

Incremental Load Patterns for Non-Replicate Data Sources

Best Practices for Compose for Data Warehouses

TABLE OF CONTENTS

Qlik Compose ETL Tasks	3
Incremental Load Patterns	4
Implementing the Process Flag Pattern	5
Implementing the Incrementing Filter Pattern	9
Applying Incremental Design Patterns to View or Query Based Mappings	14
Conclusion	16

SUMMARY

- Qlik Data Integration provides an integrated data warehouse automation solution architected to incrementally load data based on changes captured in the source system.
- In certain scenarios the automated, code-less incremental load process cannot be leveraged
- Qlik Compose provides functionality to support multiple incremental load patterns for non-Replicate delivered data

INTRODUCTION

Enterprise sources of data are evolving and becoming more varied. Traditional data sources such as relational database management systems (RDBMS) and files still provide the bulk of operational data, but enterprise analytics now require loading and transformation of a variety of data types from semi-structured (e.g. JSON files) to data sets shared via new cloud data warehouse sharing features.

Qlik's Data Integration platform provides functionality to support near real-time data delivery from traditional sources, files, and some SaaS environments via Qlik Replicate. Qlik Compose for Data Warehouses is integrated with Qlik Replicate to automate the incremental load process of the data warehouse based on change data capture (CDC) from the source system. However, there are certain scenarios where the automated incremental load processes cannot be leveraged. In those instances there are extract transform and load (ETL) patterns that can be implemented in Qlik Compose to support incremental data loads. This paper will describe these scenarios and show how to implement custom

incremental load patterns in Qlik Compose and how those patterns can be applied to query or view based mappings.

Qlik Compose ETL Tasks

Qlik Compose provides two types of ETL tasks that support different use cases / processing requirements. 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	<ul style="list-style-type: none"> ● 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 	<ul style="list-style-type: none"> ● CDC based processing for Replicate data sources ● Intra-day / near real-time data loads ● Batch data loads driven from CDC delivered data
Limitations	<ul style="list-style-type: none"> ● Out of the box workload processes the complete table / view / query result set 	<ul style="list-style-type: none"> ● Not applicable for mappings using a query or view as the source
Data Source	<ul style="list-style-type: none"> ● Table (full load for Replicate delivered data) ● Queries ● Views 	<ul style="list-style-type: none"> ● Change Tracking tables (Compose appends __CT to the mappings defined source)

Qlik Compose provides native support for incremental loading of data ingested to the data warehouse by Qlik Replicate using change data capture. This integration leverages Replicates “store changes” setting and change tracking (_ct tables) features. Whenever possible, you should strive to use Replicate STORE CHANGES settings and Compose CDC ETL tasks to process data incrementally to reduce custom code requirements and leverage automation. However, there are scenarios that prevent the use of the Compose CDC ETL tasks.

Scenarios where CDC ETL Tasks are not applicable

Qlik Compose can automate the data warehouse processing of other sources of data that are accessible from within the data warehouse. These could be tables or views within the data warehouse

environment, data shared using features like Snowflakes data sharing or even There are a few scenarios where the Compose CDC ETL task is not applicable these include –

- Data sourced by a non-Replicate technology. For example:
 - Data shared using cloud data warehouse features like Snowflake’s data sharing
 - External tables pointing to data structures in an object store, such as S3, Azure Data Lake Storage Gen2 or Google Cloud Storage.
 - Data ingested to the warehouse by a technology other than Qlik Replicate
- Replicate sources that
 - Only allow Full Loads
 - Do not support full supplemental logging or full before-image and after-image are not in the log.

An example of a data source that has restrictions due to the data managed in the transaction log is SAP HANA. Qlik Replicate provides two methods to capture changes from SAP Hana – trigger based (which has no restrictions for Compose processing) and log based capture. For log-based capture, SAP HANA does not provide primary key information in the transaction log to be captured by Qlik Replicate. As such, Qlik Replicate cannot provide the necessary change tracking table data to support out of the box CDC ETL tasks in Qlik Compose

Qlik Compose Full load tasks could be leveraged for batch based workloads to detect new and changed records between the source data and the data warehouse however, this requires a full data comparison which can be computationally expensive or not support an incremental processing SLA. When these scenarios occur, ETL patterns can be implemented in Qlik Compose to support incremental data loading. Specific data warehouse platforms may have features that can be supported with query-based mappings (for example Snowflake table streams). These environment specific features are not considered in this white paper. Instead, this provides general patterns for incremental data loading.

Incremental Load Patterns

While there are many methods to create incremental loads, there are two common incremental load patterns that can be implemented in Qlik Compose. These are ETL patterns that have stood the test of time and are common to those familiar with creating ETL processes to incrementally read source data and deliver to the target. The patterns are:

- Process Flag Pattern

- Incrementing Filter Pattern

	Process Flag	Incrementing Filter Pattern
Pattern Overview	Leverage an additional column in the source data to signify the set of data that is ready to process	Leverage an existing column in the source data like a timestamp to filter out new records
When To Use It	<ul style="list-style-type: none"> • Applicable to any data source that can be updated by Compose • Source data is incrementally appended or updated • Do not want to delete or archive landing data • Do not have a naturally occurring incrementing column in the source data • Merging tables from multiple sources in Replicate 	<ul style="list-style-type: none"> • Applicable to any data source with a naturally occurring incrementing column (e.g. timestamp) • Source data is incrementally appended or updated • External tables pointing to data in data lake that you cannot update/delete • Table in DW • Do not want to delete or archive landing data
Benefits	<ul style="list-style-type: none"> • Simple to implement • Supports any data source in the data warehouse that can be updated • Source data can be incrementally updated while processing 	<ul style="list-style-type: none"> • Simple to implement • Consume source data as is in an incremental fashion • Query based process – no additional “updates” to source data
Considerations	<ul style="list-style-type: none"> • Need to add attribute to the landing table • Custom code required to manage the processing flag • Source data requires “updating” to flag processing characteristic • Additional overhead in managing <i>process_flag</i> attribute across multiple tables 	<ul style="list-style-type: none"> • When data sourced by Replicate leverage <i>MERGE</i> features to ensure consistency of incrementing column • Column with incrementing value must be available in the source

These two patterns can be used with any data ingested to the data warehouse to provide incremental processing (including Qlik Replicate ingested data).

Implementing the Process Flag Pattern

Leveraging a “process flag” is a common development pattern when ETL needs to process new or changed data. The pattern is applicable to any incremental design process, but specifically applies when the source data does not provide an incrementing column (like a *last_update_dtm* or sequence number) and it is continuously being altered by the data ingest process. The *Process Flag pattern* requires adding a column to the landing table (source table for a Qlik Compose mapping). This column

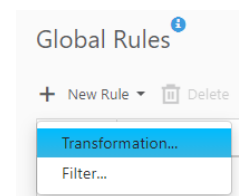
is used to filter new or changed records and track the status of records through the ETL process, from new (N) to processing (P) to completed (Y).

In this example, we will walk through configuring Qlik Replicate and Qlik Compose to use a pattern flag, however this same pattern can be implemented with data sources ingested by other means. This pattern would be applicable for Replicate sources that do not provide enough data in the log for a complete CT record, for example log-based HANA capture.

Landing Area Configuration

Data sources ingested to the data warehouse platform outside of Qlik Replicate should add the *process_flag* column and new data ingested should set the column value to 'N' to signify a new record / change that must be processed. Data ingested by Qlik Replicate can leverage a *Global Transformation Rule* to configure this new column and its value. (A global rule allows the logic to be applied to all or a subset of tables in the Qlik Replicate task). To configure a rule in Qlik Replicate:

- In the Qlik Replicate Task Designer click **Global Rules** *
- Click **New Rule** then **Transformation**
- Select **Add Column**
- Apply any table / scope filters as required
- Configure the *Transformation Action* for the column name, data type and expression (see below image)



Transformation Action

Choose what to change

Column name:	<input type="text" value="process_flag"/>
Add to Primary Key:	<input type="checkbox"/>
Expression	<input type="text" value="'N'"/> <input type="button" value="fx"/>
Set target data type to:	<input type="text" value="STRING"/>
Length:	<input type="text" value="1"/>

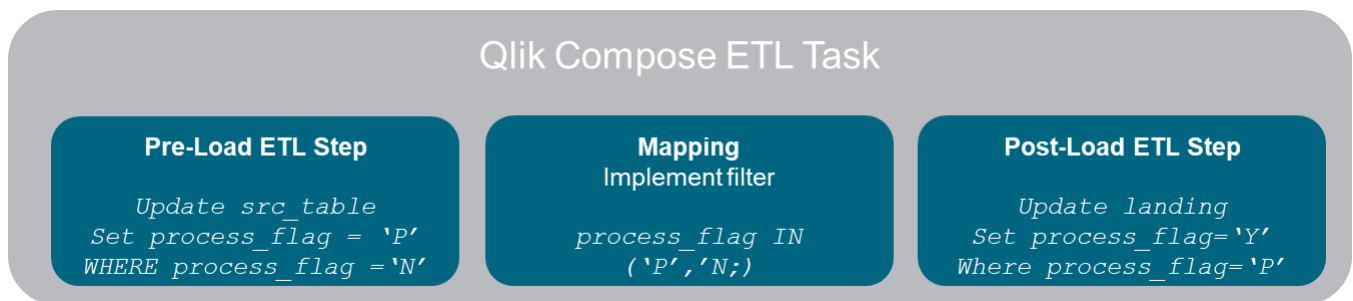
- Complete the *Global Rule* wizard and run the Qlik Replicate task

Qlik Replicate will automatically create the target tables with the *process_flag* column. Any activity performed against the target table by Qlik Replicate will set the *process_flag* value to 'N'. Qlik Compose will use this value to know which records it should process.

Qlik Compose Configuration

Qlik Compose should be configured to only process data that has been marked 'N' by the ingest process. Since new records / changes could be fed to the landing area during the ETL process, you must ensure you capture a static set of records to process, and subsequently mark these records as processed when the ETL is completed. This can be done by configuring 3 components in Qlik Compose:

- A Pre-Loading ETL step to mark records as ready to process
- A Mapping filter to only process marked records
- A Post-Loading ETL step to mark the processed records as completed

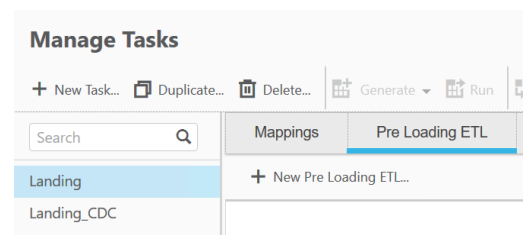


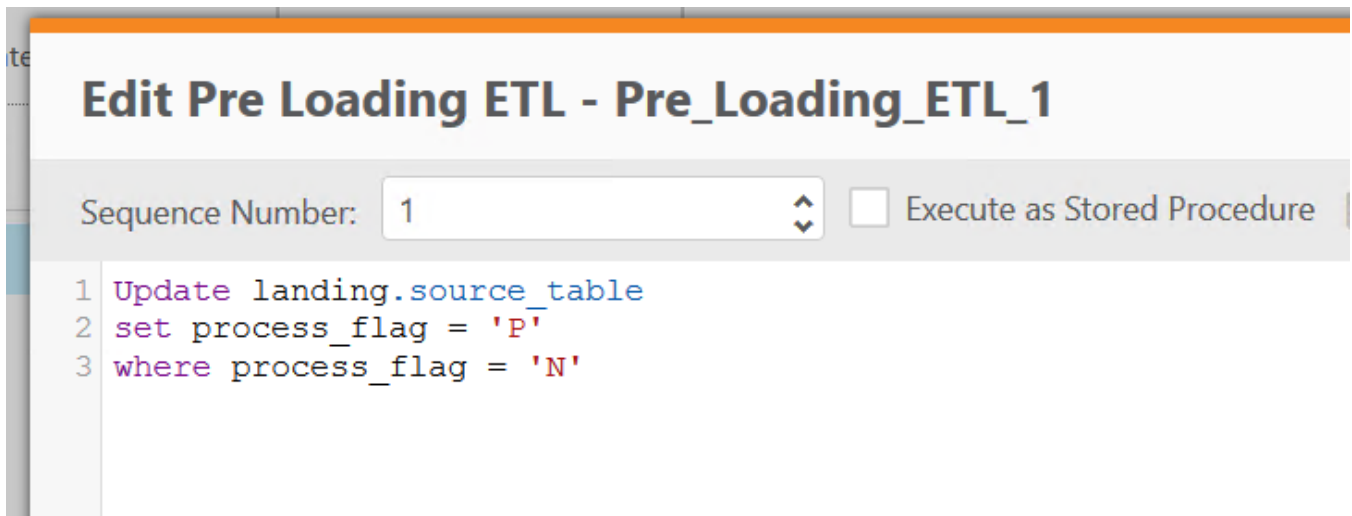
These components are applied to a single Qlik Compose ETL task to automate the processing requirements.

Pre Loading ETL Step

A *Pre Loading ETL* step can be configured to update the source tables and set the *process_flag* column to 'P' to signify the record is ready to be processed. To configure this in Qlik Compose:

- Click **Manage** under the *Data Warehouse* section
- Click **Pre Loading ETL** and **New Pre Loading ETL...**
- Enter a name for the ETL step and enter the update statement required

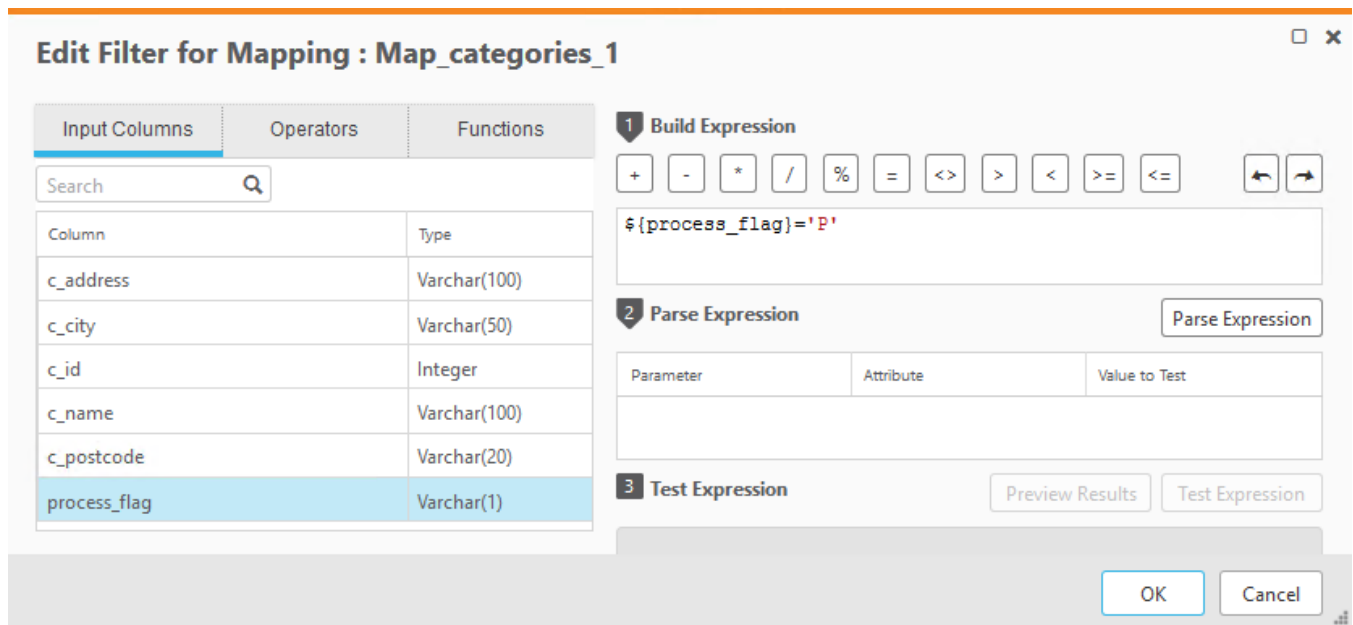




- Note you could also have Qlik Compose execute a stored procedure you have written in your data warehouse environment to update the *process_flag*

Mapping Filter

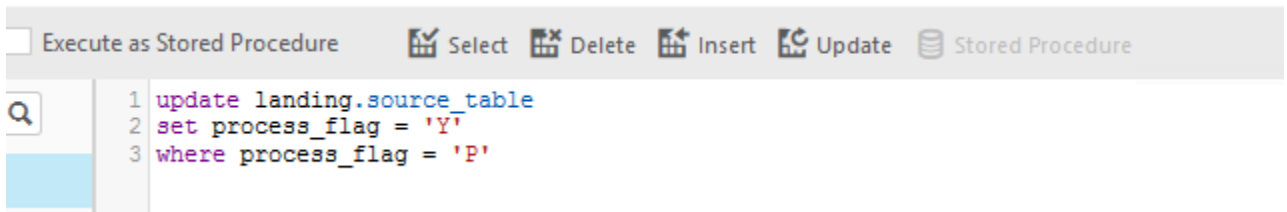
A filter should be created in the mapping to ensure Compose only processes new or changed records. A typical implementation would filter on *process_flag = 'P'* to only select the records marked by the Pre-Loading ETL step. However, if you have records that are constantly updated in the source environment, implementing a filter *process_flag IN ('N','P')* ensures that Compose picks up records marked either P or N and reduces any latency of those records that are updated continuously. To implement the filter in a mapping simply open the mapping in Compose and click on the **Filter** button and enter the required filter.



Post Loading ETL

The Post Loading ETL step should be defined to update the landing table and signify that the records marked for processing have been processed. Marking the records in the first step ensures that only those records that were static since the start of the process will be marked as completed (any new / changed records would have the `process_flag` set to 'N' by the ingestion process). To create the Post Load step –

- Click **Manage** under the *Data Warehouse* section
- Click **Post Loading ETL** and **New Pre Loading ETL...**
- Enter a name for the ETL step and enter the update statement required



The screenshot shows a toolbar with options: Execute as Stored Procedure, Select, Delete, Insert, Update, and Stored Procedure. Below the toolbar, a search icon is visible on the left, and a SQL editor contains the following code:

```
1 update landing.source_table
2 set process_flag = 'Y'
3 where process_flag = 'P'
```

- Note you could also have Qlik Compose execute a stored procedure you have written in your data warehouse environment to update the *process_flag*.

Ensure all the required Pre Loading ETL, Mappings, Post Loading steps are tagged to the appropriate full load ETL Task in Qlik Compose and generate the code. Compose will execute pre-load, mappings and automated ETL to load the data vault structures and complete the process by marking records as processed.

Considerations

Leveraging a “`process_flag`” pattern is common practice for ETL developers. One item to consider is that this incremental design pattern requires additional updates to the source table to ensure Compose processes specific records and marks those records as being processed. While these statements are very lightweight, it does incur some additional processing.

Implementing the Incrementing Filter Pattern

Many data sources provide a data ingestion timestamp, or the staging / landing tables provide an incrementing column which can be used to know when new or changed records have been applied to the landing area for Compose to process. The incrementing filter pattern leverages an incrementing (or decrementing) column to know which records are new or have been changed since the last Compose

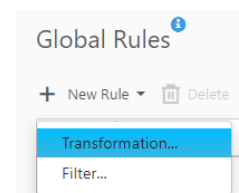
execution. This pattern requires managing the last processed value in order to ensure Compose processes only new or changed data for each execution. This can be maintained in the Compose data model (for simplicity) or it could be managed in a separate control table that would be updated and managed with Pre and Post load features for the ETL set.

In this example, we will walk through configuring Qlik Replicate and Qlik Compose to use an incrementing column to filter new and change records, however this same pattern can be implemented with data sources ingested by other means. This pattern would be applicable for Replicate sources that do not provide enough data in the log for a complete CT record, for example log-based HANA capture.

Landing Area Configuration

Data sources ingested to the data warehouse platform outside of Qlik Replicate should have a column with incrementing values that are updated anytime a new or changed record is applied. Data ingested by Qlik Replicate can leverage a *Global Transformation Rule* to configure this new column and its value. (A global rule allows the logic to be applied to all or a subset of tables in the Qlik Replicate task). Qlik Replicate provides internal variables that can be used to manage the incrementing columns. One option would be to use the `$AR_H_COMMIT_TIMESTAMP` which is the source commit timestamp. Another is to use the `$AR_H_CHANGE_SEQ` which is an always increasing value managed by Qlik Replicate based on the Replicate server UTC time. While either of these options will work, in this example we will utilize the `$AR_H_CHANGE_SEQ` value. To configure a rule in Qlik Replicate:

- In the Qlik Replicate Task Designer click **Global Rules** *
- Click **New Rule** then **Transformation**
- Select **Add Column**
- Apply any table / scope filters as required
- Configure the *Transformation Action* for the column name, data type and expression (see below image)



Transformation Action
Choose what to change

Column name:	<input type="text" value="ChangeSeq"/>
Add to Primary Key:	<input type="checkbox"/>
Expression	<input type="text" value="\$AR_H_CHANGE_SEQ"/> <input type="button" value="fx"/>
Set target data type to:	<input type="text" value="STRING"/>
Length:	<input type="text" value="50"/>

- Complete the *Global Rule* wizard and run the Qlik Replicate task

In this example, Qlik Replicate will automatically create the target tables with the *ChangeSeq* column. Any activity performed against the target table by Qlik Replicate will update the *ChangeSeq* column to a new value. Qlik Compose will use this value to know which records it should process.

Qlik Compose Configuration

Qlik Compose should be configured to only process data that has a higher *ChangeSeq* value than the last processed value. While there are different methods to manage the processed *ChangeSeq* value, a recommended approach is to add the *ChangeSeq* attribute to every entity in the Compose model as a Type 1 attribute. The ETL mappings can then filter source data where the *ChangeSeq* is newer than the highest value managed in the Compose HUB tables. This leverages the most automation within Qlik Compose and allows the value to be managed on an entity by entity basis within the Compose model. To implement this pattern in Qlik Compose –

- For each entity in the Compose Model click **New Attribute**
- Define an attribute as per the below image

- Compose will place the *ChangeSeq* attribute in the logical entities HUB table
- Edit the Compose mappings that utilize this incremental load pattern and apply a filter as per the below image (replacing the NVL function with the null replace function for your data warehouse, and setting the appropriate schema / table name)

- Compose will filter source records based on the *ChangeSeq* and update the *ChangeSeq* value in the HUB as part of its ETL processing



Extending the Configuration to Process Multiple Sources

The above scenario provides the template for supporting incremental changes when a single mapping (and thus single source) is loading a Compose entity. In a Compose data warehouse implementation it is common to have multiple sources loading an entity. How these sources are related will impact how the *ChangeSeq* value is managed in Compose. Two common scenarios are *UNIONing* or *JOINing* source data. If the data is provided by the same Qlik Replicate task (e.g. 2 tables from the same

source), then the above pattern will suffice as the *ChangeSeq* is guaranteed to be managed within a single Replicate task. However if data is delivered by different ingestion technologies or different Replicate tasks then the *ChangeSeq* needs to be managed with a *Source* granularity.

There are 2 methods to handle this. Assuming a *UNION* use case, then it is common to include a *SourceSystem* column in the data model so you have some source lineage for each record. In this scenario each mapping (UNION's are implemented by leveraging multiple mappings in the same ETL set in Compose) simply filters based on its *SourceSystem* value.

The below model shows an entity with a composite key for *CustomerID* and *SourceSystem*.



Key	Name	Data Type	History	Satellite/Hub
	CategoryID	Varchar(5)	Type 1	Hub
	SourceSystem	Varchar(50)	Type 1	Hub
	ChangeSeq	Varchar(50)	Type 1	Hub

The below image depicts the filter to be applied in the Compose mapping. Note the additional filter for *SourceSystem*.

1 Build Expression

`${ChangeSeq} >= (SELECT NVL(MAX(ChangeSeq), '0') FROM EDW_SCHEMA.TDWH_CATEGORIES_HUB WHERE SourceSystem='SOURCEA')`

In a scenario where multiple source systems (and possibly partial mappings) are used to populate a single logical entity in Compose with JOIN conditions (or partial mappings), it is recommended to manage each sources *ChangeSeq* (or equivalent column) independently (see below image).

Key	Name	Data Type	History	Satellite/Hub
	CategoryID	Varchar(5)	Type 1	Hub
	SourceSystem	Varchar(50)	Type 1	Hub
	Source1_ChangeSeq	Varchar(50)	Type 1	Hub
	Source2_ChangeSeq	Varchar(50)	Type 1	Hub

Assuming the mapping uses a query or view based source to define the joins between the two disparate sources, the filter should then take into account both *ChangeSeq* columns to detect new or changed records.

```
1 Build Expression
+ - * / % = <> > < >= <=
(
  ${Source1_ChangeSeq} >= (SELECT NVL(MAX(Source1_ChangeSeq),'0') FROM EDW_SCHEMA.TDWH_CATEGORIES_HUB
OR
  ${Source2_ChangeSeq} >= (SELECT NVL(MAX(Source2_ChangeSeq),'0') FROM EDW_SCHEMA.TDWH_CATEGORIES_HUB
)
```

Any combination of *ChangeSeq* (or equivalent) columns can be stored / maintained in the data warehouse logical entity and used in the filter.

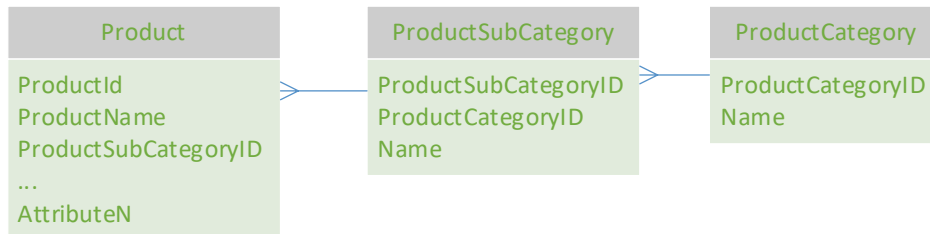
Considerations

This simple pattern does not require additional updates to the source data set (reducing additional processing requirements) and relies on Compose to manage the processed values. If this pattern is implemented in a cloud data warehouse environment (for example Snowflake or BigQuery) with Qlik Replicate as the ingestion method, it is recommended to leverage *MERGE* configuration in Qlik Replicate to ensure the micro-batch ingestion is performed in a single statement. (This ensures the *ChangeSeq* values are not incrementally adjusted out of order). Managing the last processed values in the data warehouse enables automation without additional hand-written code and ensures any combination of ETL mappings into ETL tasks will get the appropriate “current” value. Note however that this could also be implemented using a single control table which would be updated via a Compose Pre-Load and Post-Load task. This methodology is documented in a separate whitepaper (available on community.qlik.com) **Layering the Qlik Data Integration Architecture with Compose Projects**.

Applying Incremental Design Patterns to View or Query Based Mappings

The patterns described above are applicable for non-Replicate data sources, but they can also be leveraged to incrementally process data for mappings using a query or a view for the source. As described previously, CDC ETL Tasks do not support mappings with a query or view as the source for the mapping. Since a view and query can have any number of complex operations it is not possible for

Compose to automatically know the subset of records changed since the last ETL task execution. Leveraging the patterns described allow the Full Load Task to only process new or changed data and support incremental loading. The below model represents the source of data that is loaded to the data warehouse landing area by a non-Replicate ingestion process. The requirement for the Compose model is to join *Product*, *ProductSubCategory* and *ProductCategory* to represent a denormalized entity “ProductMaster”. The incremental data load must consider changes to any source table.



Process Flag Pattern

The Landing area design and pre-ETL process steps remain the same when implementing a query or view based mapping. Each source table will have a “Process_Flag” column appended and set to “N” to any new or changed rows. The Pre-ETL process will update that column to “P” when ready to process the data. The query / view requires implementing the FILTER on Process_Flag with OR conditions. See query below joining 3 product tables to produce a denormalized product data set.

```

1
2 SELECT P.* ,
3       S.Name as SubcategoryName,
4       C.ProductCategoryID,
5       C.Name as CategoryName
6
7 FROM Production.Product P
8 LEFT JOIN Production.ProductSubcategory S
9     on P.ProductSubcategoryID = S.ProductSubcategoryID
10 LEFT JOIN Production.ProductCategory C
11     on C.ProductCategoryID = S.ProductCategoryID
12 WHERE P.Process_Flag IN ('P','N')
13        OR S.Process_Flag IN ('P','N')
14        OR C.Process_Flag IN ('P','N')
    
```

The above query ensures that changes to any table are considered as part of the load to staging process.

Incremental Filter Pattern

Implementing an incremental filter pattern for query or view based mappings follows the same principle as the process flag. Filters with OR conditions can be used to determine which records have changed. The additional step is to determine the max *ChangeSeq* for each record to apply to the Compose model. The below query leverage OR conditions in the WHERE clause to ensure changes to any table are considered. The CASE statement ensures that the current “max” ChangeSeq value across the tables is written into the Product HUB.

```
1
2 SELECT P.* ,
3     S.Name as SubcategoryName,
4     C.ProductCategoryID,
5     C.Name as CategoryName,
6
7     CASE    WHEN P.ChangeSeq > S.ChangeSeq AND P.ChangeSeq > C.ChangeSeq THEN P.ChangeSeq
8             WHEN S.ChangeSeq > P.ChangeSeq AND S.ChangeSeq > C.ChangeSeq THEN S.ChangeSeq
9             ELSE C.ChangeSeq
10    END as ChangeSeq
11
12 FROM Production.Product P
13 LEFT JOIN Production.ProductSubcategory S
14     on P.ProductSubcategoryID = S.ProductSubcategoryID
15 LEFT JOIN Production.ProductCategory C
16     on C.ProductCategoryID = S.ProductCategoryID
17 WHERE P.ChangeSeq > (SELECT MAX(ChangeSeq) FROM DW_SCHEMA.TDWH_Product_HUB )
18     OR S.ChangeSeq > (SELECT MAX(ChangeSeq) FROM DW_SCHEMA.TDWH_Product_HUB )
19     OR C.ChangeSeq > (SELECT MAX(ChangeSeq) FROM DW_SCHEMA.TDWH_Product_HUB )
```

These incremental design patterns can be used for any source of data (even if using Replicate with APPLY CHANGES) when the mapping source needs to be a view or a query.

Conclusion

Qlik’s data integration platform provides solutions for automating the data warehouse lifecycle. While the combination of Qlik Replicate and Qlik Compose provides powerful features to support change data capture based incremental data loading of the data warehouse and subsequent data marts, there are conditions where the process cannot be 100% automated. In those scenarios, Qlik Compose provides the necessary features to implement incremental design patterns regardless of the data ingestion methodology or any restrictions that may arise from specific Qlik Replicate sources.

About Qlik

Qlik's vision is a data-literate world, one where everyone can use data to improve decision-making and solve their most challenging problems. Only Qlik offers end-to-end, real-time data integration and analytics solutions that help organizations access and transform all their data into value. Qlik helps companies lead with data to see more deeply into customer behavior, reinvent business processes, discover new revenue streams, and balance risk and reward. Qlik does business in more than 100 countries and serves over 50,000 customers around the world.

qlik.com

