name": "order_hub"}]
            },
            {   "field_name": "PRODUCT_ID",
                "field_type": "Varchar(20)",
                "targets": [{"table_name": "product_hub"}]
            },
            {   "field_name": "PRICE",
                "field_type": "DECIMAL(16,2)",
                "targets": [{"table_name": "order_product_sale_sat"}]
            },
            {   "field_name": "QUANTITY",
                "field_type": "DECIMAL(8,0)",
                "targets": [{"table_name": "order_product_sale_sat"}]
            },
        ],
```

```
  "tables": [
              {   "table_name": "order_product_sale_sat",
                  "table_stereotype": "sat",
                  "satellite_parent_table": "order_product_link",
                  "driving_keys": ["HK_ORDER"],
                  "diff_hash_column_name": "DIFF_ORDER_PRODUCT_SALES_SAT"
              },
              {   "table_name": "order_product_link",
                  "table_stereotype": "lnk",
                  "link_parent_tables": ["order_hub","product_hub"]
              },
              {   "table_name": "order_hub",
                  "table_stereotype": "hub",
                  "hub_key_column_name": "HK_ORDER"
              },
              {   "table_name": "product_hub",
                  "table_stereotype": "hub",
                  "hub_key_column_name": "HK_PRODUCT"
              }
              ]
```

Multi-Active Satellite {DV-5.3.2}

The example stores telephone numbers for customers. Since there is no fixed set of telefone numbers a multi active satellite is the best solution.

```
  "fields": [
              {   "field_name": "CUSTOMER_ID",
                  "field_type": "Varchar(20)",
                  "targets": [{"table_name": "customer_hub"}]
              },
              {   "field_name": "PHONE_TYPE",
                  "field_type": "Varchar(20)",
                  "targets": [{"table_name": "customer_phone_msat"}]
              },
              {   "field_name": "PHONE_NUMBER",
                  "field_type": "VARCHAR(100)",
                  "targets": [{"table_name": "customer_phone_msat"}]
              }
          ],
  "tables": [
              {   "table_name": "customer_phone_msat",
                  "table_stereotype": "sat",
                  "is_multiactive":"true",
                  "satellite_parent_table": "customer_hub",
                  "diff_hash_column_name": "DIFF_CUSTOMER_PHONE_MSAT"
              },
              {   "table_name": "customer_hub",
                  "table_stereotype": "hub",
                  "hub_key_column_name": "HK_CUSTOMER"
```

```
            }
        ]
```

Note:
The only declaration difference to a normal satellite is the "is_multiactive=true" attribute that should trigger the specfic load processing of multi active statellites.

Since a diff hash needs to be calculated over all rows of the same busniess key, there is the additional challenge, to keep a stable order over the rows. Like with the column order for the diff hash, the ordering of the rows can be just a fixed rule. Again DVPD alos provides keywords to allow control over the row ordering.

Status Tracking Satellite with sequence information {Dv-5.3.3}

In this example, the CDC information contains a change timestamp from the source system. This data field is used to prevent reloading CDC information, that had already been received.

```
"fields": [
            {   "field_name": "EMPLOYEE_ID",
                "field_type": "Varchar(20)",
                "targets": [{"table_name": "employee_hub"}]
            },
            {   "field_name": "CHANGE_TIMESTAMP",
                "field_type": "timestamp",
                "targets": [{"table_name": "employee_tracksat"}]
            },
            {   "field_name": "OPERATION",
                "field_type": "Varchar(3)",
                "targets": [{"table_name": "employee_tracksat"}]
            }
        ],
"tables": [
            {   "table_name": "employee_tracksat",
                "table_stereotype": "sat",
                "satellite_parent_table": "employee_hub",
                "compare_criteria":"key+data",
                "is_enddated" : "false"
                "uses_diff_hash":"false",
                "has_deletion_flag":"false"
            },
            {   "table_name": "employee_hub",
                "table_stereotype": "hub",
                "hub_key_column_name": "HK_EMPLOYEE"
            }
        ]
```

Note:

- The compare criteria prevent double insertion of the same information.
- Deletion flag and enddate are removed, since this satellite conatains events and not states

- Disabling the diff hash in this satellite, is more a performance optimization then really necessary. Without a diff hash, the columns of the satelltie must be compared directly. This can be much faster, when there are only two columns with a total witdh of 11 Bytes (against 20 Bytes for a sha-1 hash). leaving out the deletion flag, enddate and diff hash column, this is a structural indicator for the user of the data, that this there is some special data in the satellite.

Tracking satellite without sequence information

In this example, the CDC information contains no data about the sequence of events. Incoming events must always be stored. The load timestamp in the satellite will be the only indicator about the sequence of events. Duplication of data by reloading the same events must be prevented by the loading process.

```
"fields": [
            {   "field_name": "PRODUCT_ID",
                "field_type": "Varchar(20)",
                "targets": [{"table_name": "product_hub"}]
            },
            {   "field_name": "OPERATION",
                "field_type": "Varchar(3)",
                "targets": [{"table_name": "product_tracksat"}]
            }
        ],
"tables": [
            {   "table_name": "product_tracksat",
                "table_stereotype": "sat",
                "satellite_parent_table": "product_hub",
                "compare_criteria":"none"
            },
            {   "table_name": "product_hub",
                "table_stereotype": "hub",
                "hub_key_column_name": "HK_PRODUCT"
            }
        ]
```

Effectivity satellite on a link {DV-5.3.4}

In this classical example an order as a reletion to a produdt. Should the order by changed to a different product, an effectivty satellite must track the validitiy of the relation over time.

```
"fields": [
            {   "field_name": "ORDER_ID",
                "field_type": "integer",
                "targets": [{"table_name": "order_hub"}]
            },
            {   "field_name": "PRODUCT_ID",
                "field_type": "Varchar(20)",
                "targets": [{"table_name": "product_hub"}]
            }
    ],
```

```
  "tables": [
              {  "table_name": "order_product_sale_esat",
                 "table_stereotype": "sat",
                 "satellite_parent_table": "order_product_link",
                 "driving_keys": ["HK_ORDER"]
              },

              {   "table_name": "order_product_link",
                 "table_stereotype": "lnk",
                 "link_key_column_name": "LK_order_product",
                 "link_parent_tables": ["order_hub","product_hub"]
              },
              {   "table_name": "order_hub",
                 "table_stereotype": "hub",
                 "hub_key_column_name": "HK_ORDER"
              },
              {   "table_name": "product_hub",
                 "table_stereotype": "hub",
                 "hub_key_column_name": "HK_PRODUCT"
              }
          ]
```

Notes:
An effectivity satellite is a sattellite on a link without any field mappings. The compiler will detect this and provide this observation in the is_effectivity_sat as a table property in the DVPI.

Effectivity satellite on a link with membership columns {DV-5.3.4}

Here the source provides a field of the start date of the relation between a customer and a loaylty programm. In case of a change, the new loaylty program and the new start date will be in the row. This kind of date information is called membership data and for quiering the data later, it is very conveniant to provide a second column, that contains the date, when the membership ends.

```
  "fields": [
              {   "field_name": "CUSTOMER_ID",
                 "field_type": "integer",
                 "targets": [{"table_name": "customer_hub"}]
              },
              {   "field_name": "LOYALTYPROGRAM_NAME",
                 "field_type": "Varchar(20)",
                 "targets": [{"table_name": "loyaltyprogram_hub"}]
              },
              {   "field_name": "START_DATE",
                 "field_type": "date",
                 "targets": [{"table_name": "loyaltyprogram_customer_mbsat"}]},
      ],
  "tables": [
              {   "table_name": "loyaltyprogram_customer_mbsat",
                 "table_stereotype": "sat",
                 "satellite_parent_table": "loyaltyprogram_customer_link",
                 "driving_keys": ["HK_CUSTOMER"],
```

```
                "membership_end_columns":   [
                        {"column_name":"END_DATE",
                         "membership_start_column":"START_DATE"}]
            },
            {   "table_name": "loyaltyprogram_customer_link",
                "table_stereotype": "lnk",
                "link_key_column_name":"LK_LOYALTYPROGRAM_CUSTOMER",
                "link_parent_tables": ["order_hub","product_hub"],
            },
            {   "table_name": "customer_hub",
                "table_stereotype": "hub",
                "hub_key_column_name": "HK_CUSTOMER"
            },
            {   "table_name": "loyaltyprogram_hub",
                "table_stereotype": "hub",
                "hub_key_column_name": "HK_LOALTYPROGRAM"
            }
        ]
```

Note:
(this syntax is not supported by the compiler yet)
The declaration "membership_end_columns" add new columns to keep the membership ending. By connectiing the end column to a start column, every thing is declared for the necessary update step of the load operation.

Normalized record tracking satellites {DV-5.3.5}

Record tracking satellites just keep a record for every load process and hub/link keys, that are in the current load.

```
"fields": [
            {   "field_name": "PRODUCT_ID",
                "field_type": "Varchar(20)",
                "targets": [{"table_name": "product_hub"}]
            }
            {   "field_name": "APPERANCE",
                "field_type": "integer",
                "targets": [{"table_name": "product_rectracksat"}]
            }
    ],
"tables": [
            {   "table_name": "product_rectracksat",
                "table_stereotype": "sat",
                "satellite_parent_table": "product_hub",
                "compare_criteria":"none"
            },
            {   "table_name": "product_hub",
                "table_stereotype": "hub",
                "hub_key_column_name": "HK_PRODUCT"
            }
        ]
```

Note:
Record tracking information must be inserted every time we get it. This is achieved by setting the compare criteria to "none".

Normalized record tracking satellites keeping only last record

This is an alternative to a hub with a last seen timestampt. Like a tracking satellite every key of the current load set is inserted, but older rows for the key are removed.

```
"fields": [
            {   "field_name": "PRODUCT_ID",
                "field_type": "Varchar(20)",
                "targets": [{"table_name": "product_hub"}]
            },
            {   "field_name": "APPERANCE",
                "field_type": "integer",
                "targets": [{"table_name": "product_rectracksat"}]
            }
        ],
"tables": [
            {   "table_name": "product_rectracksat",
                "table_stereotype": "sat",
                "satellite_parent_table": "product_hub",
                "compare_criteria":"none",
                "history_depth_criteria":"versions",
                "history_depth_limit":0
            },
            {   "table_name": "product_hub",
                "table_stereotype": "hub",
                "hub_key_column_name": "HK_PRODUCT"
            }
        ]
```

The reduction to a single entry for every key is achievied via the history_depth... declarations.

"ref" tables

"ref" simple {DV-6.3.1}

```
"fields": [
            {   "field_name": "ISO_3166_ALPHA_2",
                "field_type": "Varchar(2)",
                "targets": [{"table_name": "country_code_ref"}]
            },
            {   "field_name": "ISO_3166_ALPHA_3",
                "field_type": "Varchar(3)",
                "targets": [{"table_name": "country_code_ref"}]
            },
```

```
            {   "field_name": "ISO_3166_NUMERIC",
                "field_type": "Varchar(3)",
                "targets": [{"table_name": "country_code_ref"}]
            }
    ],
"tables": [
            {   "table_name": "country_code_ref",
                "table_stereotype": "ref",
                "is_enddated": "false"
            },
        ]
```

Note:
By setting is_enddated=false, a reference table will not be able to historize rows. The load process needs to remove data, that is not in the incoming data set.

"ref" historized (hub/sat) {DV-6.3.2}

This approach uses a hub satellite model to historize data. This will only be used for reference tables with a specifc join key.

```
"fields": [
            {   "field_name": "DAY_DATE",
                "field_type": "DATE",
                "targets": [{"table_name": "calendar_refhub"}]
            },
            {   "field_name": "FISCAL_YEAR",
                "field_type": "integer",
                "targets": [{"table_name": "calendar_refsat"}]
            },
            {   "field_name": "FISCAL_QUATER",
                "field_type": "integer",
                "targets": [{"table_name": "calendar_refsat"}]
            }
    ],
"tables": [
            {   "table_name": "calendar_refsat",
                "table_stereotype": "sat",
                "satellite_parent_table": "calendar_refhub",
                "uses_diff_hash":false
            },
            {   "table_name": "calendar_refhub",
                "table_stereotype": "hub",
                "hub_key_column_name": "HK_calendar_ref"
            }
        ]
```

Note
: From the perspective of the DVPD and loading operation, there is no diference to a normal hub/sat topology. The name of the data vault tables should indicate the classification as reference data.

Also it might be helpful to present the reference data as prejoined and prefiltered view. This allows clearer reading of using queries.

"ref" historized (single table)

This is a non historized reference table extended by an enddate.

```
"fields": [
            {   "field_name": "ISO_3166_ALPHA_2",
                "field_type": "Varchar(2)",
                "targets": [{"table_name": "country_code_ref"}]
            },
            {   "field_name": "ISO_3166_ALPHA_3",
                "field_type": "Varchar(3)",
                "targets": [{"table_name": "country_code_ref"}]
            },
            {   "field_name": "ISO_3166_NUMERIC",
                "field_type": "Varchar(3)",
                "targets": [{"table_name": "country_code_ref"}]
            }
        ],
"tables": [
            {   "table_name": "country_code_ref",
                "table_stereotype": "ref",
                "diff_hash_column_name": "DIFF_COUNTRY_CODE_REF"
            },
        ]
```

Note:
The enddate is set by the loadprocess, when the specific value combination is not present in the source.

"ref" - code and descriptions {DV-6.3.3}

The code and description reference table approach, has a source agnostic stucture, that is used for multiple sources.

GROUP contains a sting constant, that identifies all rows, for the same "Codepage"

CODE contains the code, that will be translated into a text, that is stored in DESCRIPTION.

```
"fields": [
            {   "field_name": "GROUP",
                "field_type": "Varchar(20)",
                "field_value": "operations_code",
                "fiels_value":"StdOpCode",
                "targets": [{"table_name": "codes_ref"}]
            },
            {   "field_name": "STANDARD_OPERATIONS_CODE",
                "field_type": "Varchar(20)",
                "targets": [{   "table_name": "codes_ref",
```

```
                                "column_name":"CODE"
                            }]
            },
            {   "field_name": "OPERATION_NAME",
                "field_type": "Varchar(255)",
                "targets": [{   "table_name": "codes_ref",
                                "column_name":"DESCRIPTION"
                            }]
            }
    ],
"tables": [
            {   "table_name": "codes_ref",
                "table_stereotype": "ref",
                "is_enddated": "false"
            },
        ]
"deletion_detection": {
            "procedure":"stage_comparison",
            "partitioning_fields": ["GROUP"],
            "deletion_rules":[
                {"tables_to_cleanup":["codes_ref"]}
            ]
        }
```

Note:
The field value declaration defines the "code_page" this pipeline will store its source in.

The deletion detection declares the GROUP as partitioning field. This restricts the deletions detection to rows in the group that is currently loaded.

"ref" - code and descriptions, historized (hub/sat) {DV-6.3.3.1}

```
"fields": [
            {   "field_name": "GROUP",
                "field_type": "Varchar(20)",
                "fiels_value":"StdOpCode",
                "targets": [{"table_name": "code_hub"}]
            },
            {   "field_name": "STANDARD_OPERATIONS_CODE",
                "field_type": "Varchar(20)",
                "targets": [{   "table_name": "code_hub",
                                "column_name":"CODE"
                            }]
            },
            {   "field_name": "OPERATION_NAME",
                "field_type": "Varchar(255)",
                "targets": [{"table_name": "code_sat",
                                "column_name":"DESCRIPTION"
                            }]
            }
    ],
```

```
  "tables": [
              {   "table_name": "code_hub",
                  "table_stereotype": "hub",
                  "hub_key_column_name": "HK_CODE"
              },
              {   "table_name": "code_sat",
                  "table_stereotype": "sat",
                  "satellite_parent_table": "code_hub",
                  "uses_diff_hash":false
              }
          ]
  "deletion_detection": {
              "procedure":"stage_comparison",
              "partitioning_fields": ["GROUP"],
              "deletion_rules":[
                  {   "tables_to_cleanup":["code_sat"],
                      "join_path":["code_hub"]
                  }
              ]
          }
```

"ref" - code and descriptions, historized (single table)

This is a non historized reference table extended by an enddate. The enddate is set by a load run, where the specific value combination of the group is not present in the source. Restrict enddating to the group is done with the "deletion_detection" declaration.

```
  "fields": [
              {   "field_name": "GROUP",
                  "field_type": "Varchar(20)",
                  "fiels_value":"StdOpCode",
                  "targets": [{"table_name": "codes_ref"}]
              },
              {   "field_name": "STANDARD_OPERATIONS_CODE",
                  "field_type": "Varchar(20)",
                  "targets": [{   "table_name": "codes_ref",
                                  "column_name":"CODE"
                              }]
              },
              {   "field_name": "OPERATION_NAME",
                  "field_type": "Varchar(255)",
                  "targets": [{   "table_name": "codes_ref",
                                  "column_name":"DESCRIPTION"
                              }]
              }
      ],
  "tables": [
              {   "table_name": "codes_ref",
                  "table_stereotype": "ref",
                  "diff_hash_column_name": "DIFF_CODES_REF"
              },
```

```
        ]
"deletion_detection": {
            "procedure":"stage_comparison",
            "partitioning_fields": ["GROUP"],
            "deletion_rules":[
                {"tables_to_cleanup":["codes_ref"]}
            ]
        }
```

Snapshot tables

The syntax for snapshot tables (point in time tables, bridge tables) will be added in future versions. From the current perspective this will be a new stereotype "snapshot" with properties to declare the participating links and satellites, the definition of the source for the timeline, the management of the time window and granularity and the structural assets (naming of the columns, columnset for every connected asset (loaddate, key, flag))

Point in time tables {DV-6.1}

to be defined later - will be a flavour of a snapshot table

Bridge tables {DV-6.2}

to be defined later - will be a flavour of a snapshot table