DataEng: Data Storage Activity

Genevieve LaLonde
Data Engineering Winter 2021
Bruce Irvin
12 Feb 2021

I. Results

Use this table to present your results. We are not asking you to do a sophisticated performance analysis here with multiple runs, warmup time, etc. Instead, do a rough measurement using timing code similar to what you see in the load_inserts.py code. Record your results in the following table.

Method	**Code Link**	**acs2015**	**acs2017**
Simple inserts	load_inserts.py	Elapsed Time: 90.66 seconds	Elapsed Time: 93.73 seconds
Drop Indexes and Constraints	no_index.py delay_index.py	Elapsed Time when indexes are simply removed: 90.07 seconds Elapsed Time when delaying adding indexes till after load: 131.5 seconds	Elapsed Time when indexes are simply removed: 96.5 seconds 2nd run when indexes are simply removed: Elapsed Time: 90.64 seconds Elapsed Time when delaying adding indexes till after load: 119.8 seconds
Use UNLOGGED table	unlogged_set_logged.py unlogged_insert_select.py	Time for unlogged, set as logged, add indexes: 17.78 s Time for unlogged, insert select to CensusData: 18.57 s	Time for unlogged, set as logged, add indexes: 17.79 s Time for unlogged, insert select to CensusData: 17.78 s

Temp Table with memory tuning	temp_default_mem.py temp_extended_mem.py	Time with default memory: 17.59 s Extended memory: 17.62 seconds	Time with default memory: 17.64 s Extended memory:
Batching			
copy_from			

J. Observations

Use this section to record any observations about the various methods/techniques that you used for bulk loading of the USA Census data. Did you learn anything about how and RDBMS functions and why various loading approaches produce varying performance results?

Temp and unlogged tables performed the best, and were comparable to each other, even given variations like memory allocation and whether an unlogged table is subsequently set as logged, or used as the source of an INSERT SELECT statement.
Not creating indexes, or delaying their creation did not significantly impact performance. Perhaps this is related to the fact that the primary key was numeric data types, and the values of the text index is only 2 characters. So they would be performant to create. Might be interesting to repeat, with indexes on text columns with longer values and see if there is a larger impact.

Complications

Complication #1:
2015 matched expected header names
CensusTract,State,County,TotalPop,Men,Women,Hispanic,White,Black,Native,Asian,Pacific,Citizen,Income,IncomeErr,IncomePerCap,IncomePerCapErr,Poverty,ChildPoverty,Professional,Service,Office,Construction,Production,Drive,Carpool,Transit,Walk,OtherTransp,WorkAtHome,MeanCommute,Employed,PrivateWork,PublicWork,SelfEmployed,FamilyWork,Unemployment

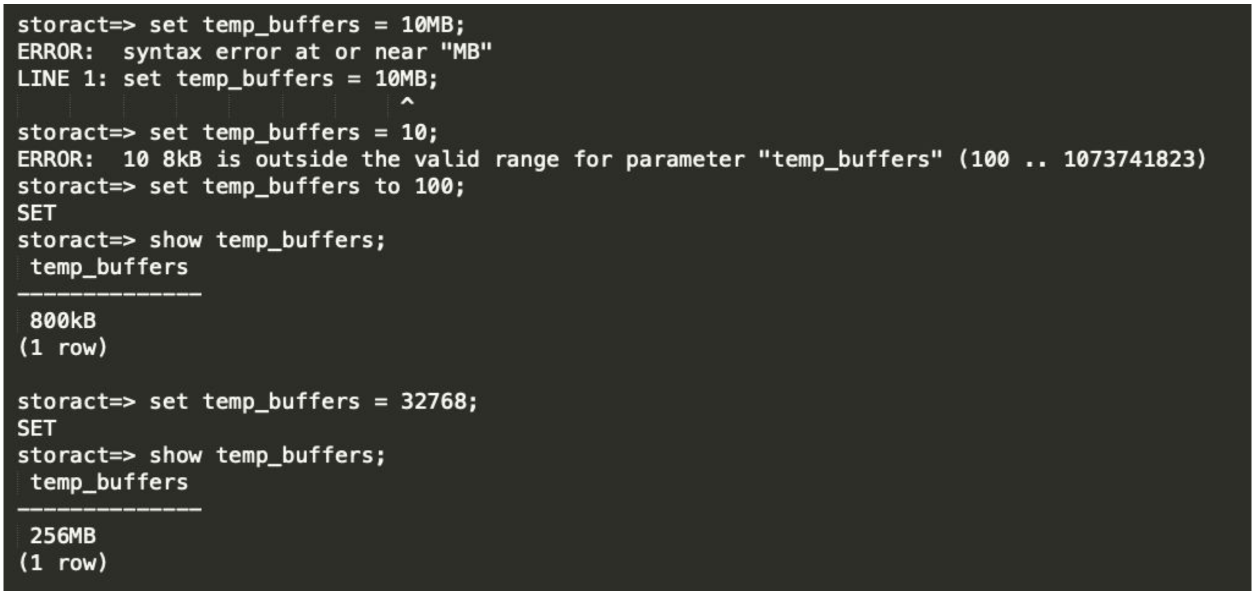
2017 had a changed name for the first column
TractId,State,County,TotalPop,Men,Women,Hispanic,White,Black,Native,Asian,Pacific,VotingAgeCitizen,Income,IncomeErr,IncomePerCap,IncomePerCapErr,Poverty,ChildPoverty,Professional,Service,Office,Construction,Production,Drive,Carpool,Transit,Walk,OtherTransp,WorkAtHome,MeanCommute,Employed,PrivateWork,PublicWork,SelfEmployed,FamilyWork,Unemployment

Resolution:

Rename TractId->CensusTract in 2017 file.

Complication #2:

Setting the temp_buffer requires math:



```
storact=> set temp_buffers = 10MB;
ERROR:  syntax error at or near "MB"
LINE 1: set temp_buffers = 10MB;
                             ^
storact=> set temp_buffers = 10;
ERROR:  10 8kB is outside the valid range for parameter "temp_buffers" (100 .. 1073741823)
storact=> set temp_buffers to 100;
SET
storact=> show temp_buffers;
 temp_buffers
--------------
 800kB
(1 row)

storact=> set temp_buffers = 32768;
SET
storact=> show temp_buffers;
 temp_buffers
--------------
 256MB
(1 row)
```

Calculated as 256 * 1024 / 8 = 32768 (KB)