## Incremental Loading

**Requirements:**

Για να είναι παραγωγικό ένα incremental load πρέπει να μπορούμε χωρίς να επιβαρύνουμε το refresh να ελέγξουμε εάν μία εγγραφή δεν υπάρχει στα υπάρχον data μας στο qlik. Αυτό γίνεται μέσω ενός workbook variable που κρατάει την max ημερομηνία κάθε refresh και αυξάνεται για τις νέες εγγραφές (η κάποιο άλλο arbitrary variable που μπορεί να γίνει evaluated με απλό T/F). Εναλλακτικά θα έπρεπε να ελέγξουμε για κάθε εγγραφή εάν υπήρχε προηγούμενος, το οποίο θα επιβάρυνε περισσότερο το refresh.

Για να υλοποιηθεί το ζητούμενο χρειαζόμαστε δύο πράγματα: μία ημερομηνία που αυξάνεται για μπορούμε να διαχωρίσουμε τα νέα στοιχεία, τα δεδομένα που χρησιμοποιούμε πριν από αυτήν σε κάθε ανανέωση να παραμένουν στατικά.

Walkthrough recording: "G:\Shared drives\GR Data & Analytics\Projects\Hedno - Data Analytics Support\4. Demand Management\0. Recordings of Trainings\Alkis handover - S2 Incremental Loading.gdrive"

**I have written detailed instructions below post recording which are more easily understood imo**

**Instructions:**

* **Step 1:** Depending on load (rows/columns) decide on appropriate step size: For example in case of Καταμέτρηση, measurement\_details\_ had ~60m rows ~35 columns indexing after 10m rows worsened exponentially. For time’s sake, whereas the refresh would eventually complete within 24 hours if split in half ~30m, it was more effective to split the step size even further and save to qvd every ~6-10m rows. This only saves time for the development stage and can complete the qvd in 1 working day (for this case) as opposed to 2-3 days with larger partitions.
* **Step 2:** Reload script for the first time for your decided partition (date range) in the where clause.

manual date formats i.e. **date ‘2023-09-22’**

Then create your initial qvd with the first partition as shown below.

//

new\_entries:

load \* (better to list columns for clarity

take care to capitalize all column names if loading from sql query);

sql

select \*

from your\_table

where (trunc(INCREMENTAL\_DATE\_COLUMN, 'DD') >= date 'first\_partition\_start') and (trunc(INCREMENTAL\_DATE\_COLUMN, 'DD') < date 'first\_partition\_end') ;

STORE new\_entries into your\_qvd\_path\_and\_name.qvd(qvd);

//

* **Step 3:** From the second reload onwards we need to load our next partition and append it to the previously created QVD, which on every iteration will append the new partition to the previously appended QVD.

Repeat this step until the last partition where we will store workbook variables required for automation.

//

new\_entries:

load \* ;

sql

select \*

from your\_table

where (trunc(INCREMENTAL\_DATE\_COLUMN, 'DD') >= date 'first\_partition\_end') and (trunc(INCREMENTAL\_DATE\_COLUMN, 'DD') < date 'third\_partition\_start') ; // this will be the second partition

Concatenate

LOAD \*

FROM your\_qvd\_path\_and\_name.qvd(qvd);

STORE new\_entries into your\_qvd\_path\_and\_name.qvd(qvd); // replace old qvd

//

* **Step 4:**  For the last partition we will fetch all the remaining rows, only one condition will remain in the where clause where INCREMENTAL\_DATE\_COLUMN>=max\_date\_of\_partition\_before\_last, also shown below.

In this step we have completed the QVD which contains our static historical data.

In addition to completing the QVD we can now assign workbook variables which will contain the max INCREMENTAL\_DATE\_COLUMN value of our latest reload.

//

new\_entries:

load \* ;

sql

select \*

from your\_table

where (trunc(INCREMENTAL\_DATE\_COLUMN, 'DD') >= date 'beforelast\_partition\_end'); // fetch all remaining rows

Concatenate

LOAD \*

FROM your\_qvd\_path\_and\_name.qvd(qvd);

STORE new\_entries into your\_qvd\_path\_and\_name.qvd(qvd); // replace old qvd

max\_date\_table:

load date(max(INCREMENTAL\_DATE\_COLUMN), 'YYYY-MM-DD') as max\_date

FROM your\_qvd\_path\_and\_name.qvd(qvd);

LET Max\_date = Peek('max\_date', 0, 'max\_date\_table');

LET Max\_date\_num = floor(Peek('max\_date', 0, 'max\_date\_table'));

DROP Table max\_date\_table;

//

* **Step 5:** Automate the query using the set workbook variables.

For new entries it is essential to use >= to handle multiple refreshes in a single day.

This means < needs to be used on the QVD to avoid duplicate rows.

All this means is that if a workbook was to be refreshed multiple times it would fetch the new\_entries of the

day every time, and use the QVD until today()-1.

//

new\_entries:

load \*;

sql

select \*

from your\_table

where trunc(INCREMENTAL\_DATE\_COLUMN, 'DD') >= date '$(Max\_date)';

Concatenate

LOAD \*;

FROM your\_qvd\_path\_and\_name.qvd(qvd)

where floor(INCREMENTAL\_DATE\_COLUMN)<$(Max\_date\_num);

STORE new\_entries into your\_qvd\_path\_and\_name.qvd(qvd);

max\_date\_table:

load date(max(INCREMENTAL\_DATE\_COLUMN), 'YYYY-MM-DD') as max\_date

FROMyour\_qvd\_path\_and\_name.qvd(qvd)

where floor(INCREMENTAL\_DATE\_COLUMN)>=$(Max\_date\_num);

// look from the last max date variable onwards to minimize the search

// >= required to handle multiple refreshes in single day

LET Max\_date = Peek('max\_date', 0, 'max\_date\_table');

LET Max\_date\_num = floor(Peek('max\_date', 0, 'max\_date\_table'));

DROP Table max\_date\_table;

//