

Ingestion patterns: CDC

***Owner:***

Rafael Castro

***Contributors***:

Rafael Castro

Mattia Fantoni

Yannick Rödl

Alex Schaepper

***Reviews*:**

Colin Sutherland

***Versions:***

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Author(s)** | **Notes** |
| 0.1 | 2020-09-08 | Rafael Castro | First formal proposal |
| 0.2 | 2020-09-20 | Rafael Castro | Post-meeting corrections by Alex et al. |

[1 Introduction 2](#_Toc51593182)

[1.1 Project Landscape 2](#_Toc51593183)

[1.2 Role of the Design Decisions 2](#_Toc51593184)

[1.3 Decision Template 2](#_Toc51593185)

[2 CDC patterns 2](#_Toc51593186)

[2.1 DATE\_MODIFIED or Sequential\_ID 3](#_Toc51593187)

[2.2 Diff 3](#_Toc51593188)

[2.3 Log-Based Change Data Capture 4](#_Toc51593189)

[3 Streaming DIFF CDC pattern 4](#_Toc51593190)

[3.1 Architecture 5](#_Toc51593191)

[3.2 Flow 5](#_Toc51593192)

[4 Design Decisions 6](#_Toc51593193)

[4.1 Discriminating between new or modified records 6](#_Toc51593194)

[4.2 Select fields for CDC 7](#_Toc51593195)

[4.3 When does the CDC happen 7](#_Toc51593196)

[4.4 How the CDC is invoked 7](#_Toc51593197)

[4.5 CDC table materialisation 8](#_Toc51593198)

[4.6 Deleted records detection 8](#_Toc51593199)

[5 Open questions 9](#_Toc51593200)

# Introduction

## Project Landscape

This document describes the design decisions for the Change Data Capture (CDC) ingestion pattern, in particular for non-incremental CDC as applicable to Midcorp and other bulk data imports.

## 1.2 Role of the Design Decisions

The design decision in this document summarize best practices as a suggested pattern for GDP rollout projects with OEs. The OEs are free to accept, change, or reject the design suggestions for their rollout projects.

However, deviating from the standard outlined in this document can have a severe impact on the efforts and risks for rollout projects.

## 1.3 Decision Template

Decisions are documented in the following format:

|  |  |
| --- | --- |
| **Key** | **Value** |
| ID |  |
| Date |  |
| Item Description |  |
| Design Pattern |  |

# CDC patterns

As its name suggests, [Change Data Capture](https://www.hvr-software.com/product/change-data-capture/)(CDC) techniques are used to identify changes. CDC can be the basis to synchronize another system with the same incremental changes, or to store an audit trail of changes. The audit trail may subsequently be used for other uses e.g. to update a data warehouse or to run analyses across the changes e.g. to identify patterns of changes. Below are described three common methods to perform CDC: Date\_Modified or Sequential\_ID, DIFF and Log-Based Change Data Capture, with a deep dive on a streaming variation of the DIFF method.

## 2.1 DATE\_MODIFIED or Sequential\_ID

Many transactional applications keep track of metadata in every row including who created and/or most-recently modified the row, as well as when the row was created and last modified. The approach to CDC in such an environment is to keep track of when changes are extracted, and in a subsequent run filter on the DATE\_MODIFIED column to only retrieve rows that were modified since the most recent time data was extracted (or in the case of a sequential ID, retrieve rows with ID greater than the last extracted ID). This approach has a few challenges that may or may not be a concern, depending on the application:

* Data deletes are a challenge because there is no DATE\_MODIFIED for a deleted row (unless deletes are logical and update a flag in the row indicates the row was deleted). The extreme case of delete is truncate table which is uncommon in transactional applications but does occur sometimes.
* DATE\_MODIFIED must be available on all tables and must be reliably set. Database triggers may be a good way to set the values but these may introduce overhead on the transactional application.
* Extracting the changes uses a lot of resources. Of course DATE\_MODIFIED may be indexed to lower the impact of the select statement at the cost of storing (and continuously updating) the additional index.

Using DATE\_MODIFIED for CDC works well for traditional data warehouse applications that are populated using Extract, Transform and Load (ETL) jobs, when the source tables don’t process deletes or deletes should not be replicated to the downstream systems.

## 2.2 Diff

The diff method for change data capture compares the current state of the data with previous state of the data to identify what changed. Challenges with this approach include:

* To perform the diff requires a lot of resources to compute the differences between the data, and resource consumption grows at least linearly with the growth in data volume.
* CDC cannot be performed in real-time because the diff realistically takes too many resources to perform all the time.

Compared to the DATE\_MODIFIED CDC method the Diff method does not have the challenge with deleted rows. The Diff method works well for low data volumes or when small latencies (for the CDC processing) are acceptable.

## 2.3 Log-Based Change Data Capture

Transactional databases store all changes in a transaction log in order to recover the committed state of the database should the database crash for whatever reason. [Log-based CDC](https://www.hvr-software.com/product/change-data-capture/) takes advantage of this aspect of the transactional database to read the changes from the log. The challenges with log-based CDC are:

* Interpreting the changes in the transaction log is difficult because there are no documented standards on how the changes are stored (i.e. transaction logs from different database vendors are completely different), and there are many scenarios that must all be considered and tested (e.g. consider clustered databases, rollbacks and savepoints, many different ways to perform inserts, updates and deletes, etc.).
* Database vendors may not provide an interface to the transaction logs – documented or not – and even if there is one it may be relatively slow and/or resource intensive.
* Most databases have been optimized to only use internal identifiers to recover database row changes which is insufficient to perform CDC and record the changes on a different system. Supplemental logging of primary key columns is required to retrieve the context of the updates. The introduction of supplemental logging will increase the volume of data written to the transaction logs but generally only by a small percentage, and generally, there is very little if any measurable performance impact on the transactional application.

Log-based CDC should be the ultimate goal, but difficult to implement since it has dependencies to source system changes. For current projects, diff on bulk exports may be more pragmatic. The biggest benefit of log-based change data capture is the asynchronous nature of CDC: changes are captured independent of the source application performing the changes and most of the drawbacks are addressed by applications like [Debezium](https://debezium.io/).

# 3 Streaming DIFF CDC pattern

Compared to the classic DIFF method with streaming there are less resources used, as each incoming record is compared independently with the current data table, without having to load the incoming bulk data again. When using KTables this is particularly efficient, as the keys format is the same and the access is similar to that in a hash table.

Data deletes are still a challenge, as deletes do not emit an event, for now we will assume deletes are logical (therefore get captured).

## 3.1 Architecture

Upon a snapshot ingestion (into a “pre-raw” topic), in the presence of non-incremental CDC, we stream the newly ingested records and do a LEFT JOIN with a newly streamed table from the existing current raw topic, where this table has the same keys and a hashed value of a subset or all the fields of the given entity. The join filters the new and modified records, marking these records with a flag as new or modified record.

These records are then archived into the compacted raw topic (or as parquet files as a more durable alternative) and also get applied the transformations as specified by the mappings in the UI.

Note that when importing for the first time we create an empty CDC table with all records being new.

![A screenshot of a cell phone

Description automatically generated]()

## 3.2 Flow

UI

![A screenshot of a cell phone

Description automatically generated]()

Clicking on the CDC (could be re-labelled as “Bulk export”) checkbox and optionally selecting which fields to compare for CDC (advanced feature, initially will just do a hash of all fields) triggers behind the scenes the CDC process, consisting of:

- the initial ingestion of the new records to a “pre-raw” topic, with an SMT inserting a new flag field, to mark later the records as new or modified.

-  the creation of the CDC table with records (key, hashed\_value) from the existing archive topic

-  the streaming of the new records

-  the left outer join between the two, resulting in a new raw topic which is then picked by the ingestion service for transformation.

# 4 Design Decisions

## 4.1 Discriminating between new or modified records

|  |  |
| --- | --- |
| Key | Value |
| ID | 1 |
| Date | 2020-09-08 |
| Item Description | Discriminating between new or modified records |
| Design Pattern | In order to discriminate between new and modified records an additional text field is inserted upon insertion via an SMT, “is\_new” initially empty, and set to either “I” for insertions, “U” for updates,”D” for deletes after CDC is applied. |

## 4.2 Select fields for CDC

|  |  |
| --- | --- |
| Key | Value |
| ID | 2 |
| Date | 2020-09-08 |
| Item Description | Selecting which fields to use for CDC |
| Design Pattern | There will be a future requirement to be able to select specific fields to track changes, this will require a bit of additional work on the UI and perhaps a change in the design of the CDC KTable (e.g. store all fields, rather than just hashed values, for the increased flexibility of hashing some fields or others). In the first iteration we will hash all fields. |

## 4.3 When does the CDC happen

|  |  |
| --- | --- |
| Key | Value |
| ID | 3 |
| Date | 2020-09-08 |
| Item Description | When does CDC happen |
| Design Pattern | The CDC will produce new and modified records only, the resulting topic with these records is then used for transformation. Therefore, CDC happens before the raw topic and should be transparent to the user (with the CDC table and intermediate topics being purely technical and not exposed to the metadata server). |

## 4.4 How the CDC is invoked

|  |  |
| --- | --- |
| Key | Value |
| ID | 4 |
| Date | 2020-09-08 |
| Item Description | How the CDC is invoked |
| Design Pattern | CDC is done in a Kafka streams app, due to limitations in KSQL. The CDC app should be a separate service and have an API wrapping the CDC app for invocation. |

## 4.5 CDC table materialisation

|  |  |
| --- | --- |
| Key | Value |
| ID | 5 |
| Date | 2020-09-08 |
| Item Description | How the CDC table is materialised |
| Design Pattern | Due to the distributed nature of Kafka to ensure a proper join either a GlobalKTable needs to be used or the same co-partitioning strategy between the underlying stream/table topics needs to be ensured. The GlobalKTable approach requires replication of all data to all the nodes and therefore lots of synchronization, and at the same time we use the same keys, partitioners and number of partitions between the two topics, ensuring co-partitioning with the stream and CDC KTable.  In the first iteration we are using a non-materialised KTable, instead creating it on demand every time there is an import as imports are not so frequent. As the import frequency increases might make more sense to materialise the CDC KTable. |

## 4.6 Deleted records detection

|  |  |
| --- | --- |
| Key | Value |
| ID | 6 |
| Date | 2020-09-08 |
| Item Description | Dealing with deleted records |
| Design Pattern | Tracking deletions can be done by doing a full outer join and observing the null keys between old and new dumps. The user should be able to enable/disable deletes upon stream definition but for now we will assume deletes are logical (therefore get captured as modified records). |

# 5 Open questions

Questions (bizkeys):

* CDC on PKs or Bizkeys?
* Where to generate bizkey
* Reading keys info from Informatica/Metadata server