WHITE PAPER

Layering the Qlik Data Integration Architecture with Compose Projects

Best Practices for Compose for Data Warehouses

LEAD WITH DATA™ Qlik

TABLE OF CONTENTS

SUMMARY

- In most Data Warehouse implementation scenarios, a standard QDI configuration having a single product Compose Data Warehouse project is sufficient for a complete solution.
- In some instances, layering the architecture by chaining Compose projects can provide additional flexibility by breaking down areas of the implementation supported by chaining of Primary and Secondary Data Warehouses.
- The additional flexibility needs to be carefully weighed against additional implementation customizations.

INTRODUCTION

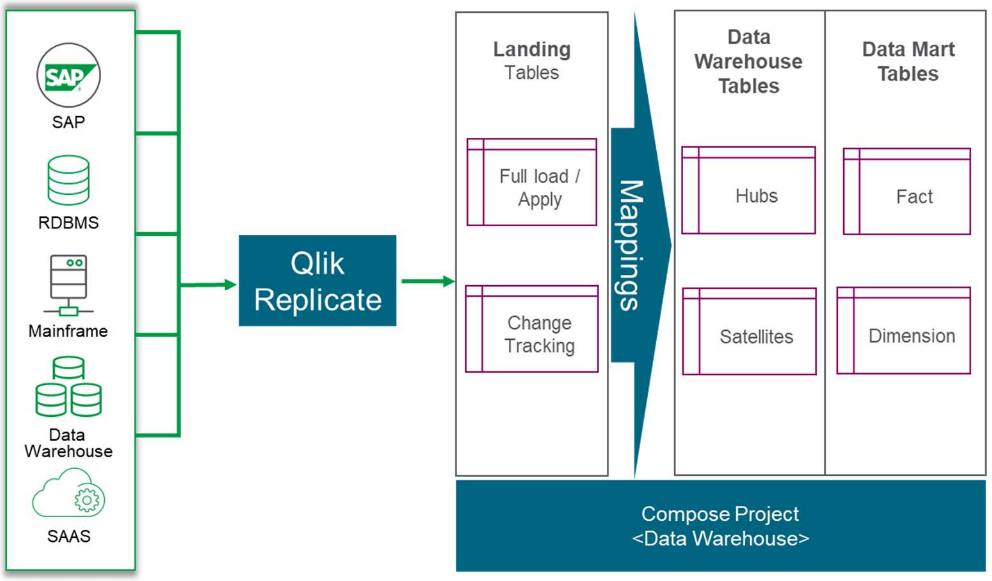
The Qlik Data Integration pipeline is powerful yet easy to implement to enable a complete Data Warehouse solution with minimal customization. The automation, both Replicate for data movement and Compose for data transformation to an analytics ready presentation layer, simplifies the tasks and effort require to deploy and operate the Data Warehouse environment.

In addition, Compose projects themselves can be further linked or chained to achieve enhanced flexibility of the resulting pipeline. This would be the case if for example we are looking to implement a super set Primary Data Warehouse (akin to a relational data lake) as first step with a Secondary Data Warehouse using a subset of the Primary as the second step. Another example would be implementing a subject area specific Data Warehouse, with very granular level of detail about its respective area, as a first step and Secondary DW that consolidates multiple subject area specific Data Warehouses at a higher level with less detail per subject. This flexibility requires task customizations and tighter dependencies between Compose projects and should be carefully evaluated against the status quo option of a single Compose project.

This paper will describe the considerations needed to implement the scenarios specified and how layering the architecture using multiple Compose projects solves for each of them.

Architecture Use Cases

A standard Qlik Compose project manages a single Data Warehouse as depicted below



In this scenario, Replicate acts as a feeder into the Landing Area of the Data Warehouse with Compose model and mappings populating the warehouse (AKA vault) table and ultimately the Data Mart tables.

In most cases this architecture suffices and it’s the preferred approach to implementing the requirements of the Data Warehouse. Particularly if we need to both minimize the amount of customization and effort to deploy as well as optimize for near real time needs.

Primary Data Warehouse/Relational Data Lake

There are use cases however that require additional layers in the architecture to implement. The first of these is the super set Primary Data Warehouse, A.K.A Relational Data Lake. The logical architecture is depicted below.

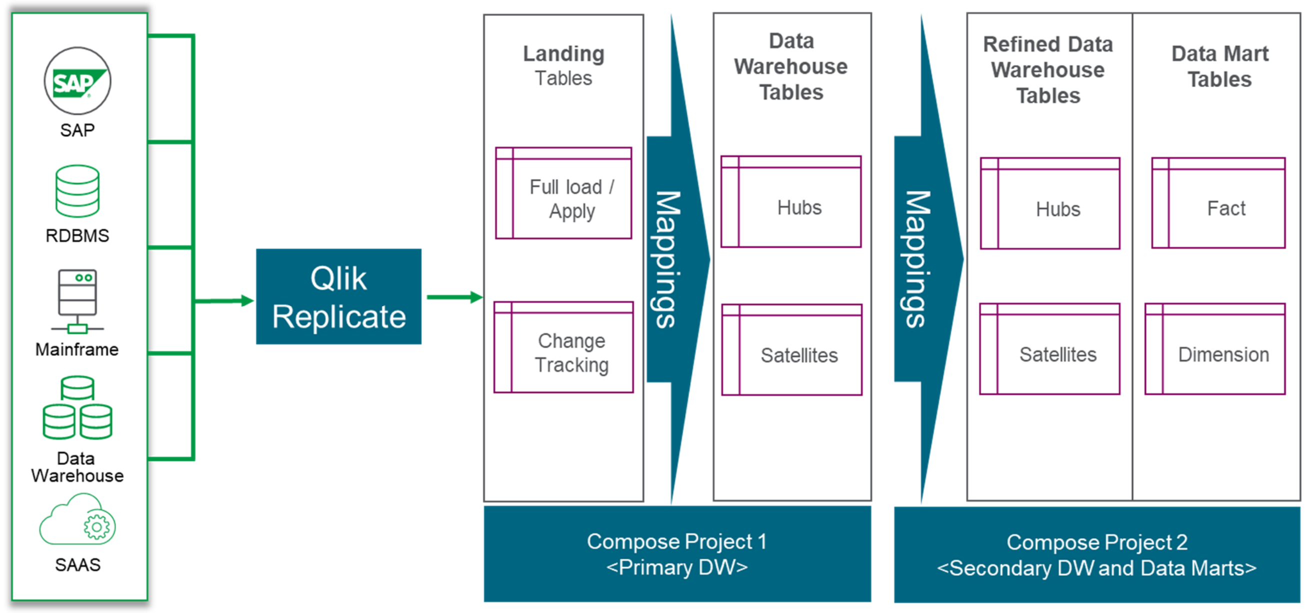


Figure 1 – Relational Data Lake Architecture

In this use case, we treat the Primary Data Warehouse, managed by the first Compose project, as a raw data area or zone. The Primary Data Warehouse just stores detailed data and potentially all its history but may not at this point have defined relationships between entities or even know appropriate dimensional representation for them. The Secondary Data Warehouse and project(s) would implement a more refined model utilizing a subset of the data in the Primary and defining appropriate relationships between entities to be leveraged by the resulting Data Mart in this project.

Benefits

One of the main benefits of this architectural option is the ability to defer modeling decisions to a later point while immediately capturing relevant data and history. It allows for quicker onboarding of data in the Data Warehouse and to create separate raw and refined zones.

Subject Area/Generalized Data Warehouse

The second use case for this layering architecture allows for implementing detailed subject area specific Data Warehouse(s) then consumed by a generalized Data Warehouse. The logical architecture is depicted below.

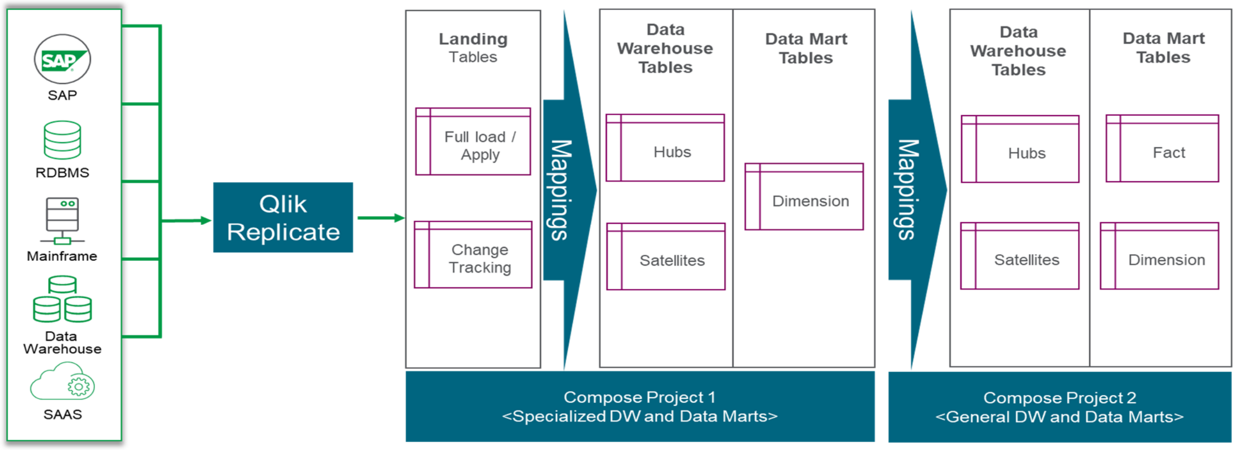


Figure 2 - Subject Area Data Warehouse Architecture

In this instance the subject area Data Warehouse and corresponding Compose project might contain highly normalized models for each subject area with higher level denormalized structure implemented as dimensions in the Data Mart for the subject area project. The generalized Data Warehouse and Compose project then implement a model where the data mart dimensions of the subject area become the entities of the general model with additional relationships across subject areas specified in it. For instance, large manufacturing firms could create an inventory and materials subject area warehouse to manage all the details related to products that themselves contain thousands for components while also having dedicated subject area warehouses for other functions such as Human Resources, Finance, etc. with similar level of detail. Each of these Subject Area Data Warehouses would generate a Data Mart with refined dimensions that contain only the detail needed at the cross-subject area view of the enterprise that would be consumed by the General Data Warehouse for higher level analytics/

Benefits

One of the main benefits of this architectural option is the ability to specialize complex subject area processing and details containing hundreds of entities and relationships and offering this detail to those that need it while also offering a cross subject area view of the enterprise with less detail and broader view of the enterprise.

Architectural considerations

Both options discussed tradeoff increased flexibility for increased implementation complexity, explained in detail in the next section, and potential runtime latency, due to having lag between Primary and dependent project execution. Implementors should carefully weigh these and consider the standard implementation (one Compose project/Data Warehouse) as the default recommendation.

Qlik Compose ETL Tasks

A traditional Compose project relies on two different types of ETL tasks for ingesting data. These are known as full load tasks and CDC tasks.   ETL Tasks are a collection of mappings that run together. The task type defines data source and processing characteristics within Compose and impacts how Compose automates the Extract, Load and Transform (ELT) code. A comparison of the two task types is defined in the table below.

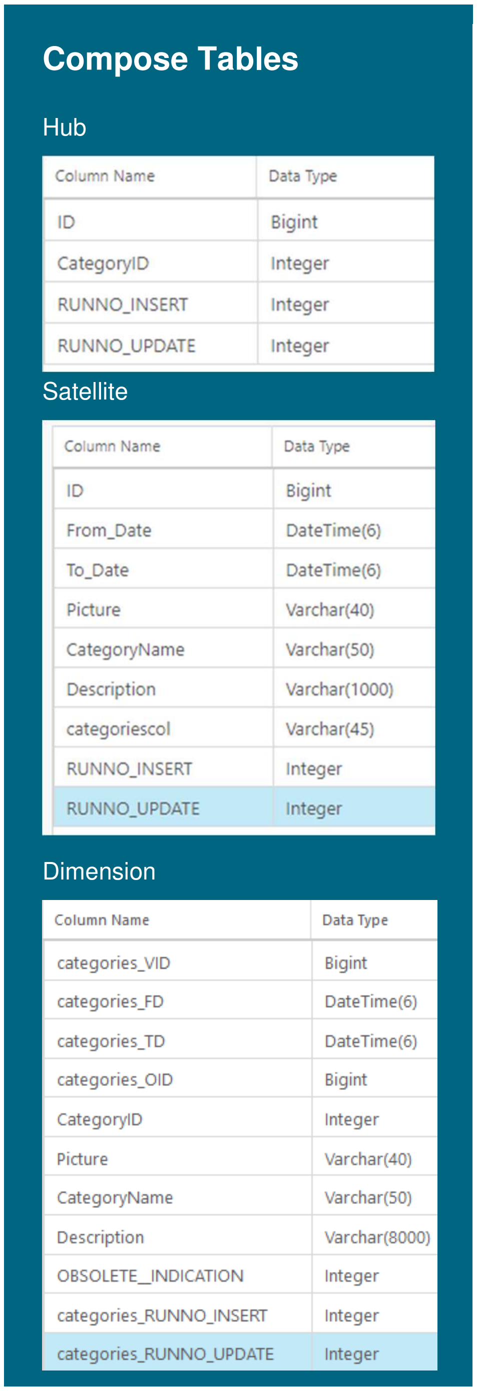
	Full Load Task	CDC Task
What It Does	Processes a full data set, comparing it to the data warehouse. The task detects new and changed records, processing them based on the model's characteristics (type 1 and type 2 attributes)	Leverages Replicates STORE CHANGES option and uses the change tracking (__CT) tables to only process new changes instead of the full data set. This task still compares the changed records to the data warehouse to ensure the change is applicable to the data warehouse model.
When To Use It	‣ Initial load of the data warehouse ‣ End of day or batch based processes that require processing a complete dataset ‣ Mappings using query or views as a source ‣ Custom Incremental load processes	‣ CDC based processing for Replicate data sources ‣ Intra-day / near real-time data loads ‣ Batch data loads driven from CDC delivered data
Limitations	‣ Out of the box workload processes the complete table / view / query result set	‣ Not applicable for mappings using a query or view as the source
Data Source	‣ Table (full load for Replicate delivered data) ‣ Queries ‣ Views	‣ Change Tracking tables (Compose appends __CT to the mappings defined source

In either of the layering use cases, the Compose project(s) in the first layer would utilize either of these two task types to load and manage data. The Compose project(s) on the second layer however will utilize a customized Full Load Task mappings to load data from first layer project(s) Data Warehouse Hub/Satellite or Data Mart dimension tables. These customized mappings must contain additional processing criteria to pick up incremental changes to tables in the Primary or Subject area warehouse.

Custom Change Detection

Qlik Compose generates two types of structures in its data warehouse (vault) area, Hubs, which contain business keys and type 1 attributes, and Satellites which contain type 2 attributes and corresponding effective dates (FromDate and ToDate). In the Data Mart sections, dimensions can be configured to be type 1 or type 2. Any of these table structures contains process columns, maintained by Compose, that tag or assign to each record the individual Compose run number (RUNNO) that inserted or update the record (*RUNNO_INSERT*) and (*RUNNO_UPDATE*). Examples of these are depicted below.

Both architectural approaches leverage the RUNNO_UPDATE columns for use by the chained or dependent projects to process only new or changed records into the corresponding warehouse.



Compose Tables

Hub

Column Name	Data Type
ID	Bigint
CategoryID	Integer
RUNNO_INSERT	Integer
RUNNO_UPDATE	Integer

Satellite

Column Name	Data Type
ID	Bigint
From_Date	DateTime(6)
To_Date	DateTime(6)
Picture	Varchar(40)
CategoryName	Varchar(50)
Description	Varchar(1000)
categoriescol	Varchar(45)
RUNNO_INSERT	Integer
RUNNO_UPDATE	Integer

Dimension

Column Name	Data Type
categories_VID	Bigint
categories_FD	DateTime(6)
categories_TD	DateTime(6)
categories_OID	Bigint
CategoryID	Integer
Picture	Varchar(40)
CategoryName	Varchar(50)
Description	Varchar(8000)
OBSOLETE_INDICATION	Integer
categories_RUNNO_INSERT	Integer
categories_RUNNO_UPDATE	Integer

Processing changes in dependent/chained warehouse

A control table, housed within the dependent/chained project warehouse tables is needed to maintain the state with respect to processed changes. This method maintains the last processed RUNNO, and the source projects max completed RUNNO in a table which is then used in mapping filters to ensure the dependent/chained project only processes new or changed records.

Pre-Load ETL and Post-Load ETL steps are implemented to track the processed and source max values for RUNNO along with a filter for each query / table being processed. The control table can have multiple records to segment ETL processing in dependent/chained project if required.

Below is an example control/tracking table to manage the RUNNO

CONFIG_NAME	PROCESSED_RUNNO	SRC_MAX_RUNNO
SRC1_ETLSET1	2	4
SRC2_ETLSET1	3	4

CONFIG_NAME column provides a method of managing multiple RUNNO's in the event multiple landing areas are used to source Project2 entities.

PROCESSED_RUNNO – the max RUNNO that has been processed by dependent/chained project

SRC_MAX_RUNNO – the max RUNNO that the source project has completed processing

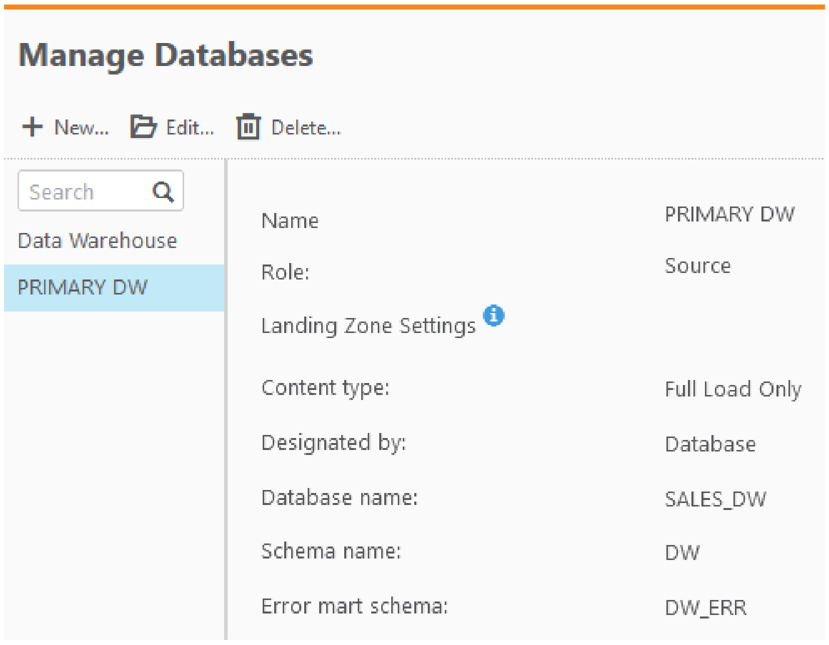
Within a Compose project, the ETL RUNNO are managed in the logging table `TPIL_RUNS.` This table in the Primary project can be used to determine the last completed RUNNO in the source project and update the control table in the Secondary/dependent project.

Primary/Secondary Data Warehouse Implementation

The premise of this sample implementation is a Primary Data Warehouse that contains our relational lake. It takes one such entity in the relational lake and uses its data including history as a foundation for the entity in the Dependent Data Warehouse.

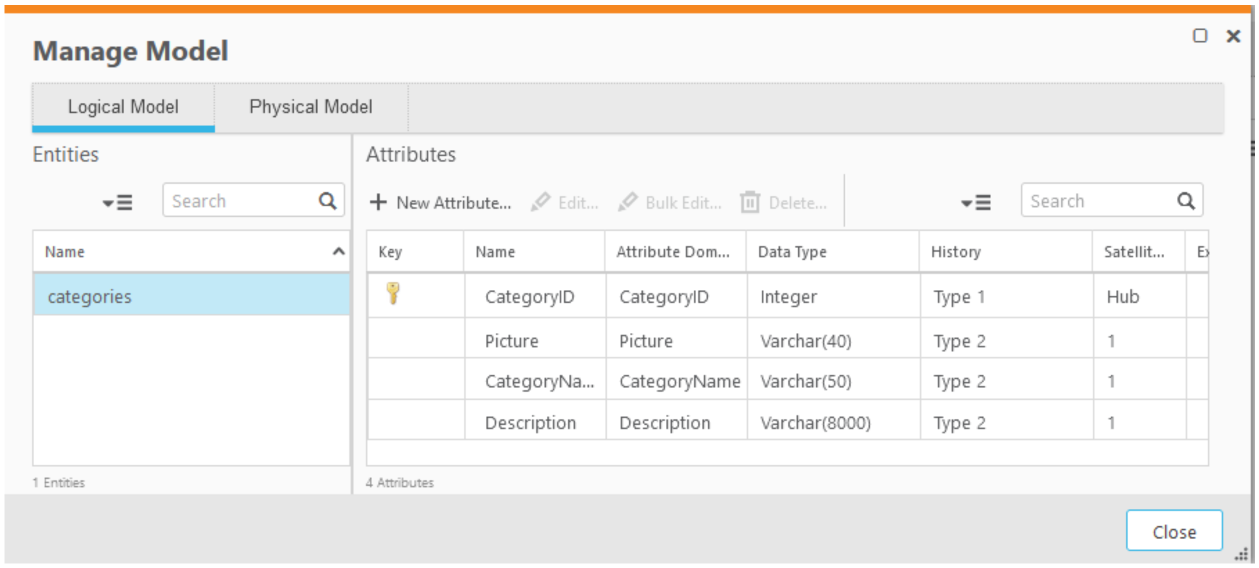
1. Configure Compose project for Primary Data Warehouse as usual with only a Data Warehouse layer and standard table based Full/Bulk Load and CDC tasks that load from Replicate landing area.

2. Setup the dependent/chained project landing area database(s) with a Full Load Only source connection to the Primary project DW database.



Name	PRIMARY DW
Role:	Source
Landing Zone Settings	
Content type:	Full Load Only
Designated by:	Database
Database name:	SALES_DW
Schema name:	DW
Error mart schema:	DW_ERR

3. Setup the dependent/chained project Model with the entities that will be sourced from the Primary project


Manage Model

Logical Model | Physical Model

Entities

Name
categories

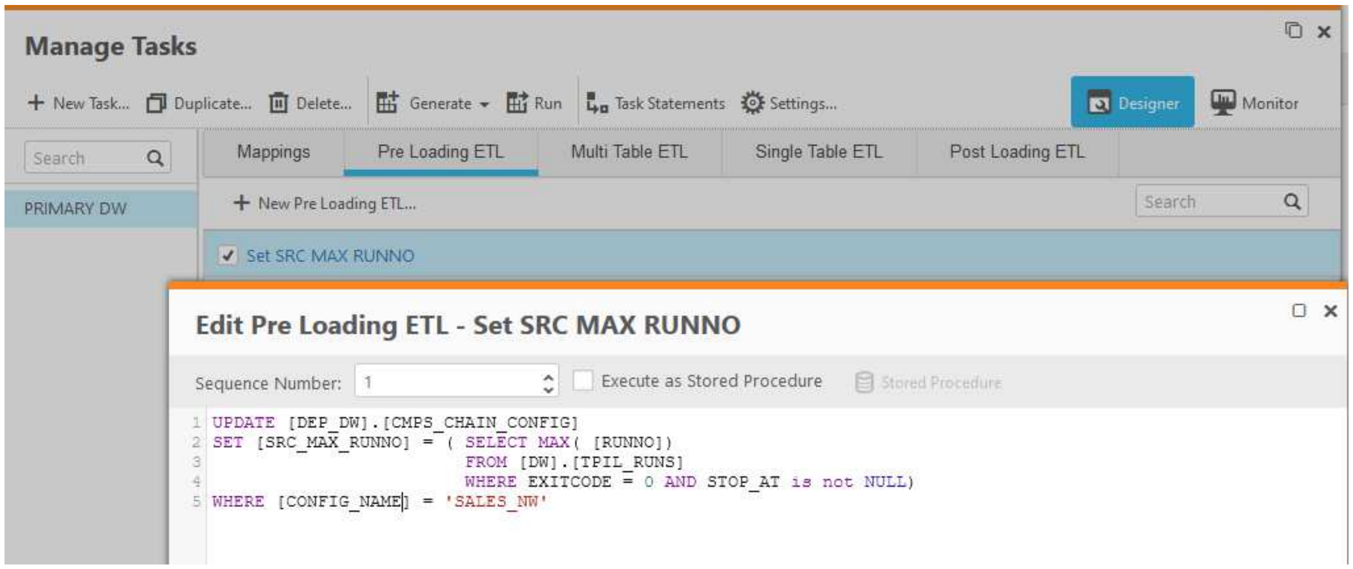
1 Entities

Attributes

Key	Name	Attribute Dom...	Data Type	History	Satellit...
🔑	CategoryID	CategoryID	Integer	Type 1	Hub
	Picture	Picture	Varchar(40)	Type 2	1
	CategoryNa...	CategoryName	Varchar(50)	Type 2	1
	Description	Description	Varchar(8000)	Type 2	1

4 Attributes

4. Create a Pre-Load ETL step to update a control table record and set the SRC_MAX_RUNNO


Manage Tasks

PRIMARY DW

Mappings | Pre Loading ETL | Multi Table ETL | Single Table ETL | Post Loading ETL

✓ Set SRC MAX RUNNO

Edit Pre Loading ETL - Set SRC MAX RUNNO

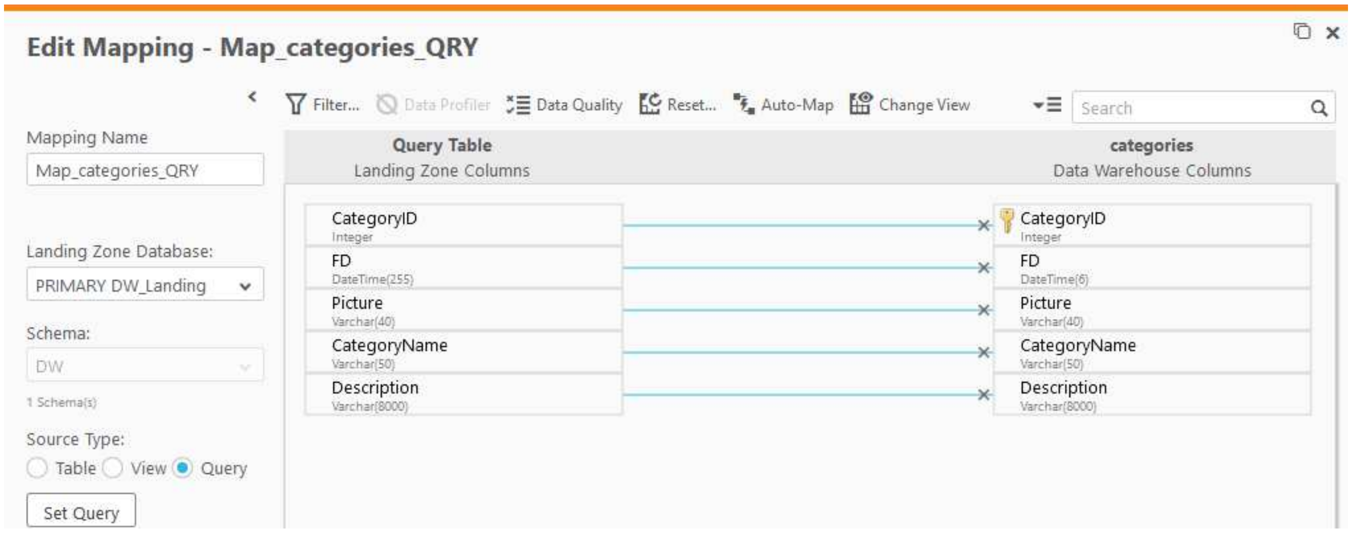
Sequence Number: 1

Execute as Stored Procedure

```
UPDATE [DEP_DW].[CMPS_CHAIN_CONFIG]
SET [SRC_MAX_RUNNO] = ( SELECT MAX( [RUNNO])
                        FROM [DW].[TPIL_RUNS]
                        WHERE EXITCODE = 0 AND STOP_AT is not NULL)
WHERE [CONFIG_NAME] = 'SALES_NW'
```

Note: EXITCODE = 0 ensures we only select the highest successful RUNNO and STOP_AT not NULL ensures we don’t pick up a run in-progress.

5. Design a query-based mapping to load the dependent project entities from the Primary Data Warehouse.



Mapping Name: Map_categories_QRY

Landing Zone Database: PRIMARY DW_Landing

Schema: DW

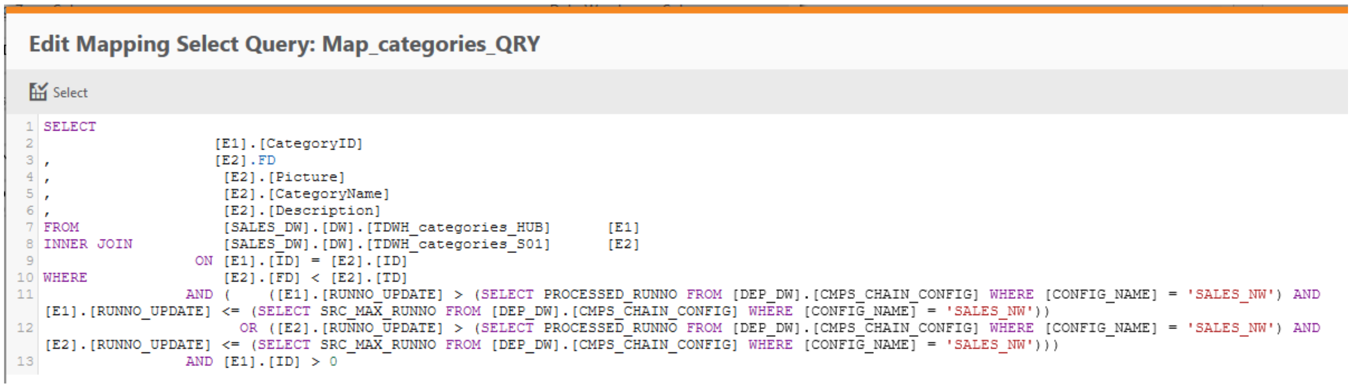
1 Schema(s)

Source Type: Table View [✓ Query]

Set Query

Query Table Landing Zone Columns	categories Data Warehouse Columns
CategoryID Integer	CategoryID Integer
FD DateTime(255)	FD DateTime(6)
Picture Varchar(40)	Picture Varchar(40)
CategoryName Varchar(50)	CategoryName Varchar(50)
Description Varchar(8000)	Description Varchar(8000)

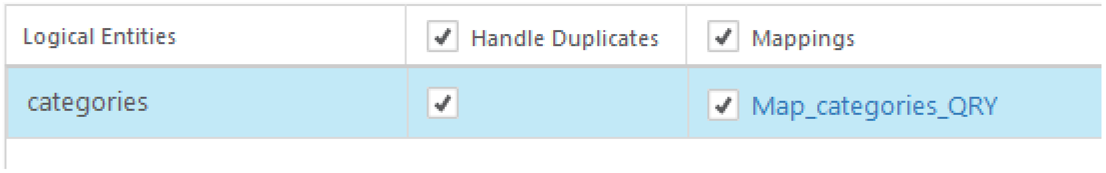
Each of these mappings will have a filter to only processed data that has changed Primary DW since the last RUNNO that the dependent DW processed.



```
SELECT
                       [E1].[CategoryID]
,                      [E2].FD
,                       [E2].[Picture]
,                       [E2].[CategoryName]
,                       [E2].[Description]
FROM                    [SALES_DW].[DW].[TDWH_categories_HUB]       [E1]
INNER JOIN              [SALES_DW].[DW].[TDWH_categories_S01]       [E2]
                     ON [E1].[ID] = [E2].[ID]
WHERE                   [E2].[FD] < [E2].[TD]
                    AND (     ([E1].[RUNNO_UPDATE] > (SELECT PROCESSED_RUNNO FROM [DEP_DW].[CMPS_CHAIN_CONFIG] WHERE [CONFIG_NAME] = 'SALES_NW') AND
[E1].[RUNNO_UPDATE] <= (SELECT SRC_MAX_RUNNO FROM [DEP_DW].[CMPS_CHAIN_CONFIG] WHERE [CONFIG_NAME] = 'SALES_NW'))
                          OR ([E2].[RUNNO_UPDATE] > (SELECT PROCESSED_RUNNO FROM [DEP_DW].[CMPS_CHAIN_CONFIG] WHERE [CONFIG_NAME] = 'SALES_NW') AND
[E2].[RUNNO_UPDATE] <= (SELECT SRC_MAX_RUNNO FROM [DEP_DW].[CMPS_CHAIN_CONFIG] WHERE [CONFIG_NAME] = 'SALES_NW')))
                    AND [E1].[ID] > 0
```

Note: This example shows the common scenario of retrieving type 1 data from the Hub table and type 2 data from Satellite 1. The condition to select the RUNNO requires an OR because there could be change in either Hub or Satellite(s) that need to be processed. This OR is not needed if there's only Hub table data to process.

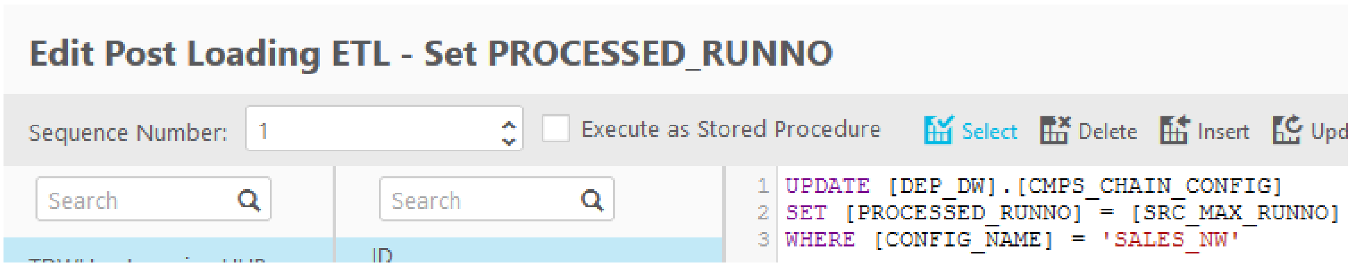
6. Enable the Mapping



Logical Entities	✓ Handle Duplicates	✓ Mappings
categories	✓	✓ Map_categories_QRY

Note: Handle duplicates is only needed if were processing a type 2 data from the Primary Data Warehouse.

7. Create a Post Loading ETL step to update the control table record and set the PROCESSED_RUNNO to the SRC_MAX_RUNNO being used in the current run.


Edit Post Loading ETL - Set PROCESSED_RUNNO
Sequence Number: 1
Execute as Stored Procedure
Select
Delete
Insert
Upd
Search
Search
ID

```
UPDATE [DEP_DW].[CMPS_CHAIN_CONFIG]
SET [PROCESSED_RUNNO] = [SRC_MAX_RUNNO]
WHERE [CONFIG_NAME] = 'SALES_NW'
```

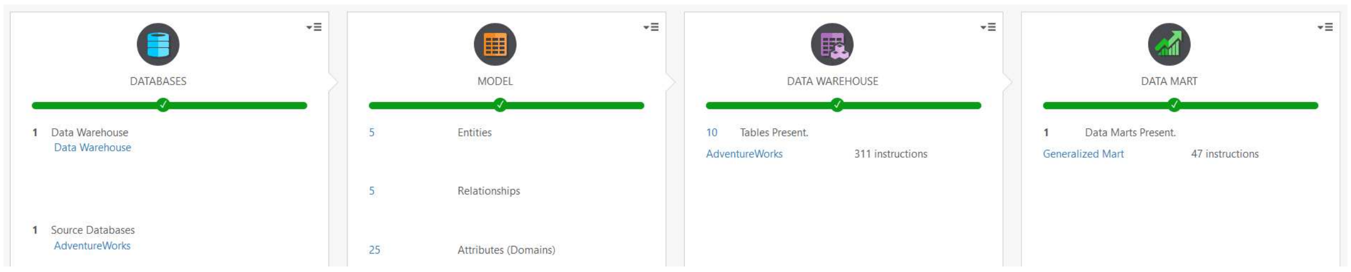
If data needs to be reprocessed for a specific ETL Task / source, then the control table can always be updated manually so the PROCESSED_RUNNO is set back to a prior value.

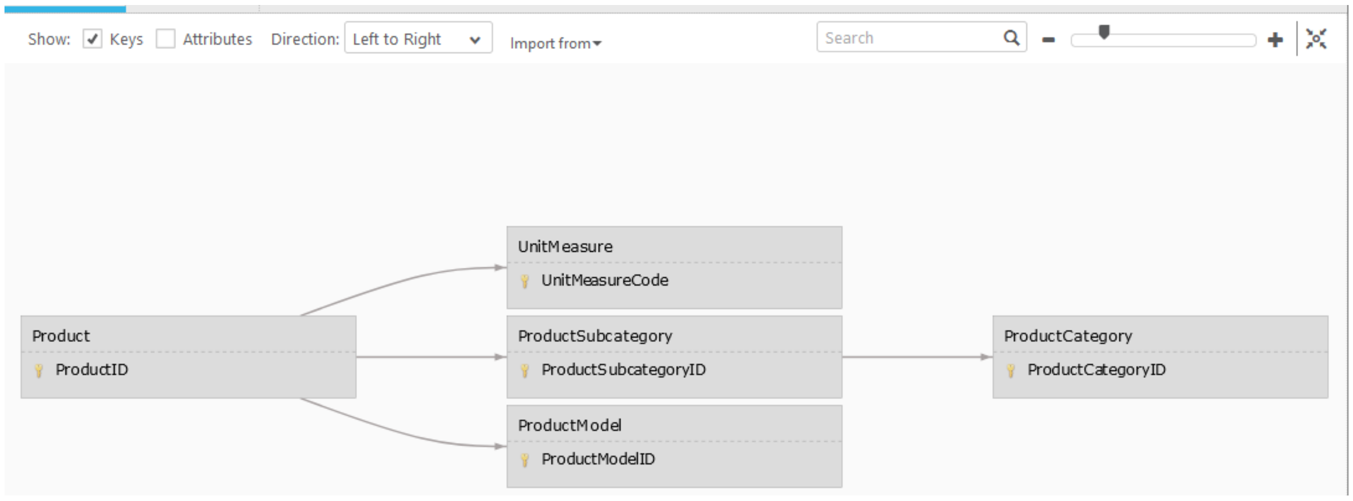
8. The entity is now ready to be used in the Secondary data warehouse. The same process would be repeated for subsequent entities that need to be source from the Primary relational lake warehouse into the dependent warehouse

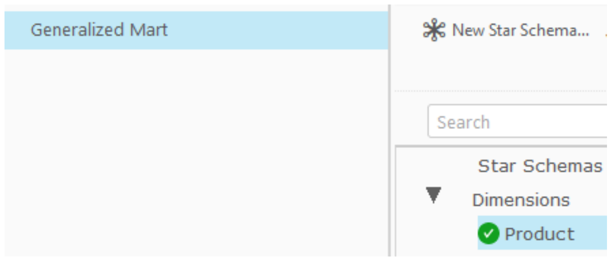
Subject Area/Generalized Data Warehouse Implementation

The premise for this sample implementation is a detailed Subject Area Warehouse for products from which a higher-level product dimension is built and then used as standalone entity in the Generalized Data Warehouse.

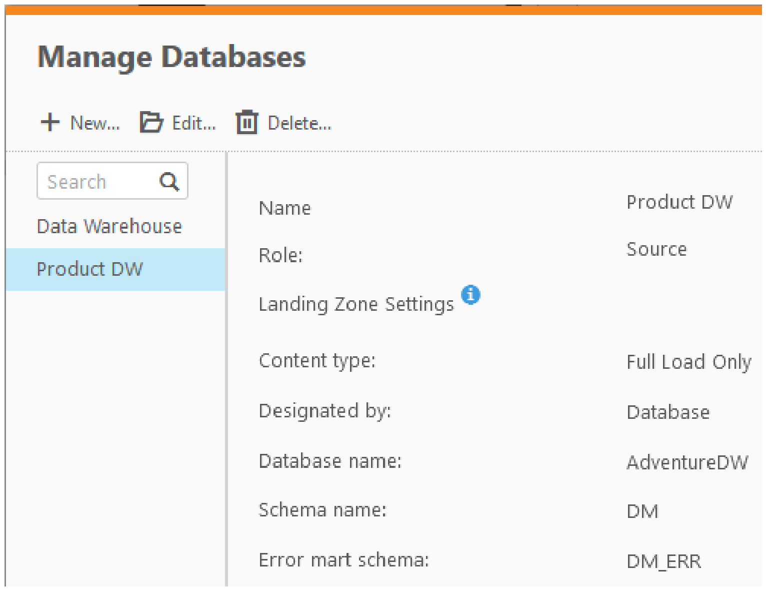
1. Configure Compose project for Subject Area Warehouse with the normalized model for the subject area, a Data Warehouse layer and a Data Mart with the dimension(s) that will be used by the Generalized Data Warehouse





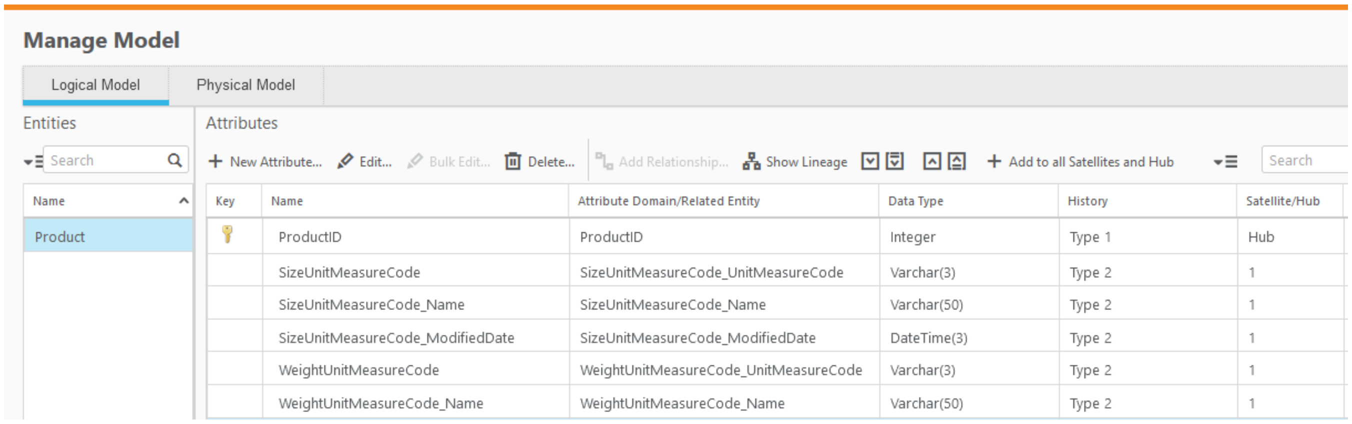


2. Setup the Generalized Data Warehouse project database(s) with a Full Load Only source connection to the Subject Area project Data Mart database.



Name	Product DW
Role:	Source
Landing Zone Settings	
Content type:	Full Load Only
Designated by:	Database
Database name:	AdventureDW
Schema name:	DM
Error mart schema:	DM_ERR

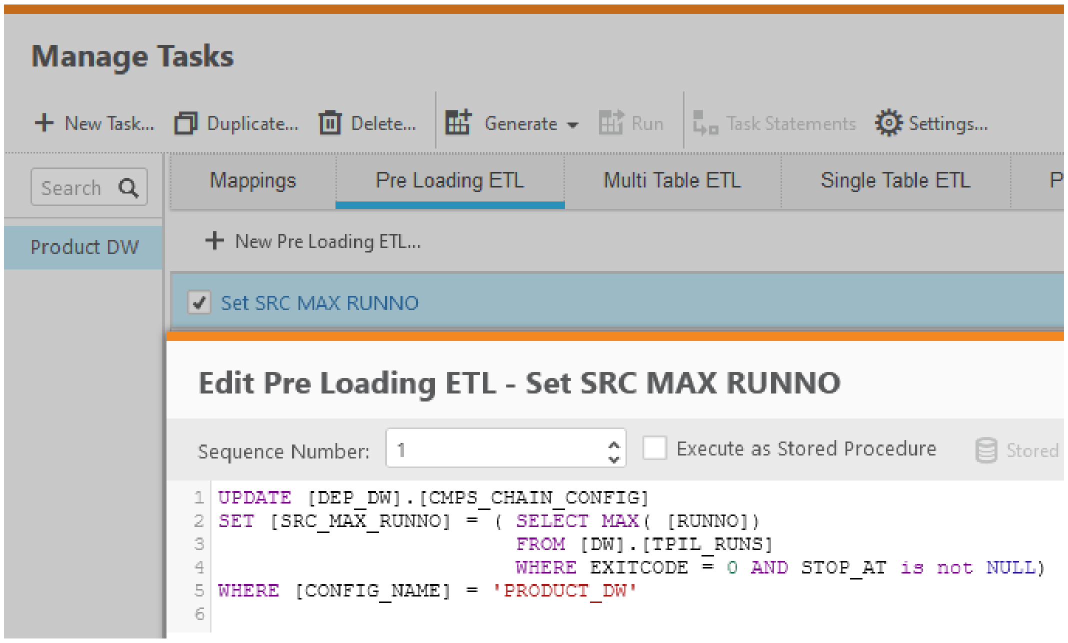
3. Discover the entity(ies) from the Subject Area Database. Rename the entity as needed and replace the business key to be the natural key of the entity instead of the OID/VID key from the dimension. For clarity it is recommended to delete these along with the Compose process columns



Name
Product

Key	Name	Attribute Domain/Related Entity	Data Type	History	Satellite/Hub
🔑	ProductID	ProductID	Integer	Type 1	Hub
	SizeUnitMeasureCode	SizeUnitMeasureCode_UnitMeasureCode	Varchar(3)	Type 2	1
	SizeUnitMeasureCode_Name	SizeUnitMeasureCode_Name	Varchar(50)	Type 2	1
	SizeUnitMeasureCode_ModifiedDate	SizeUnitMeasureCode_ModifiedDate	DateTime(3)	Type 2	1
	WeightUnitMeasureCode	WeightUnitMeasureCode_UnitMeasureCode	Varchar(3)	Type 2	1
	WeightUnitMeasureCode_Name	WeightUnitMeasureCode_Name	Varchar(50)	Type 2	1

4. Create a Pre-Load ETL step to update a control table record and set the SRC_MAX_RUNNO



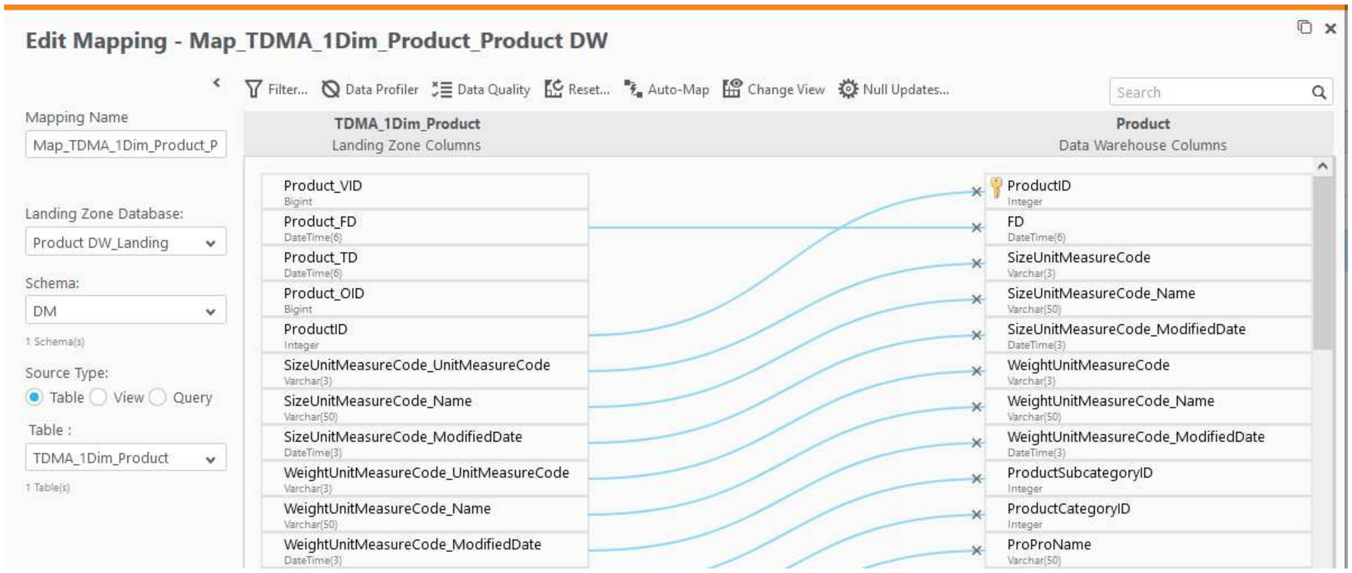
Sequence Number: 1

```
UPDATE [DEP_DW].[CMPS_CHAIN_CONFIG]
SET [SRC_MAX_RUNNO] = ( SELECT MAX( [RUNNO])
                        FROM [DW].[TPIL_RUNS]
                        WHERE EXITCODE = 0 AND STOP_AT is not NULL)
WHERE [CONFIG_NAME] = 'PRODUCT_DW'

```

Note: EXITCODE = 0 ensures we only select the highest successful RUNNO and STOP_AT not NULL ensures we don't pick up a run in-progress.

5. Design a table-based mapping to load the dependent project entities from the Subject Area Data Warehouse.


Edit Mapping - Map_TDMA_1Dim_Product_Product DW

Mapping Name: Map_TDMA_1Dim_Product_P

Landing Zone Database: Product DW_Landing

Schema: DM

1 Schema(s)

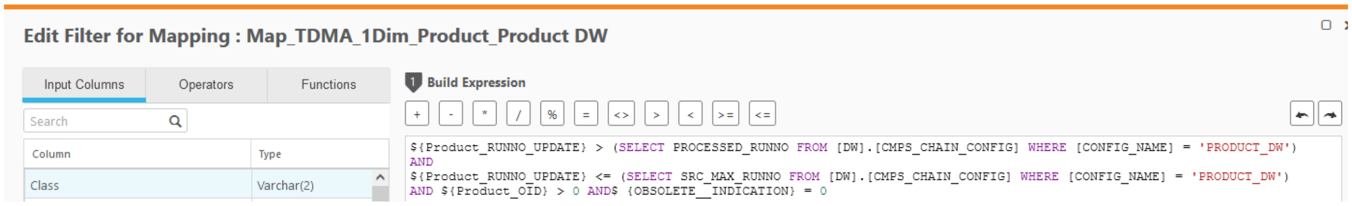
Source Type: [✓ Table] View Query

Table : TDMA_1Dim_Product

1 Table(s)

TDMA_1Dim_Product Landing Zone Columns	Product Data Warehouse Columns
Product_VID (Bigint)	ProductID (Integer)
Product_FD (DateTime(6))	FD (DateTime(6))
Product_TD (DateTime(6))	SizeUnitMeasureCode (Varchar(3))
Product_OID (Bigint)	SizeUnitMeasureCode_Name (Varchar(50))
ProductID (Integer)	SizeUnitMeasureCode_ModifiedDate (DateTime(3))
SizeUnitMeasureCode_UnitMeasureCode (Varchar(3))	WeightUnitMeasureCode (Varchar(3))
SizeUnitMeasureCode_Name (Varchar(50))	WeightUnitMeasureCode_Name (Varchar(50))
SizeUnitMeasureCode_ModifiedDate (DateTime(3))	WeightUnitMeasureCode_ModifiedDate (DateTime(3))
WeightUnitMeasureCode_UnitMeasureCode (Varchar(3))	ProductSubcategoryID (Integer)
WeightUnitMeasureCode_Name (Varchar(50))	ProductCategoryID (Integer)
WeightUnitMeasureCode_ModifiedDate (DateTime(3))	ProProName (Varchar(50))

Each of these mappings will have a filter to only processed data that has changed in the Subject Area DM since the last RUNNO that the dependent DM processed.

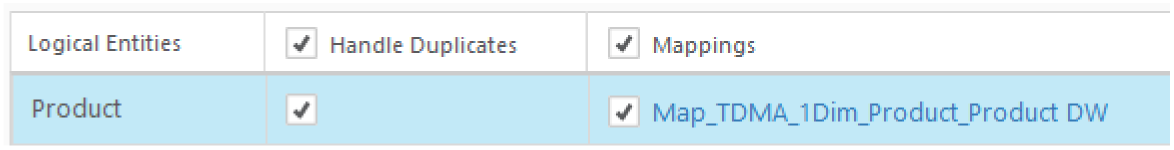

Edit Filter for Mapping : Map_TDMA_1Dim_Product_Product DW

Column	Type
Class	Varchar(2)

```
${Product_RUNNO_UPDATE} > (SELECT PROCESSED_RUNNO FROM [DW].[CMPS_CHAIN_CONFIG] WHERE [CONFIG_NAME] = 'PRODUCT_DW')
AND
${Product_RUNNO_UPDATE} <= (SELECT SRC_MAX_RUNNO FROM [DW].[CMPS_CHAIN_CONFIG] WHERE [CONFIG_NAME] = 'PRODUCT_DW')
AND ${Product_OID} > 0 AND$ {OBSOLETE__INDICATION} = 0
```

Note: In this example, the From Date (FD) is mapped to the From Date of the dimension record. This is only needed if the original dimension is type 2.

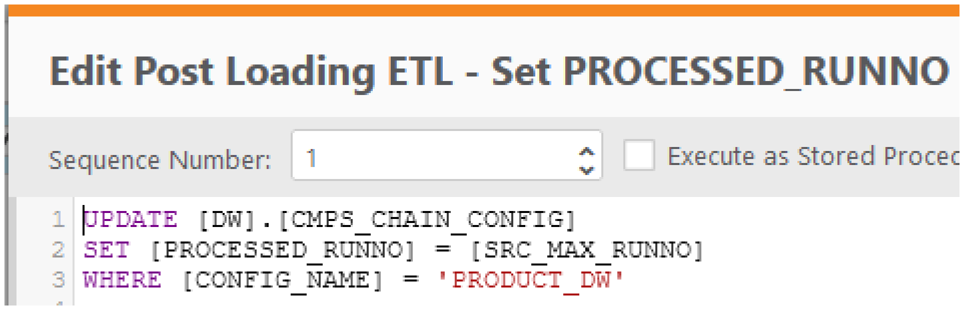
6. Enable the Mapping



Logical Entities	✓ Handle Duplicates	✓ Mappings
Product	✓	✓ Map_TDMA_1Dim_Product_Product DW

Note: Handle duplicates is only needed if were processing a type 2 data from the Subject Area Data Warehouse.

7. Create a Post Loading ETL step to update the control table record and set the PROCESSED_RUNNO to the SRC_MAX_RUNNO being used in the current run.



Sequence Number: 1

```sql
UPDATE [DW].[CMPS_CHAIN_CONFIG]
SET [PROCESSED_RUNNO] = [SRC_MAX_RUNNO]
WHERE [CONFIG_NAME] = 'PRODUCT_DW'
```

If data needs to be reprocessed for a specific ETL Task / source, then the control table can always be updated manually so the PROCESSED_RUNNO is set back to a prior value.

8. The entity is now ready to be used in the generalized data warehouse. The same process would be repeated for other subject areas and entities needed in the generalized warehouse

Conclusion

Qlik Data Integration standard pipeline using Replicate and Compose provides end to end automation for most data warehouse implementations. Specialized use cases such as the Relational Data Lake or Subject Area/Generalized Data Warehouse can be implemented with additional layering of multiple Compose projects. Implementation requires knowledge and use of more advanced Compose features along with custom steps to ensure change processing into the Secondary Compose managed warehouse. The flexibility and customization potential of Qlik Compose provides a path to implement these complex use cases.

Appendix – DDL/DML SQL and Mapping code used

```
/****** Object:  Table [dbo].[DW.CMPS_CHAIN_CONFIG]    Control Table DDL ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [DW].[CMPS_CHAIN_CONFIG](
       [CONFIG_NAME] [varchar](255) NOT NULL,
       [PROCESSED_RUNNO] [int] NOT NULL,
       [SRC_MAX_RUNNO] [int] NOT NULL
) ON [PRIMARY]
GO

ALTER TABLE [DW].[CMPS_CHAIN_CONFIG] ADD  CONSTRAINT [DF_CMPS_CHAIN_CONFIG_PROCESSED_RUNNO]
DEFAULT ((0)) FOR [PROCESSED_RUNNO]
GO

ALTER TABLE [DW].[CMPS_CHAIN_CONFIG] ADD  CONSTRAINT [DF_CMPS_CHAIN_CONFIG_SRC_MAX_RUNNO]
DEFAULT ((0)) FOR [SRC_MAX_RUNNO]
GO

/****** Primary/Secondary Data Warehouse SQL code ******/
/****** Pre Load ETL ******/
UPDATE [DEP_DW].[CMPS_CHAIN_CONFIG]
SET [SRC_MAX_RUNNO] = ( SELECT MAX( [RUNNO])
                                            FROM [DW].[TPIL_RUNS]
                                            WHERE EXITCODE = 0 AND STOP_AT is not NULL)
WHERE [CONFIG_NAME] = 'SALES_NW'

/****** Query to retrieve Hub and Sattellite data from Primary DW ******/
SELECT
                   [E1].[CategoryID]
,                              [E2].FD
,                   [E2].[Picture]
,                   [E2].[CategoryName]
,                   [E2].[Description]
FROM                [SALES_DW].[DW].[TDWH_categories_HUB]       [E1]
INNER JOIN          [SALES_DW].[DW].[TDWH_categories_S01]       [E2]
                 ON [E1].[ID] = [E2].[ID]
WHERE               [E2].[FD] < [E2].[TD]
                AND (    ([E1].[RUNNO_UPDATE] > (SELECT PROCESSED_RUNNO FROM
[DEP_DW].[CMPS_CHAIN_CONFIG] WHERE [CONFIG_NAME] = 'SALES_NW') AND [E1].[RUNNO_UPDATE] <=
(SELECT SRC_MAX_RUNNO FROM [DEP_DW].[CMPS_CHAIN_CONFIG] WHERE [CONFIG_NAME] = 'SALES_NW'))
                      OR ([E2].[RUNNO_UPDATE] > (SELECT PROCESSED_RUNNO FROM
[DEP_DW].[CMPS_CHAIN_CONFIG] WHERE [CONFIG_NAME] = 'SALES_NW') AND [E2].[RUNNO_UPDATE] <=
(SELECT SRC_MAX_RUNNO FROM [DEP_DW].[CMPS_CHAIN_CONFIG] WHERE [CONFIG_NAME] = 'SALES_NW')))
                AND [E1].[ID] > 0

/****** Post Load ETL ******/
UPDATE [DEP_DW].[CMPS_CHAIN_CONFIG]
SET [PROCESSED_RUNNO] = [SRC_MAX_RUNNO]
WHERE [SOURCE_NAME] = 'SALES_NW'
```

```
/****** Subject Area/Generalized Warehouse SQL code ******/
/****** Pre Load ETL ******/
UPDATE [DW].[CMPS_CHAIN_CONFIG]
SET [SRC_MAX_RUNNO] = ( SELECT MAX( [RUNNO])
                                         FROM [AdventureDW].[DW].[TPIL_RUNS]
                                         WHERE EXITCODE = 0 AND STOP_AT is not NULL)
WHERE [CONFIG_NAME] = 'PRODUCT_DW'

/****** Table Mapping Filter to retrieve Subject Area Data Mart data  ******/
${Product_RUNNO_UPDATE} > (SELECT PROCESSED_RUNNO FROM [DW].[CMPS_CHAIN_CONFIG] WHERE
[CONFIG_NAME] = 'PRODUCT_DW')
AND
${Product_RUNNO_UPDATE} <= (SELECT SRC_MAX_RUNNO FROM [DW].[CMPS_CHAIN_CONFIG] WHERE
[CONFIG_NAME] = 'PRODUCT_DW')
AND ${Product_OID} > 0 AND$ {OBSOLETE__INDICATION} = 0

/****** Post Load ETL ******/
UPDATE [DW].[CMPS_CHAIN_CONFIG]
SET [PROCESSED_RUNNO] = [SRC_MAX_RUNNO]

WHERE [CONFIG_NAME] = 'PRODUCT_DW'
```