cd H:\InProgress\CSVTest\Chrissy

Import-Module .\CsvSqlimport.psm1 -Force

Import-CsvToSql -Csv H:\InProgress\CSVTest\Chrissy\allCountries\1milCountries.txt -SqlServer WIN-N53LUO6QQ58\SQL14 -Database tempdb -Table allCountriesChrissy -Delimiter "`t" -FirstRowColumns -Truncate -Turbo

if table doesn't exist => creates all columns as VARCHAR(MAX)

**2.65** million rows in a minute

[\*] 2650000 rows have been inserted in 60.91 seconds

if table does

[\*] Truncating table

[\*] Starting bulk copy for allCountries.txt

[\*] 50000 rows have been inserted in 1.26 seconds

[\*] 100000 rows have been inserted in 2.45 seconds

[\*] 150000 rows have been inserted in 3.57 seconds

[\*] 200000 rows have been inserted in 4.7 seconds

[\*] 250000 rows have been inserted in 5.98 seconds

[\*] 300000 rows have been inserted in 7.08 seconds

[\*] 350000 rows have been inserted in 8.22 seconds

[\*] 400000 rows have been inserted in 9.4 seconds

[\*] 450000 rows have been inserted in 10.57 seconds

[\*] 500000 rows have been inserted in 11.74 seconds

[\*] 550000 rows have been inserted in 12.84 seconds

[\*] 600000 rows have been inserted in 14.01 seconds

[\*] 650000 rows have been inserted in 15.21 seconds

[\*] 700000 rows have been inserted in 16.33 seconds

[\*] 750000 rows have been inserted in 17.44 seconds

[\*] 800000 rows have been inserted in 18.55 seconds

[\*] 850000 rows have been inserted in 19.71 seconds

[\*] 900000 rows have been inserted in 20.9 seconds

[\*] 950000 rows have been inserted in 22 seconds

[\*] 1000000 rows have been inserted in 23.08 seconds

[\*] 1050000 rows have been inserted in 24.19 seconds

[\*] 1100000 rows have been inserted in 25.32 seconds

[\*] 1150000 rows have been inserted in 26.47 seconds

[\*] 1200000 rows have been inserted in 27.63 seconds

[\*] 1250000 rows have been inserted in 28.75 seconds

[\*] 1300000 rows have been inserted in 29.87 seconds

[\*] 1350000 rows have been inserted in 31.06 seconds

[\*] 1400000 rows have been inserted in 32.17 seconds

[\*] 1450000 rows have been inserted in 33.28 seconds

[\*] 1500000 rows have been inserted in 34.47 seconds

[\*] 1550000 rows have been inserted in 35.57 seconds

[\*] 1600000 rows have been inserted in 36.72 seconds

[\*] 1650000 rows have been inserted in 37.86 seconds

[\*] 1700000 rows have been inserted in 39.01 seconds

[\*] 1750000 rows have been inserted in 40.16 seconds

[\*] 1800000 rows have been inserted in 41.35 seconds

[\*] 1850000 rows have been inserted in 42.45 seconds

[\*] 1900000 rows have been inserted in 43.55 seconds

[\*] 1950000 rows have been inserted in 44.68 seconds

[\*] 2000000 rows have been inserted in 45.81 seconds

[\*] 2050000 rows have been inserted in 46.91 seconds

[\*] 2100000 rows have been inserted in 48.04 seconds

[\*] 2150000 rows have been inserted in 49.23 seconds

[\*] 2200000 rows have been inserted in 50.36 seconds

[\*] 2250000 rows have been inserted in 51.62 seconds

[\*] 2300000 rows have been inserted in 52.77 seconds

[\*] 2350000 rows have been inserted in 53.94 seconds

[\*] 2400000 rows have been inserted in 55.08 seconds

[\*] 2450000 rows have been inserted in 56.23 seconds

[\*] 2500000 rows have been inserted in 57.41 seconds

[\*] 2550000 rows have been inserted in 58.61 seconds

[\*] 2600000 rows have been inserted in 59.79 seconds

[\*] 2650000 rows have been inserted in 60.91 seconds

Agenda:

What is PowerShell?

Things that make csv import faster:

In new: **$reader = New-Object System.IO.StreamReader($file)**

In existing: $columnparser = New-Object Microsoft.VisualBasic.FileIO.TextFieldParser($parserfile)

In both: **$datatable = New-Object System.Data.DataTable**

In new:

**$bulkcopy = New-Object Data.SqlClient.SqlBulkCopy($sqlconn,"Default",$transaction)**

**$bulkcopy.WriteToServer($datatable)**

#generic column adding (not hardcoded each column name), use .Add rather than +=

In both: use generic columns **$row = $datatable.Rows.Add($line.Split($delimiter))**

In both: **batch it!** and find the best batch size

More speed tips:

* Use DB simple recovery model
* Find the best batch size
* Use accurate datatypes. we probably can't know datatypes? also didn't make a difference in speed
* Remove write outputs - I didn't notice any speed gain with outputs in every 50 000 at least
* Use tablelock
* Runspaces: it creates a new thread on the existing powershell.exe host. Enables running multiple commands concurrently, with each command invoked in a different runspace.

Result:

from 1.7m rows a minute to 2.65m rows a minute - that is 36%

Chrissy: 5.3 million a minute "All of my testing took place in an optimized lab enviornment with no indexes, no other users and smoking fast SSD"

4 core 3.9GHz, 32GB RAM, 512GB Local SSD.

TODO: try on a different db