Detect Customer workload (Statistics on INSERT, DELETE, UPDATE, FETCHED, RETURNED TUPLE)

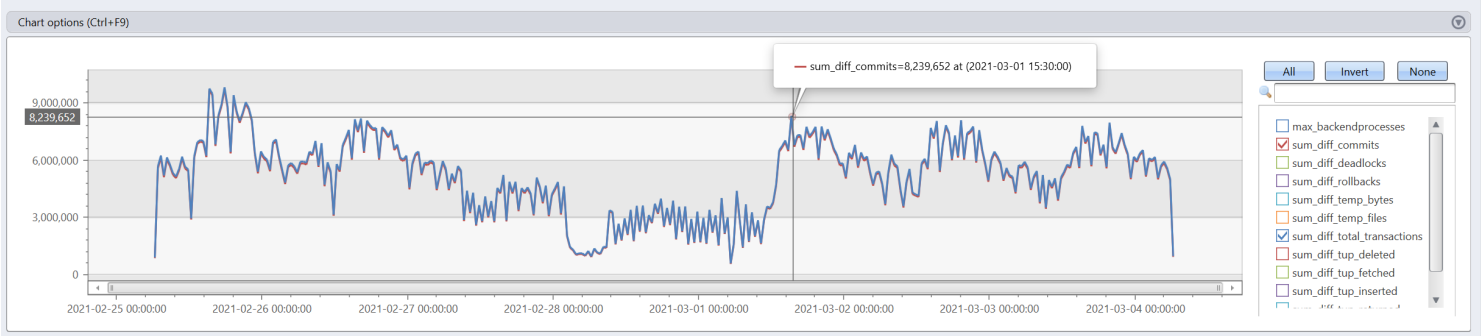
Last updated by | Rich Flanigan | May 28, 2021 at 7:34 AM PDT

The query plots the number of tuples, inserted, deleted etc for every 30minutes (you can change this to 5m granularity if required.)
High write workload will show in insert, delete, update tuple counts or read workload will show in fetched, returned tuple counts.

Comment the yellow line to get the result at 5 mins granularity that this telemetry is emitted.
For looking over few days or data it is better to have that line and look at 30 minute intervals

```
let TimeCheckStart = ago(7d);
let TimeCheckEnd = now();
let ServerName = 'TypeServerNameHere';
MonDmPgSqlTransactionStats
| where LogicalServerName =~ ServerName
| where TIMESTAMP > TimeCheckStart and TIMESTAMP < TimeCheckEnd
| where database_name !in ('template1','template0', 'azure_sys','azure_maintenance')
| summarize
sum(tolong(tup_inserted)),sum(tolong(tup_updated)),sum(tolong(tup_deleted)),sum(tolong(commits)),sum
(tolong(rollbacks)),sum(tolong(total_transactions)),sum(tolong(deadlocks)),sum(tolong(numbackends)),
sum(tolong(tup_fetched)),sum(tolong(tup_returned)),
sum(tolong(temp_bytes)),sum(tolong(temp_files)) by bin(TIMESTAMP,1m)
| order by TIMESTAMP asc
| serialize
| extend tup_inserted_lastrecord=prev(sum_tup_inserted,1)
| extend tup_updated_lastrecord=prev(sum_tup_updated,1)
| extend tup_deleted_lastrecord=prev(sum_tup_deleted,1)
| extend commits_lastrecord=prev(sum_commits,1)
| extend rollbacks_lastrecord=prev(sum_rollbacks,1)
| extend total_transactions_lastrecord=prev(sum_total_transactions,1)
| extend deadlocks_lastrecord=prev(sum_deadlocks,1)
| extend tup_fetched_lastrecord=prev(sum_tup_fetched,1)
| extend tup_returned_lastrecord=prev(sum_tup_returned,1)
| extend temp_bytes_lastrecord=prev(sum_temp_bytes,1)
| extend temp_files_lastrecord=prev(sum_temp_files,1)
| project TIMESTAMP,
backendprocesses=sum_numbackends,
diff_tup_inserted=iff( (sum_tup_inserted-tup_inserted_lastrecord)<0,0, (sum_tup_inserted-
tup_inserted_lastrecord) ),
diff_tup_updated=iff( (sum_tup_updated-tup_updated_lastrecord)<0, 0,(sum_tup_updated-
tup_updated_lastrecord) ),
diff_tup_deleted=iff( (sum_tup_deleted-tup_deleted_lastrecord)<0,0, (sum_tup_deleted-
tup_deleted_lastrecord) ),
diff_commits=iff( (sum_commits-commits_lastrecord)<0,0, (sum_commits-commits_lastrecord) ),
diff_rollbacks=iff( (sum_rollbacks-rollbacks_lastrecord)<0,0, (sum_rollbacks-rollbacks_lastrecord)
),
diff_total_transactions=iff( (sum_total_transactions-total_transactions_lastrecord)<0,0,
(sum_total_transactions-total_transactions_lastrecord) ),
diff_deadlocks=iff( (sum_deadlocks-deadlocks_lastrecord)<0,0,(sum_deadlocks-deadlocks_lastrecord)),
diff_tup_fetched=iff( (sum_tup_fetched-tup_fetched_lastrecord)<0,0,(sum_tup_fetched-
tup_fetched_lastrecord)),
diff_tup_returned=iff( (sum_tup_returned-tup_returned_lastrecord)<0,0,(sum_tup_returned-
tup_returned_lastrecord)),
diff_temp_bytes=iff( (sum_temp_bytes-temp_bytes_lastrecord)<0,0,(sum_temp_bytes-
temp_bytes_lastrecord)),
diff_temp_files=iff( (sum_temp_files-temp_files_lastrecord)<0,0,(sum_temp_files-
temp_files_lastrecord))
| summarize
max(backendprocesses),sum(diff_tup_inserted),sum(diff_tup_updated),sum(diff_tup_deleted),sum(diff_co
```

```
mmits),sum(diff_rollbacks),
sum(diff_total_transactions),sum(diff_deadlocks),sum(diff_tup_fetched),sum(diff_tup_returned),sum(di
ff_temp_bytes),sum(diff_temp_files) by bin(TIMESTAMP,30m)
| render timechart
```



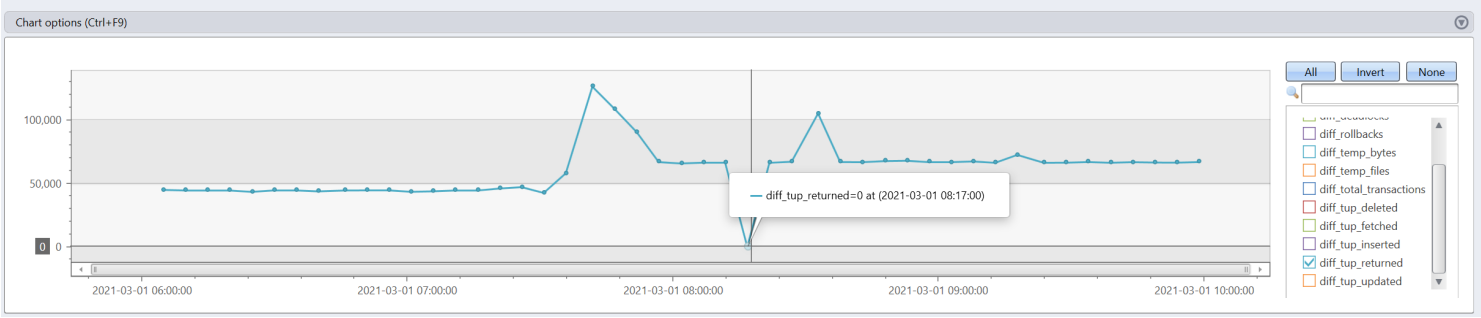
This statistics are taken from pg_stat_database view and the raw data in the Kusto table has the following fields:

Column	Description
datid	OID of a database
datname	Name of this database
numbackends	Number of backends currently connected to this database. This is the only column in this view that returns a value reflecting current state; all other columns return the accumulated values since the last reset.
tup_returned	Number of rows returned by queries in this database
tup_fetched	Number of rows fetched by queries in this database
tup_inserted	Number of rows inserted by queries in this database
tup_updated	Number of rows updated by queries in this database
tup_deleted	Number of rows deleted by queries in this database
temp_files	Number of temporary files created by queries in this database. All temporary files are counted, regardless of why the temporary file was created (e.g., sorting or hashing), and regardless of the log_temp_files setting.
temp_bytes	Total amount of data written to temporary files by queries in this database. All temporary files are counted, regardless of why the temporary file was created, and regardless of the log_temp_files setting.
deadlocks	Number of deadlocks detected in this database
blk_read_time	Time spent reading data file blocks by backends in this database, in milliseconds
blk_write_time	Time spent writing data file blocks by backends in this database, in milliseconds
stats_reset	Time at which these statistics were last reset

Caviats
In case it will be used a higher granularity (without the last summarize on 30min) it can be a situation where all the statistics drops to 0. The query uses the difference between current and previous value, therefore in case the

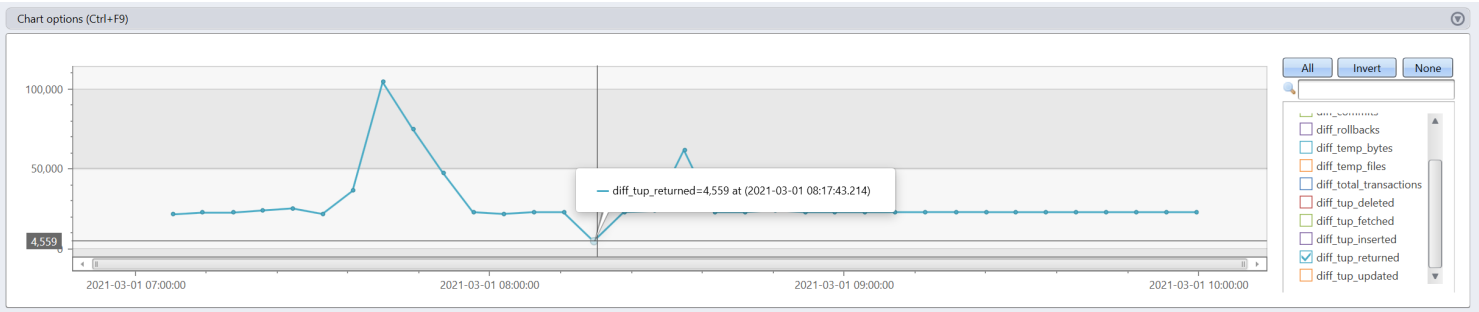
Customer resets the statistics for a database the difference will be negative, thus to avoid the noise, 0 was chosen instead. Ex I have reset the statistics and we can see the drop to 0 :

```
let TimeCheckStart = datetime(2021-03-01 06:00);
let TimeCheckEnd = datetime(2021-03-01 10:00);
let ServerName = 'postgre11';
MonDmPgSqlTransactionStats
 | where LogicalServerName =~ ServerName
 | where TIMESTAMP > TimeCheckStart and TIMESTAMP < TimeCheckEnd
| where database_name !in ('template1','template0', 'azure_sys','azure_maintenance')
| summarize sum(tolong(tup_inserted)),sum(tolong(tup_updated)),sum(tolong(tup_deleted)),sum(tolong(
sum(tolong(temp_bytes)),sum(tolong(temp_files)) by bin(TIMESTAMP,1m)
| order by TIMESTAMP asc
| serialize
| extend tup_inserted_lastrecord=prev(sum_tup_inserted,1)
| extend tup_updated_lastrecord=prev(sum_tup_updated,1)
| extend tup_deleted_lastrecord=prev(sum_tup_deleted,1)
| extend commits_lastrecord=prev(sum_commits,1)
| extend rollbacks_lastrecord=prev(sum_rollbacks,1)
| extend total_transactions_lastrecord=prev(sum_total_transactions,1)
| extend deadlocks_lastrecord=prev(sum_deadlocks,1)
| extend tup_fetched_lastrecord=prev(sum_tup_fetched,1)
| extend tup_returned_lastrecord=prev(sum_tup_returned,1)
| extend temp_bytes_lastrecord=prev(sum_temp_bytes,1)
| extend temp_files_lastrecord=prev(sum_temp_files,1)
| project TIMESTAMP,
 backendprocesses=sum_numbackends,
 diff_tup_inserted=iff( (sum_tup_inserted-tup_inserted_lastrecord)<0,0, (sum_tup_inserted-tup_inser
 diff_tup_updated=iff( (sum_tup_updated-tup_updated_lastrecord)<0, 0,(sum_tup_updated-tup_updated_l
 diff_tup_deleted=iff( (sum_tup_deleted-tup_deleted_lastrecord)<0,0, (sum_tup_deleted-tup_deleted_l
 diff_commits=iff( (sum_commits-commits_lastrecord)<0,0, (sum_commits-commits_lastrecord) ),
 diff_rollbacks=iff( (sum_rollbacks-rollbacks_lastrecord)<0,0, (sum_rollbacks-rollbacks_lastrecord)
 diff_total_transactions=iff( (sum_total_transactions-total_transactions_lastrecord)<0,0,(sum_total
 diff_deadlocks=iff( (sum_deadlocks-deadlocks_lastrecord)<0,0,(sum_deadlocks-deadlocks_lastrecord))
 diff_tup_fetched=iff( (sum_tup_fetched-tup_fetched_lastrecord)<0,0,(sum_tup_fetched-tup_fetched_la
 diff_tup_returned=iff( (sum_tup_returned-tup_returned_lastrecord)<0,0,(sum_tup_returned-tup_return
 diff_temp_bytes=iff( (sum_temp_bytes-temp_bytes_lastrecord)<0,0,(sum_temp_bytes-temp_bytes_lastrec
 diff_temp_files=iff( (sum_temp_files-temp_files_lastrecord)<0,0,(sum_temp_files-temp_files_lastrec
//| summarize max(backendprocesses),sum(diff_tup_inserted),sum(diff_tup_updated),sum(diff_tup_delet
//sum(diff_total_transactions),sum(diff_deadlocks),sum(diff_tup_fetched),sum(diff_tup_returned),sum
| render timechart
```



If really granularity is a key point and needed, the least error prone is to use below query for a single database. This version, does not use any aggregation functions, and when a negative difference is detected is not using the difference but the current value, thus it is not causing any noise. Besides that we can track here also the frozenxmin movement which I could not use it when aggregating the data for all databases. Below query shows the exact number of tuples or other statistics that occured within a single database on a 5 minute interval.

```
let TimeCheckStart = datetime(2021-03-01 07:00);
let TimeCheckEnd = datetime(2021-03-01 10:00);
let ServerName = 'postgre11';
MonDmPgSqlTransactionStats
 | where LogicalServerName =~ ServerName
 | where TIMESTAMP > TimeCheckStart and TIMESTAMP< TimeCheckEnd
| where database_name == 'replication'
| project TIMESTAMP,AppName,LogicalServerName,database_name,stats_reset,database_oid,tup_inserted,t
   deadlocks,blk_read_time,blk_write_time,numbackends,tup_fetched,tup_returned,temp_bytes,temp_fil
| order by TIMESTAMP asc
| serialize
| extend tup_inserted_lastrecord=prev(tup_inserted,1)
| extend tup_updated_lastrecord=prev(tup_updated,1)
| extend tup_deleted_lastrecord=prev(tup_deleted,1)
| extend commits_lastrecord=prev(commits,1)
| extend rollbacks_lastrecord=prev(rollbacks,1)
| extend total_transactions_lastrecord=prev(total_transactions,1)
| extend deadlocks_lastrecord=prev(deadlocks,1)
| extend blk_read_time_lastrecord=prev(blk_read_time,1)
| extend blk_write_time_lastrecord=prev(blk_write_time,1)
| extend tup_fetched_lastrecord=prev(tup_fetched,1)
| extend tup_returned_lastrecord=prev(tup_returned,1)
| extend temp_bytes_lastrecord=prev(temp_bytes,1)
| extend temp_files_lastrecord=prev(temp_files,1)
| extend age_frozenxid_lastrecord=prev(age_frozenxid,1)
| project TIMESTAMP,AppName,LogicalServerName,database_name,stats_reset,database_oid,
 backendprocesses=tolong(numbackends),
 diff_tup_inserted=iff( (tolong(tup_inserted)-tolong(tup_inserted_lastrecord))<0,tolong(tup_inserte
 diff_tup_updated=iff( (tolong(tup_updated)-tolong(tup_updated_lastrecord))<0,tolong(tup_updated),(
 diff_tup_deleted=iff( (tolong(tup_deleted)-tolong(tup_deleted_lastrecord))<0,tolong(tup_deleted),(
 diff_commits=iff( (tolong(commits)-tolong(commits_lastrecord))<0,tolong(commits),(tolong(commits)-
 diff_rollbacks=iff( (tolong(rollbacks)-tolong(rollbacks_lastrecord))<0,tolong(rollbacks),(tolong(r
 diff_total_transactions=iff( (tolong(total_transactions)-tolong(total_transactions_lastrecord))<0,
 diff_deadlocks=iff( (tolong(deadlocks)-tolong(deadlocks_lastrecord))<0,tolong(deadlocks),(tolong(d
 diff_blk_read_time=iff( (tolong(blk_read_time)-tolong(blk_read_time_lastrecord))<0,tolong(blk_read
 diff_blk_write_time=iff( (tolong(blk_write_time)-tolong(blk_write_time_lastrecord))<0,tolong(blk_w
 diff_tup_fetched=iff( (tolong(tup_fetched)-tolong(tup_fetched_lastrecord))<0,tolong(tup_fetched),(
 diff_tup_returned=iff( (tolong(tup_returned)-tolong(tup_returned_lastrecord))<0,tolong(tup_returne
 diff_temp_bytes=iff( (tolong(temp_bytes)-tolong(temp_bytes_lastrecord))<0,tolong(temp_bytes),(tolo
 diff_temp_files=iff( (tolong(temp_files)-tolong(temp_files_lastrecord))<0,tolong(temp_files), (tol
 diff_age_frozenxid=iff( (tolong(age_frozenxid)-tolong(age_frozenxid_lastrecord))<0,tolong(age_froz
| project TIMESTAMP, backendprocesses, diff_commits,diff_tup_inserted,diff_tup_updated,diff_tup_del
| render timechart
```



In order to find all databases Customer is having, use this query:

```
let TimeCheckStart = datetime(2021-03-01 07:00);
let TimeCheckEnd = datetime(2021-03-01 10:00);
let ServerName = 'postgre11';
MonDmPgSqlTransactionStats
 | where LogicalServerName =~ ServerName
 | where TIMESTAMP > TimeCheckStart and TIMESTAMP< TimeCheckEnd
 | where database_name !in ('template1','template0', 'azure_sys','azure_maintenance')
 | distinct database_name
```

database_name
postgres
replication
statsinsert

What we can learn from here

1. **Read queries**
   For the read queries we should look at diff_tup_fetched and diff_tup_returned

   tup_returned will be the number of the rows scanned or read, but it is not what is returned to the client. If I run a query like:
   select * from tbl limit 10; -- if table has 10k tuples, tup_returned = 10.000, tup_fetched=10 as 10 tuples were returned to the client.

   If tup_fetched == tup_returned and is a very big number => CX is doing select * from tbl;

   If tup_returned >>>> tup_fetched, then may be CX has poor written queries(probably a lot of joins, lack of indexes and many reads to provide a certain result)

2. **Write queries**
   For the write queries we should look at diff_tup_inserted, diff_tup_updated and diff_tup_deleted. I think the name are self explanatory

3. **Transaction statistics**
   To understand CX transactions pattern and the amount of them, look at: diff_commits, diff_rollbacks, diff_total_transactions

   This can be very helpfull in certain situations: see if CPU is not increasing in the same time; also for the replication if there is a very high number of inserts/deletes/updates and immediately after we see that replication lag starts to increase, then we can see the correlation and the cause of the replica lag is the workload itself

4. **Background processes**
   To monitor the number of background processes across time use: backendprocesses

5. **Deadlocks**
   To monitor if there was any deadlocks use: diff_deadlocks

6. **Temporary files**
   To monitor if temporary files or bytes occured use : diff_temp_bytes and diff_temp_files

If you see a certain pattern here or if CX is having daily temporary files, it could be an indication that the joins, sorts, orders could benefit of increasing work_mem parameter, therefore there will be less spills on disk. A join, sort, order is always faster when performed in memory comparing to the one performed on disk,

Another way of understanding Cutomer workload is seeing if the slope is growing.

The below query plots the number of inserts, deletes, updates at every 1 hour interval.
An increasing rate of growth during any period of time should show an increased amount of activity.
It is the same data as in earlier query, but just shown as a single line.
If it is going up rapidly, it means more activity.

```
//EastUS2
MonDmPgSqlTransactionStats
| where LogicalServerName == "snipgsqlpr01"
| where database_name <> 'azure_maintenance' and database_name <> 'azure_sys'
| project PreciseTimeStamp, LogicalServerName, database_name, database_oid, tup_inserted,
tup_deleted, tup_updated, tup_fetched, tup_returned
| order by PreciseTimeStamp asc
| summarize max(tolong(tup_inserted)), max(tolong(tup_deleted)), max(tolong(tup_updated)) by
bin(PreciseTimeStamp, 1h)
| render timechart
```