

# Merging with Time Travel

written by DaltonRuer | June 30, 2021



## Yikes

Sorry science fiction fans this post isn't about time travel or the new fascination with surviving while traveling through a black hole.

It's about something much scarier and harder to deal with. End users who are super demanding.

You know the ones I'm talking about. The ones that no matter how often you refresh the data in your Qlik Sense application, they still complain. Because they want the power to press a magic button and get the latest and greatest data in the database right then. You know, so that they have complete control.

There are times you can't really blame them but you are in a bit of a fix. You would like to use [Dynamic Views because you loved my post](#) but they need the ability to interact with all of the data or ask for Insights. You quickly pivot as you know how to do [Partial Reloads in Qlik Sense](#) and you know how to do **incremental loads** but then reality hits you in the face. You read the comments in the load script and they tell you that the reason the scheduled reloads are hourly is because the tables in this system don't have any **Last\_Modified\_Timestamp** type fields that would allow you to load just a handful of records.

It would be out of this world, if you could just travel through time and collect the records that have changed since the application was last reloaded on it's schedule, and then merge them with all of the existing data. Wouldn't it?

## Time Travel

As it turns out my friends. You can travel through time if you are using Snowflake. Well sort of. You see they have a cool feature set built into the product for something called **Time Travel**. One form of that maintains [Changes](#) that are made to the data. It adds metadata columns behind the scenes to your tables that allow you to modify your typical SQL Selection and say "give me only the changes that have occurred since X time". I'm not kidding, check out my query and the results.

```
78 SELECT "METADATA$ACTION", cast(CURRENT_TIMESTAMP() as varchar(100)) as "Timestamp_c",
79      "Hospital_Account_ID", "Primary_ICD_Diagnosis_Code", "Admit_ICD_Diagnosis_Code",
80      "Primary_ICD_Procedure_Code", "Primary_Payor_ID", "Total_Account_Balance_$",
81      "Total_Account_Adjustment_$", "Total_Account_Charge_$", "Total_Account_Payment_$", "HRRP_Condition"
82 FROM "GENERALHOSPITALDB"."dbo"."Accounts"
83 CHANGES(INFORMATION => DEFAULT) AT(TIMESTAMP => '2021-06-29 18:14:18.779 -0400'::timestamp-tz) END(TIMESTAMP => CURRENT_TIMESTAMP())
84 where not("METADATA$ISUPDATE" = TRUE and "METADATA$ACTION" = 'DELETE')
```

Results Data Preview

✓ Query ID SQL 853ms 3 rows

Filter result...



Copy

Row	METADATA\$ACTION	Timestamp_c	Hospital_Account_ID	Primary_ICD_Diagnosis_Code	Admit_ICD_Diagnosis_Code	Primary_ICD_Procedure_Code	Primary_Payor_ID	Total_Account_Balance_\$	Total_Account_Adjustment_\$
1	INSERT	2021-06-29 18:14:18.779 -0400	11111111	777.77	V22.1	74.1	300010	2112700.0000	-65.2000
2	INSERT	2021-06-29 18:14:18.779 -0400	11111122	660.11	660.13	74.1	300029	17600.0000	0.0000
3	INSERT	2021-06-29 18:14:18.779 -0400	11403777	852.05	959.9	39.79	300063	11723577.7000	0.0000

The link for Time Travel Changes above has all of the details so I will focus on the highlights. I've wrapped a normal query with a little bit of syntax that says "get me only the changes that have occurred since '2021-06-29 18:14:18.779 -0400'" One tricky part is the last line that says "Hey Snowflake do me a favor, I know that you store updates as deletes and inserts, and frankly I don't need those because I just care about the new data." Oh Snowflake will still return any actual deletes that occur, but only the ones that aren't part of an update or upsert if you will. You gotta like that.

PS – [Snowflake's Time Travel includes other fantastic features](#)

[as well as this change stuff I'm showing you here. If you really want to have your mind blown check them out.](#)

## Merge

I hear you out there. "That's cool and all but how does that help me. It's not like Qlik Sense will just let me merge that result set from Snowflake with my current gigantic tables or anything."

You see while you have been having so much fun Asking the Insight Advisor questions, setting up Alerting and adding Collaboration to your applications Qlik released a new method of incremental loading that literally does exactly that. [The new method is aptly named Merge](#). As you can see below I literally take the command from above and just tell Qlik Sense to Merge it with the Accounts table and that the Hospital\_Account\_ID field is the Primary Key value it should use.

```
317 // Merge any changes that have been applied since last data load
318 MERGE (Timestamp_c, latestTimestampVar) on "Hospital_Account_ID" concatenate (Accounts) LOAD *;
319 SQL SELECT "METADATA$ACTION", cast(CURRENT_TIMESTAMP() as varchar(100)) as "Timestamp_c",
320 "Hospital_Account_ID", "Primary_ICD_Diagnosis_Code", "Admit_ICD_Diagnosis_Code",
321 "Primary_ICD_Procedure_Code", "Primary_Payor_ID", "Total_Account_Balance_$",
322 "Total_Account_Adjustment_$", "Total_Account_Charge_$", "Total_Account_Payment_$", "HRRP_Condition"
323 FROM "GENERALHOSPITALDB"."dbo"."Accounts" CHANGES(INFORMATION => DEFAULT)
324 AT(TIMESTAMP => '$(lastTimestampVar)':timestamp_tz) END(TIMESTAMP => CURRENT_TIMESTAMP())
325 // don't need DELETE row event generated for Updates, only Insert treated as Upsert but deletes are needed
326 where not("METADATA$ISUPDATE" = TRUE and "METADATA$ACTION" = 'DELETE');
327
```

Your DBA left you hanging and you had no way to know which records could be loaded incrementally, but Qlik and Snowflake had your back.

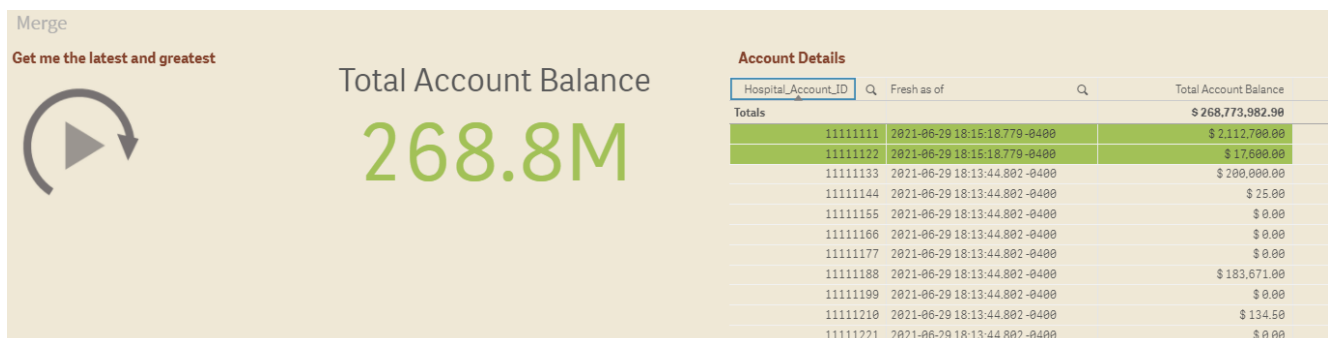
## We've got your back

But it gets better when you think about this from a performance standpoint. No more of that complicated incremental load script code to handle deletes. They are handed to you on a silver platter, quickly, and the Merge handles them for you. Which means you can invoke it as often

as you would like. Like those situations where pesky users want to press a button.

In this screenshot I've taken the Tiny Reload button and added it to the screen so that end users can press it when they want to know the exact up to the second account balances (and any other account details.) If you aren't familiar with how to add extensions this one, or are curious why I chose it please check out my video called [Qlik SaaS Enablement – Extensions](#). In it I demonstrate how to add it and demonstrate why.

While slightly overly dramatic, you will see that in my details table I have a column that shows me when the data was last refreshed, and any rows that were loaded by the end user are displayed in green. (Be sure to let me know if you think that's a bit cheesy or a neat feature your end users might like.)



Merge

Get me the latest and greatest

Total Account Balance

268.8M

Account Details

Hospital_Account_ID	Fresh as of	Total Account Balance
Totals		\$ 268,773,982.90
11111111	2021-06-29 18:15:18.779-0400	\$ 2,112,700.00
11111112	2021-06-29 18:15:18.779-0400	\$ 17,600.00
11111133	2021-06-29 18:13:44.802-0400	\$ 200,000.00
11111144	2021-06-29 18:13:44.802-0400	\$ 25.00
11111155	2021-06-29 18:13:44.802-0400	\$ 0.00
11111166	2021-06-29 18:13:44.802-0400	\$ 0.00
11111177	2021-06-29 18:13:44.802-0400	\$ 0.00
11111188	2021-06-29 18:13:44.802-0400	\$ 183,671.00
11111199	2021-06-29 18:13:44.802-0400	\$ 0.00
11111210	2021-06-29 18:13:44.802-0400	\$ 134.50
11111221	2021-06-29 18:13:44.802-0400	\$ 0.00

## Partial Reload

[Please refer to the documentation for Partial Reload for all of the information you might need, but I wanted to at least show you just how easy it is.](#)

As a developer you need to modify your code slightly and call the system function [IsPartialReload](#) to see if the loading is for a full reload or a partial reload. If it is False then you do your normal table loading like always. At the onset of my "full reload" I run a simple query against Snowflake to ask for it's current timestamp so I know when (in it's time

system) I know when I'm starting my data load.

```
27 // Logic for initial Full reload and Partial reload to import Changes
28 If Not IsPartialReload() Then
29
30     // Read the Timestamp of the current state of base table from Snowflake so we know when we made the last full load
31     TimestampSink:
32     LOAD *;
33     SQL select cast(CURRENT_TIMESTAMP() as varchar(100)) as "lastTimestamp";
34
35     // lastTimestamp will track the last Timestamp of changes processed across Full and Partial Reloads
36     Let lastTimestampVar = peek('lastTimestamp');
37     Let OriginalTimestampVar = peek('lastTimestamp');
38     drop table TimestampSink;
39
40
41     DiabetesData:
42     Load Text("Master_Patient_ID") as "Master_Patient_ID",
43         "race",
44         "gender",
45         "age",
```

After I finish all of the table loading script there is naturally an **Else** where this magic happens. That incredibly complex, I mean super simple, Merge statement that does it's Snowflake Time Travel. Lastly I swap out the variables and reset my LastTimestamp table.

```
315 Else
316
317     // Merge any changes that have been applied since last data load
318     MERGE (Timestamp_c, latestTimestampVar) on "Hospital_Account_ID" concatenate (Accounts) LOAD *;
319     SQL SELECT "METADATA$ACTION", cast(CURRENT_TIMESTAMP() as varchar(100)) as "Timestamp_c",
320         "Hospital_Account_ID", "Primary_ICD_Diagnosis_Code", "Admit_ICD_Diagnosis_Code",
321         "Primary_ICD_Procedure_Code", "Primary_Payor_ID", "Total_Account_Balance_$",
322         "Total_Account_Adjustment_$", "Total_Account_Charge_$", "Total_Account_Payment_$", "HRRP_Condition"
323     FROM "GENERALHOSPITALDB"."dbo"."Accounts" CHANGES(INFORMATION => DEFAULT)
324     AT(TIMESTAMP => '${lastTimestampVar}':timestamp_tz) END(TIMESTAMP => CURRENT_TIMESTAMP())
325     // don't need DELETE row event generated for Updates, only Insert treated as Upsert but deletes are needed
326     where not("METADATA$ISUPDATE" = TRUE and "METADATA$ACTION" = 'DELETE');
327
328
329     // If MERGE LOAD found changes, update the lastTimestampVar variable,
330     // used to find unprocessed changes during the next run
331     if(latestTimestampVar > '') then
332         lastTimestampVar = latestTimestampVar;
333     end if
334 End If; // End of Full or Partial reload
335
336 lastTimestampT:
337 replace load * inline [
338     lastTimestamp
339     ${lastTimestampVar}
340 ];
341
```

## Heads up

When I got started on this journey through time with Snowflake I read in their help that Time Travel is enabled for all Databases. "WooHoo! I can just run my code." To which I was promptly greeted with an error message saying that time travel wasn't enabled. That seems like a contradiction. But then it

hit me. There is storage required to handle those metadata columns I mentioned, and perhaps I need to tell them specifically that I need to do want these changes for particular tables. Sure enough I needed to execute the following statement and then I was cooking with gas.

```
Alter Table "GENERALHOSPITALDB"."dbo"."Accounts" set  
change_tracking = true
```

## Summary

As you are fully aware at this point in your career, there is no such a thing as "one size fits all". There are use cases where this makes sense to allow end users to do partial reloads so they are in control, and there are use cases where this doesn't make sense. You know tables with tons of data that are simply used for historical analysis; Trends over time.

It's comforting to know that for nearly 30 years Qlik has continued to adapt it's solutions to provide developers with ways to handle disparate data from anywhere and ensure you can respond to those demanding use cases. Even cases like this where you are on the leading, with data that does time travel with Snowflake? Yeah they've got you covered. In an easy to implement way that will put a smile on your end users faces.