## Shared Bench-Marking Steps

### Populate Database “Tpcc”

When running the relevant TPC-C application for each database system, a windows form should appear after initiating the .exe file as seen in Figure 2. The populating the database begins with the “Recreate DB” button. This opens another cmd window which displays the progress of the creation of the database “Tpcc” as seen in Figure 3. Once the initial database structure is created and items are added, the “Add Stock” button is used to fill the stock in each warehouse. Once this is finished, “Fill District(s)” is used to complete the population of the database by filling the customer and district tables .

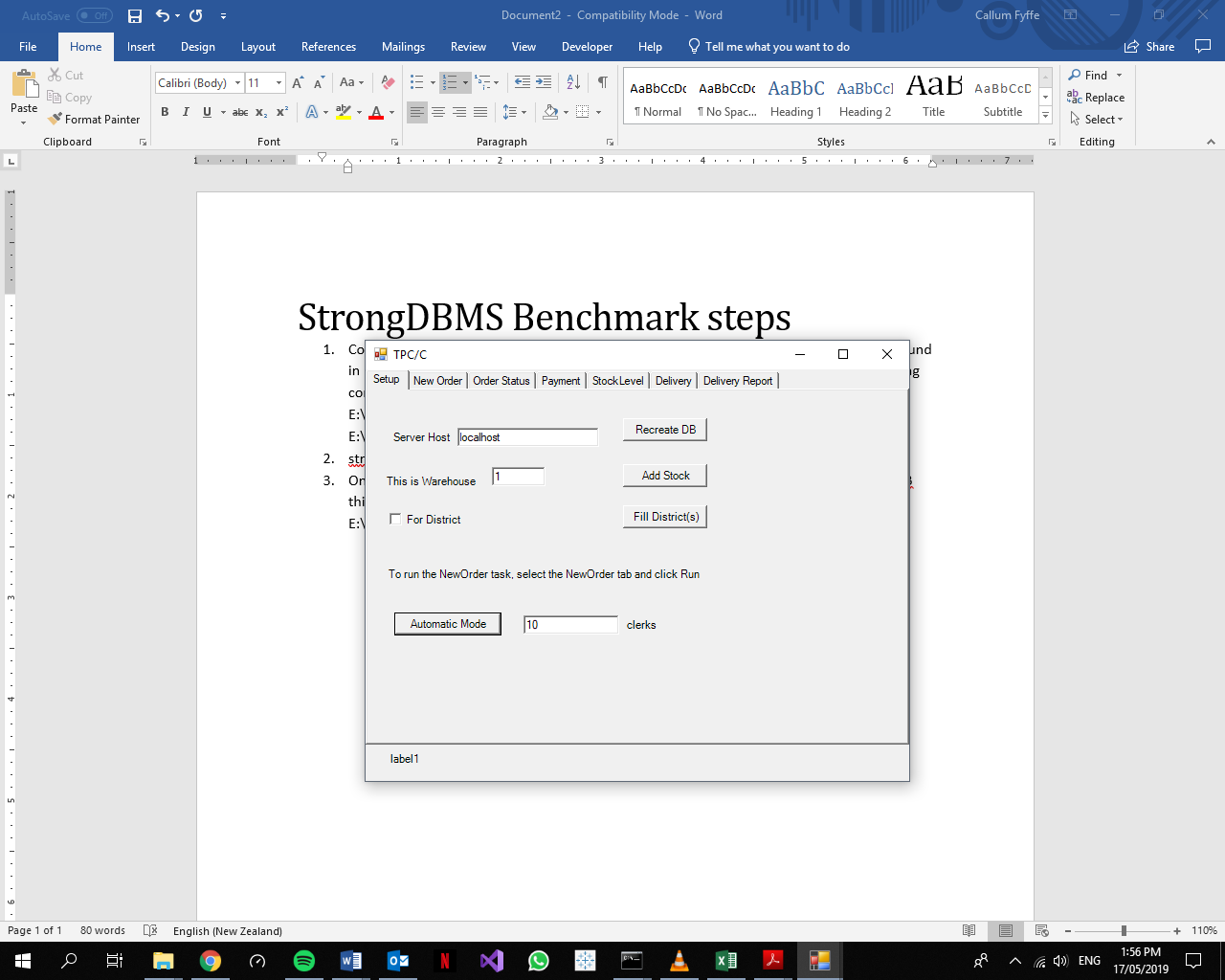


Figure 2: TPC/C Application Window

Figure 3 is an example of what should appear in the TPC-C cmd prompt window when the database is fully populated with 1 warehouse.

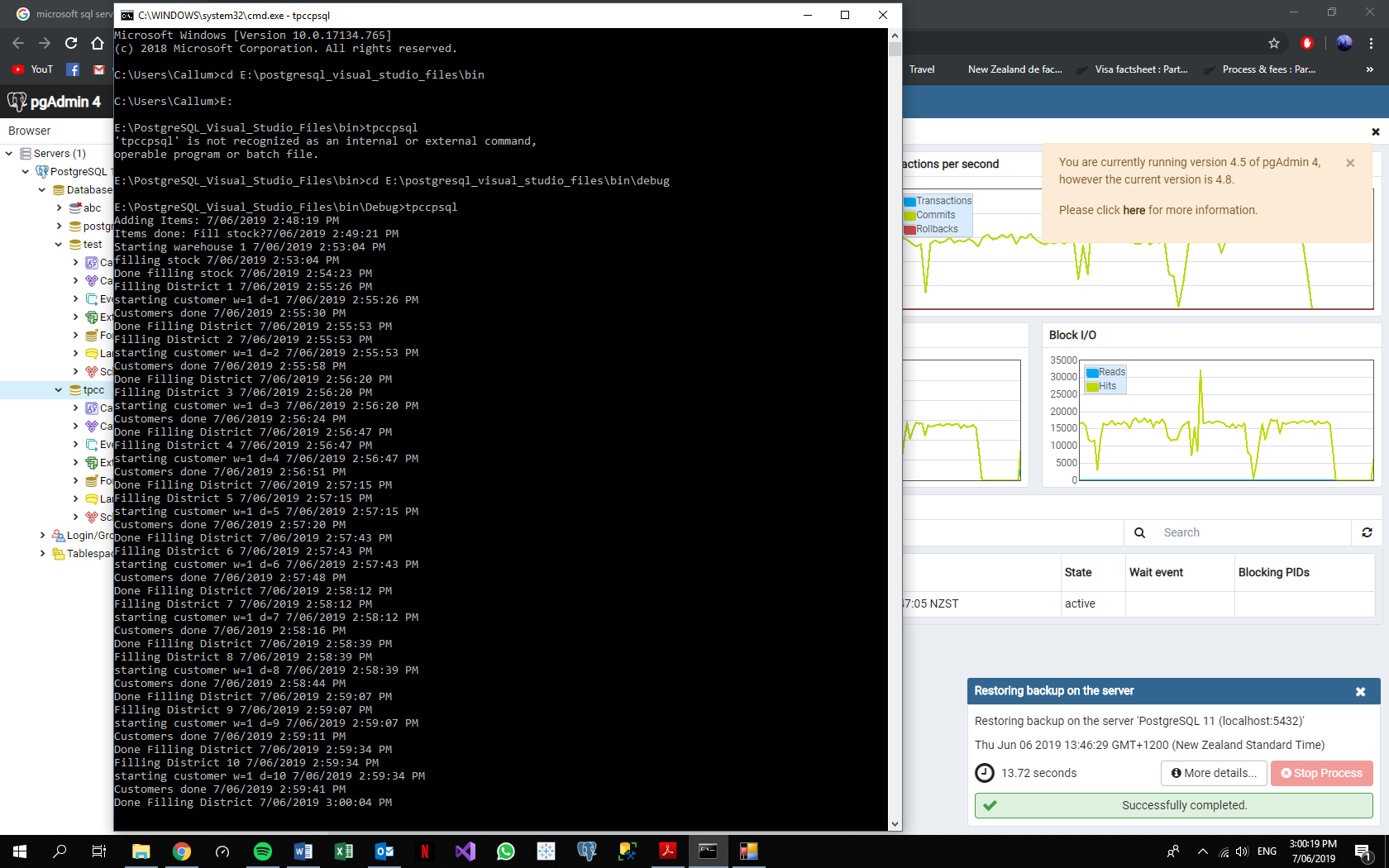


Figure 3: TPC-C populating a PostgreSQL Database

### Run Benchmark (Automatic Mode)

The “Automatic Mode” button in Figure 2 will begin the benchmark. This will simulate a number of clerks attempting to execute transactions with the database “Tpcc”. The cmd prompt window that opens with Tpcc.exe is designed to display statistics of the current benchmark once it is completed. These include the number of clerks being used, a timestamp, the number of commits to the NEW\_ORDER table, the number of read conflicts, the number of write conflicts, and a timestamp 10 minutes after the initialisation of the benchmark. Figure 4 is an example of this used on Strong.

The requests.txt file found in E:\ShareableDataStructures-master\0.1\bin should be stored somewhere else with an appropriate name (ie. “Strong\_requests\_10” for a 10 clerk TPC-C benchmark on Strong) so it can be analysed manually for anomalies. Copying this file must be done before another test is initialised as it will be over-written during the next benchmark. For Strong, the server must be disconnected to access the requests.txt file.

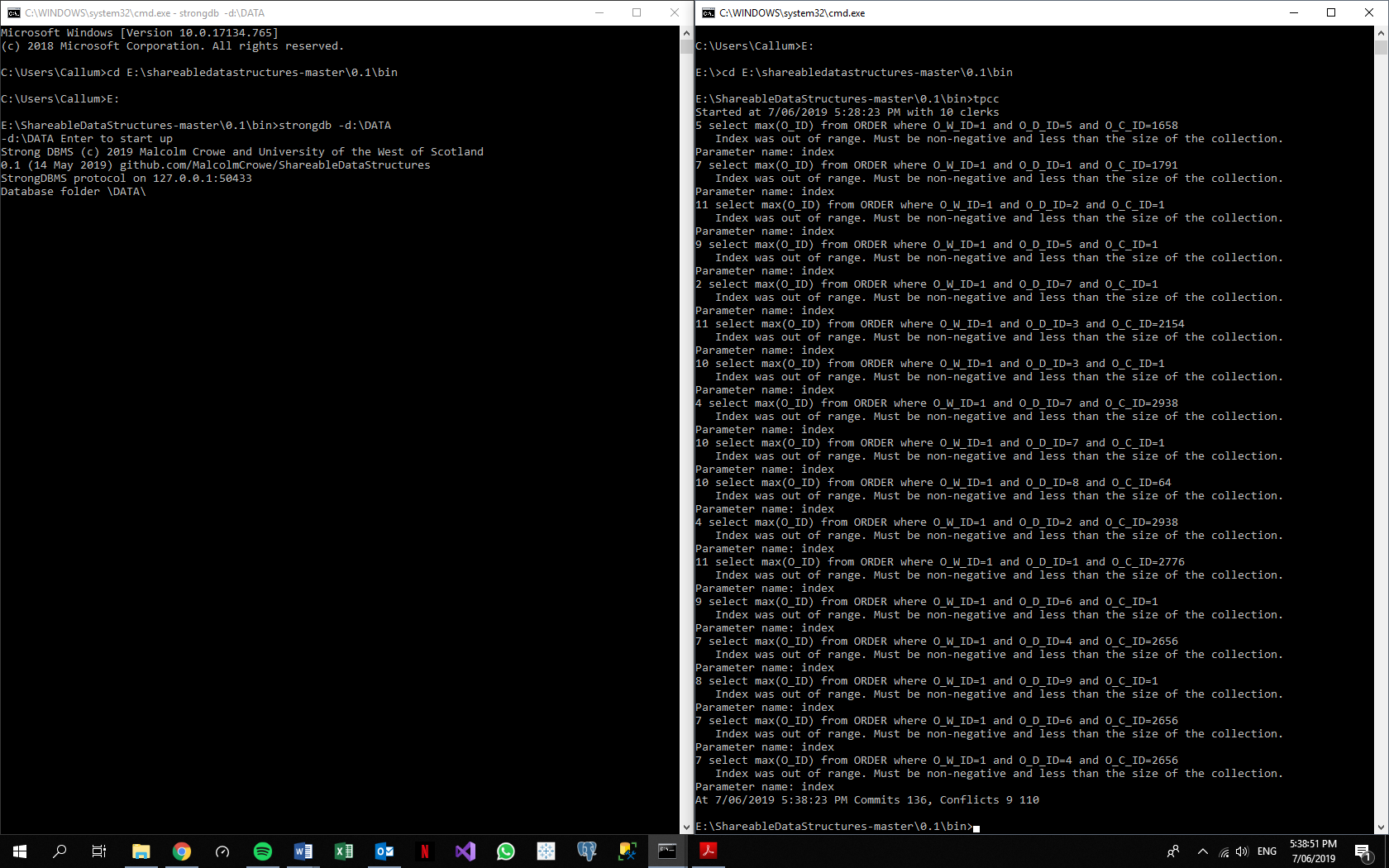


Figure 4: TPC/C cmd Window After a Strong Benchmark Test

## StrongDBMS Benchmark steps

### Server Initialisation

The Strong server is initialised using windows cmd prompt. In the case of Figure 5, the most up to date repository is found in E:\ShareableDataStructures-master\0.1\bin. As the default directory is set to the C: drive, the directory must be changed by entering the command “cd” followed by the desired path. “E:” is entered to change the path to the E: drive. As seen in Figure 5, “strongdb” is entered to run the executable file, “-d:\DATA” selects the path to access database files from the folder “DATA” (E:\DATA). The folder “DATA” is created before server initialisation to store the database “tpcc”. The user is then prompted to hit the “Enter” key on the keyboard to finalise the initialisation.

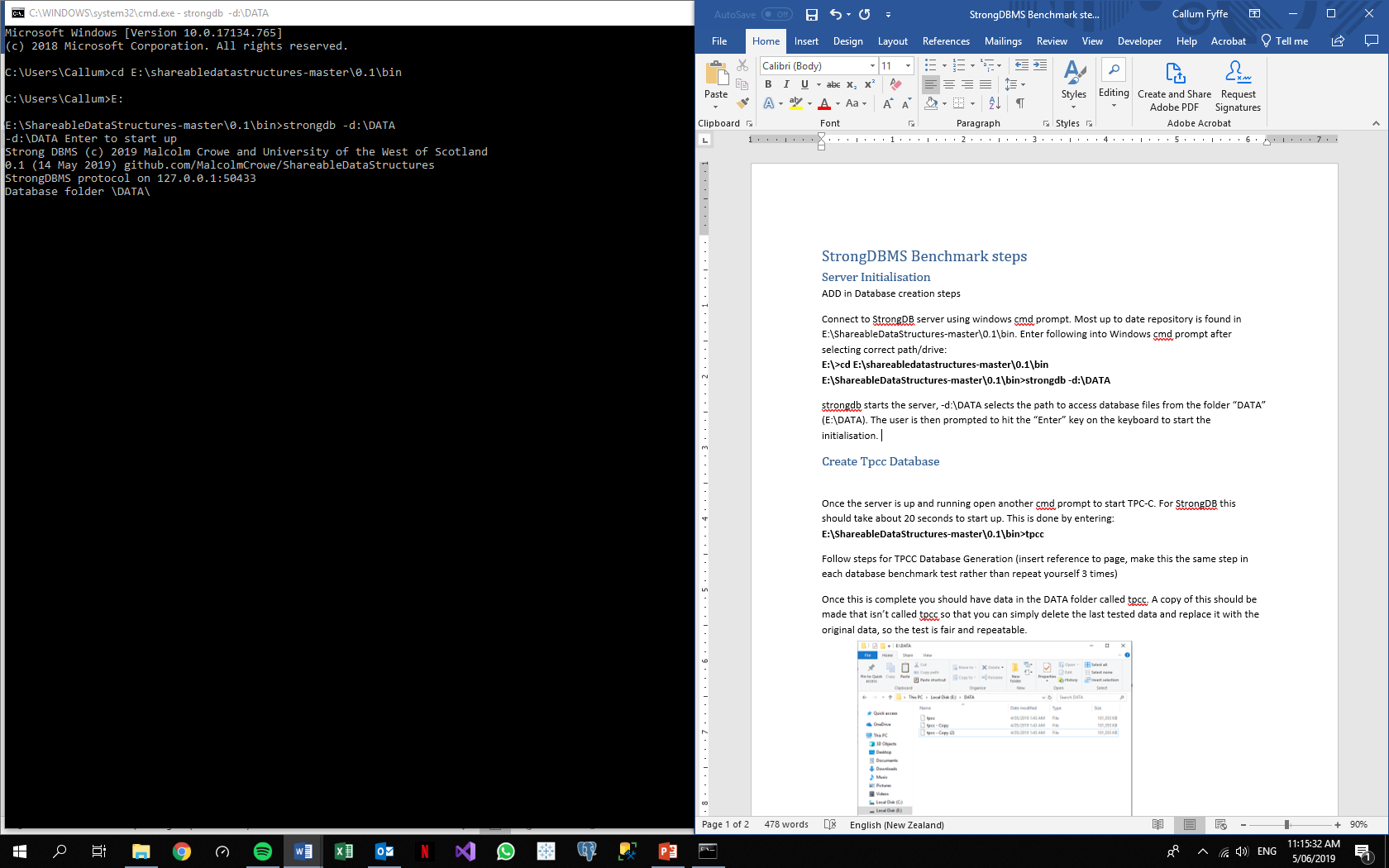


Figure 5: Strong Server Initialisation

### Create Tpcc Database

Once the server is running, another cmd prompt is opened to start the TPC-C application. For Strong this should take about 20 seconds to start up. The TPC-C application for Strong is found in the same path as the server initialisation file, “E:\ShareableDataStructures-master\0.1\bin” and the Strong TPC-C executable is called “tpcc.exe”. The TPC-C application creates a database called “tpcc” by itself when the “Recreate DB” button seen in Figure 2 is pressed.

### Populate Tpcc

Refer to common Populate Database “Tpcc” section.

### Backup Database

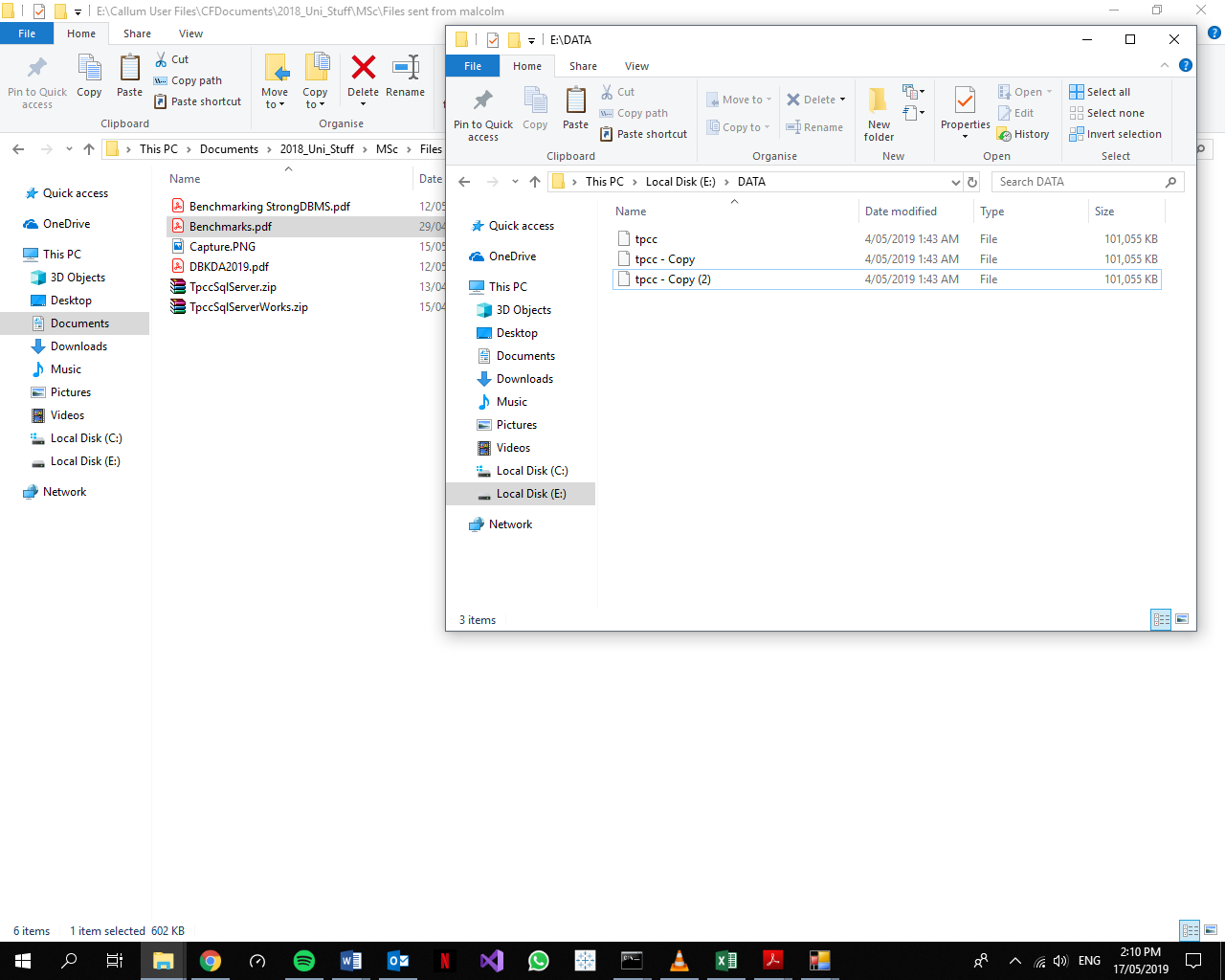
Once the database is created and populated, a file in the “DATA” folder called “tpcc”, as seen in Figure 6, appears. The TPC-C application recognises this name and uses it for the benchmark. A copy of this is made that isn’t called “tpcc” so that the application doesn’t use it for the TPC-C test. This copy is used later on to restore the database to its original state.

Figure 6: DATA Folder

### Query for Table Condition

Before testing, the original state of the database is found by querying it. To open a client application window, the “strongcmd.exe” file must be executed from windows cmd prompt and told to open the database “tpcc”. The following query needs to be entered to view the status of the tables:

select from \_Tables

The resulting table is copied by left clicking and dragging the mouse over the table and pressing “ctrl” + “c”. The cmd lines will be highlighted white. If the table is not highlighted white before pressing “ctrl” + “c”, the application will disconnect from the Tpcc database as it is a Windows keyboard shortcut for closing applications in the cmd window. The table is copied and pasted into a word file to keep for comparison. As seen in Figure 7, using Strong SQL syntax is used to find the number of tuples in tpcc’s tables.

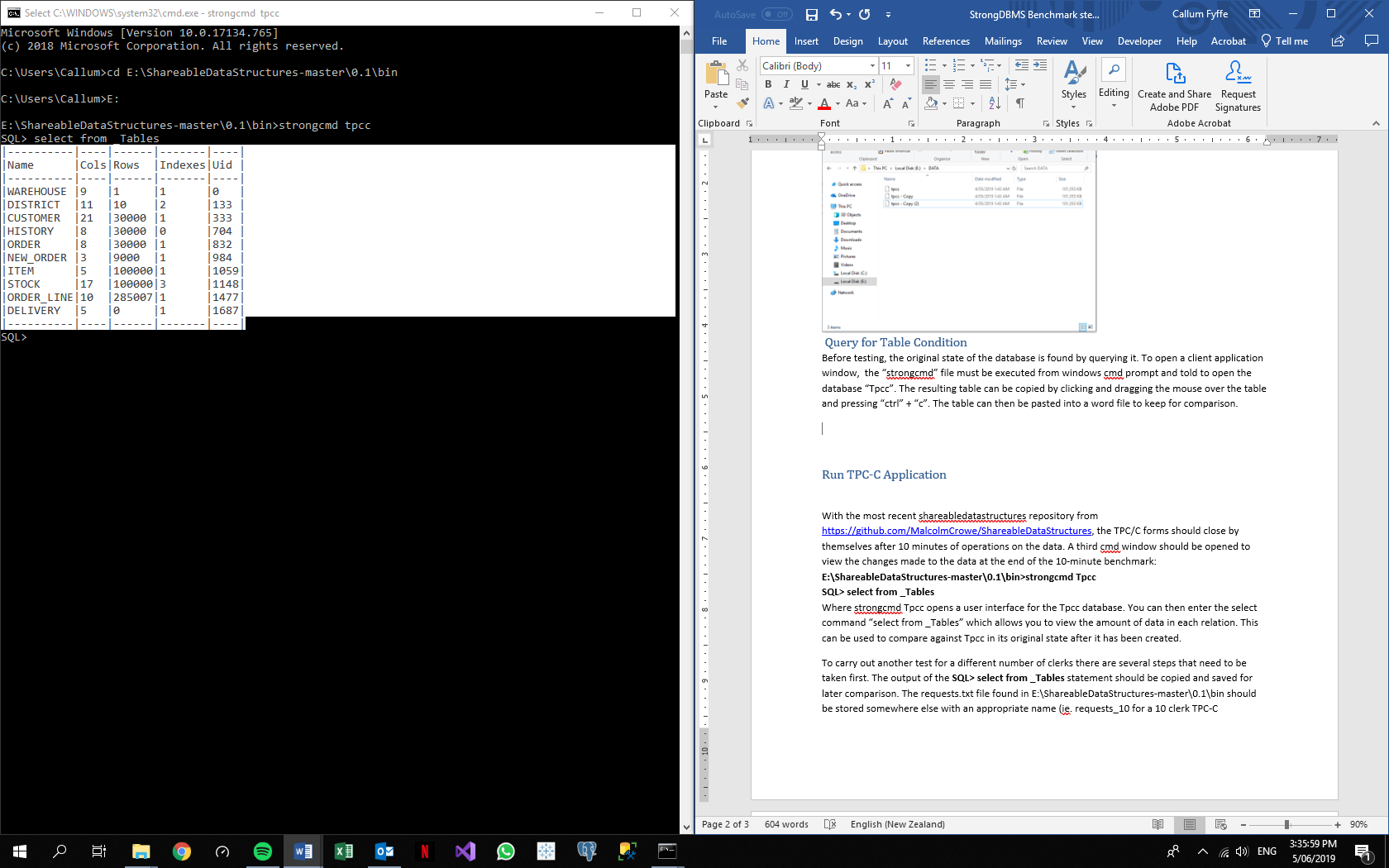


Figure 7: Strong Query for Table Condition

### Run TPC-C Application

Refer to common Run TPCC section.

### Result Collection

The query displayed in Figure 7 is run to display the tables’ properties then copied into a word document to be analysed later.

### Restore Database

The file named “Tpcc” in the “E:\DATA” folder is deleted after all instances of the Database are closed. A copy of the database is then renamed “Tpcc” for future tests.

## MS SQL Server Benchmark Steps

### Server Initialisation

The SQL Server is Initialised from Microsoft SQL Server Management Studio. The user is prompted to connect to the local server that is created upon installation as seen in Figure 8.

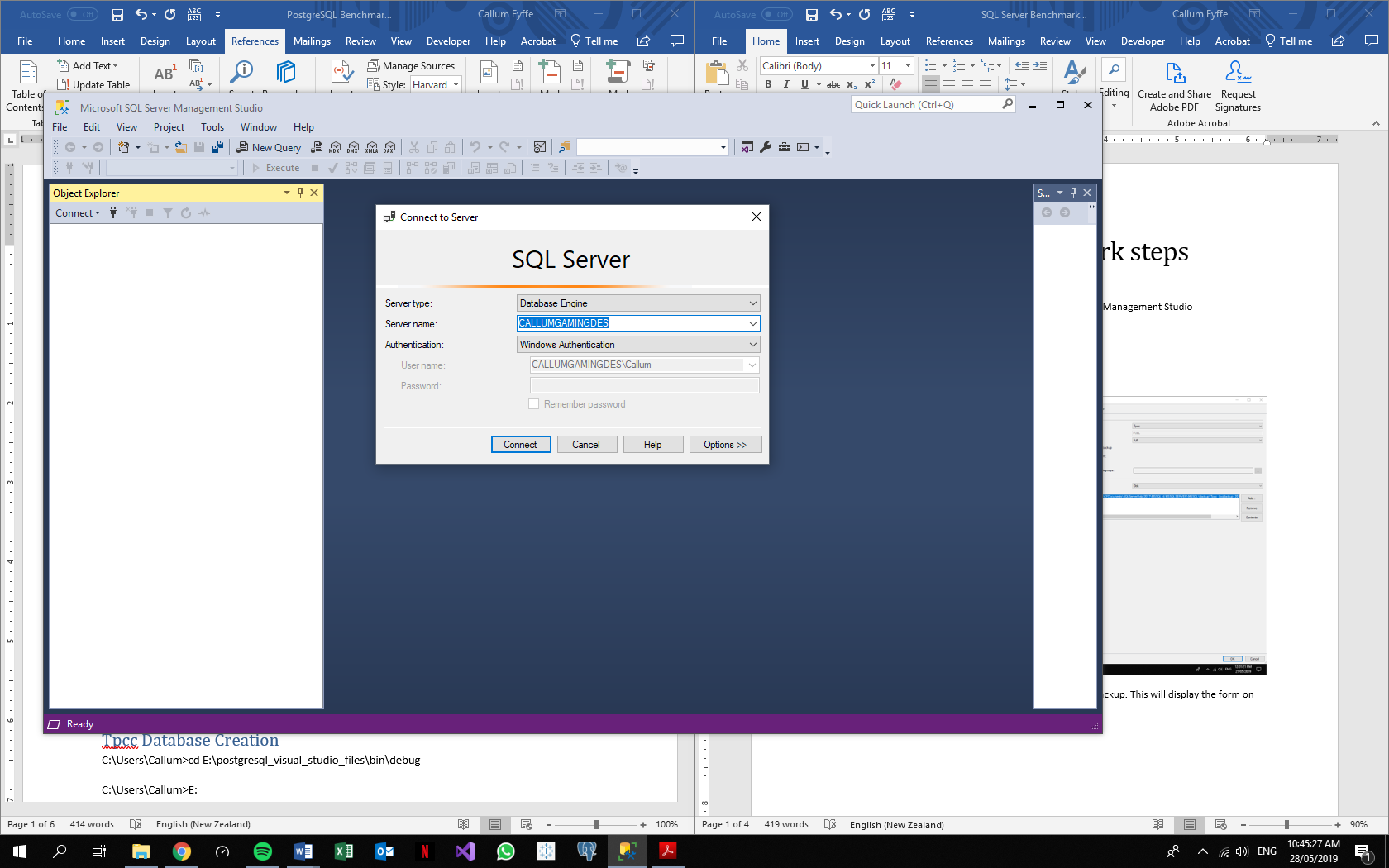


Figure 8: Microsoft SQL Server Management Studio user interface

### Create Tpcc Database

Right clicking on the “Databases” folder in the object explorer will give the option to create a new database. This must be called “Tpcc” for the TPC-C application to recognise it.

### Populate Tpcc

Refer to common Populate “Tpcc” Database section.

### Backup Database

By right clicking on database “Tpcc”, hovering over “Tasks” then, selecting “Backup” displays the form on the right-hand side of Figure 9. Clicking “OK” then begins the backup. This will take a few seconds to process and complete.

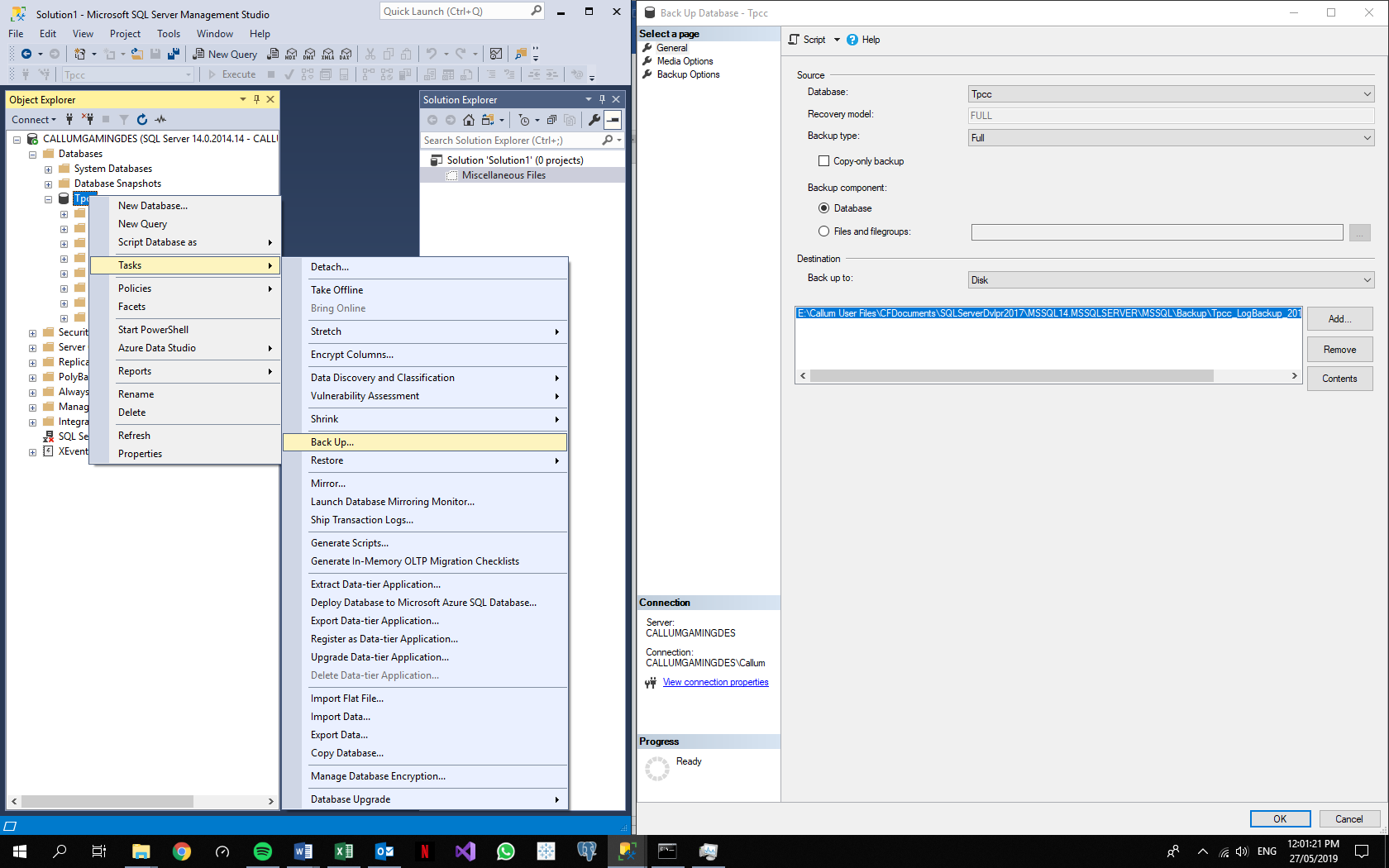


Figure 9: SQL Server Database Backup

### Query for Table Condition

The original state of the table must be known before testing begins. The “New Query” button on the toolbar displays the query window into which the following query is entered to display the number of rows in the tables of the database “Tpcc”:

SELECT t.name, s.row\_count from sys.tables t

JOIN sys.dm\_db\_partition\_stats s

ON t.object\_id = s.object\_id

AND t.type\_desc = 'USER\_TABLE'

AND t.name not like '%dss%'

AND s.index\_id IN (0,1)

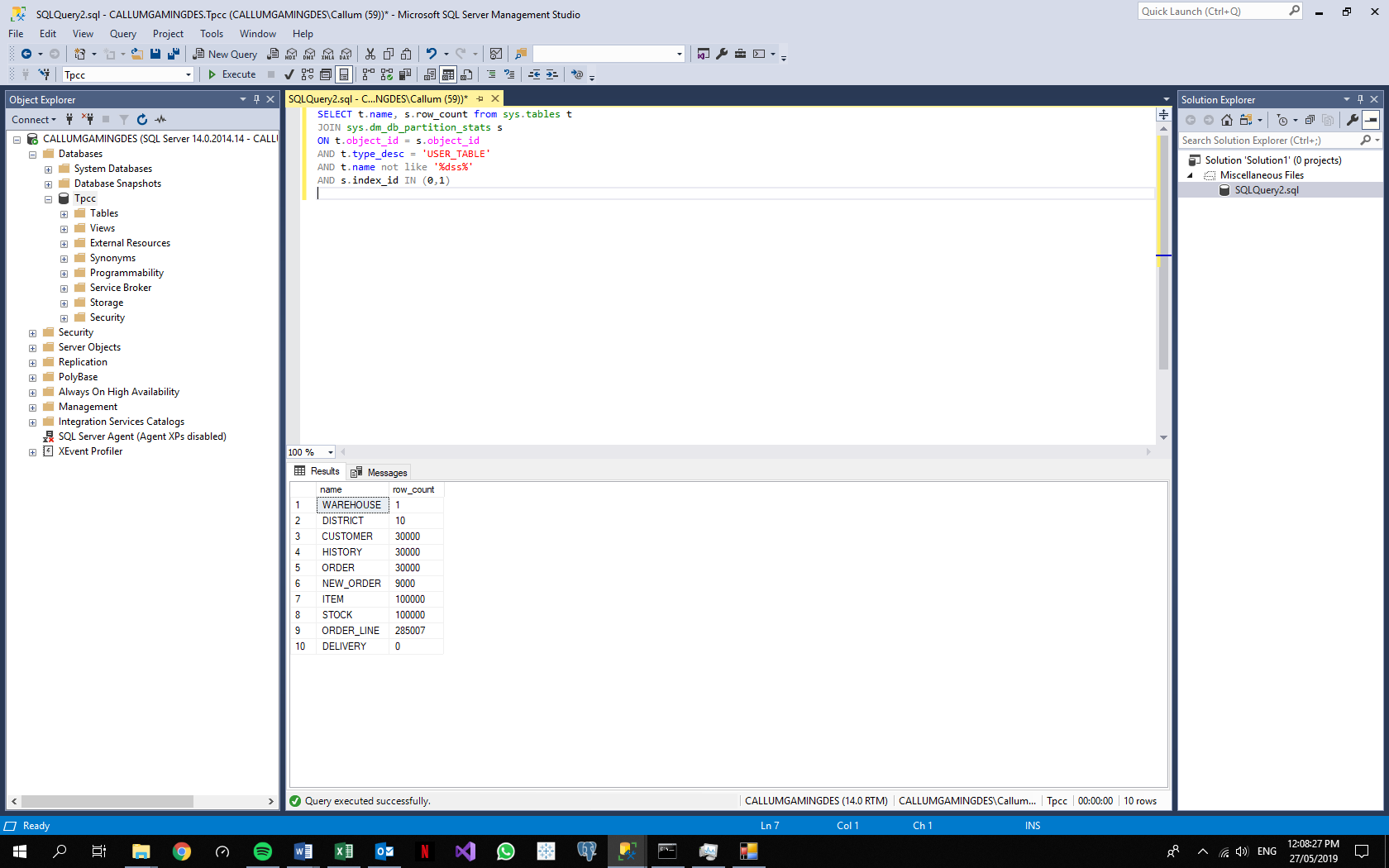
The “Execute” button is left clicked to run the query and this should display a table as seen in the bottom window of Figure 10. This is copied to a word document by left clicking in the top left-hand corner of the resultant table to highlight the entire table. This is copied to be used for comparison with the other tests. 

Figure 10: SQL Server Tables-Rows Query

### Run TPC-C Application

Refer to common Run TPCC section.

### Result Collection

The query is run again, and the resultant table is copied, as before, into a word document.

If the windows form running the benchmark does not close at the 10-minute mark, then it has to be assumed that the automatic mode is still running and adding rows to the tables. If this occurs, then any query regarding the number of rows in the table can be regarded as inaccurate. The benchmark must stop running, or be manually terminated, on the 10-minute mark for any queries thereafter to be considered fair for comparison.

### Restore database

To begin restoration of the “Tpcc” database, “Tpcc” is selected with the right click, “Tasks” is hovered over, then “Restore” is hovered over before left clicking on “Database” as shown on the left window pane of Figure 11. This displays the form on the right of Figure 11. The transaction log is deselected as the database needs to be restored from its original state. Note: the database will not restore if a query execution window is open. After this is complete, the benchmark can be ran again using the database from the original state.

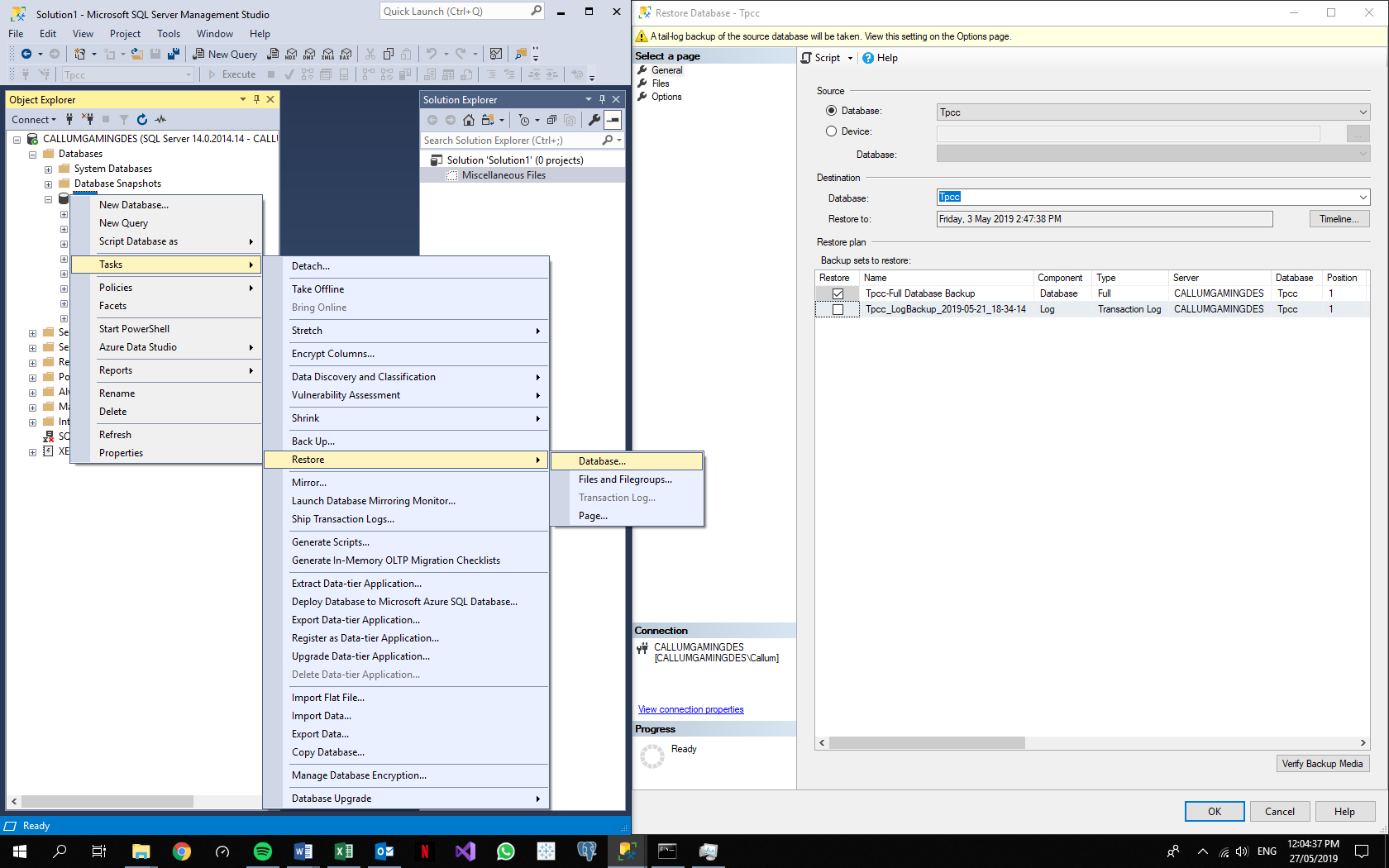


Figure 11: SQL Server Restore Database

## PostgreSQL Benchmark Steps

### Server Initialisation

The PostgreSQL Server is initialised by opening the application pgAdmin. This opens a page in the default browser (Figure 12 ) of the computer which displays interface to carry out administration actions. Once this is open, the server has been initialised.

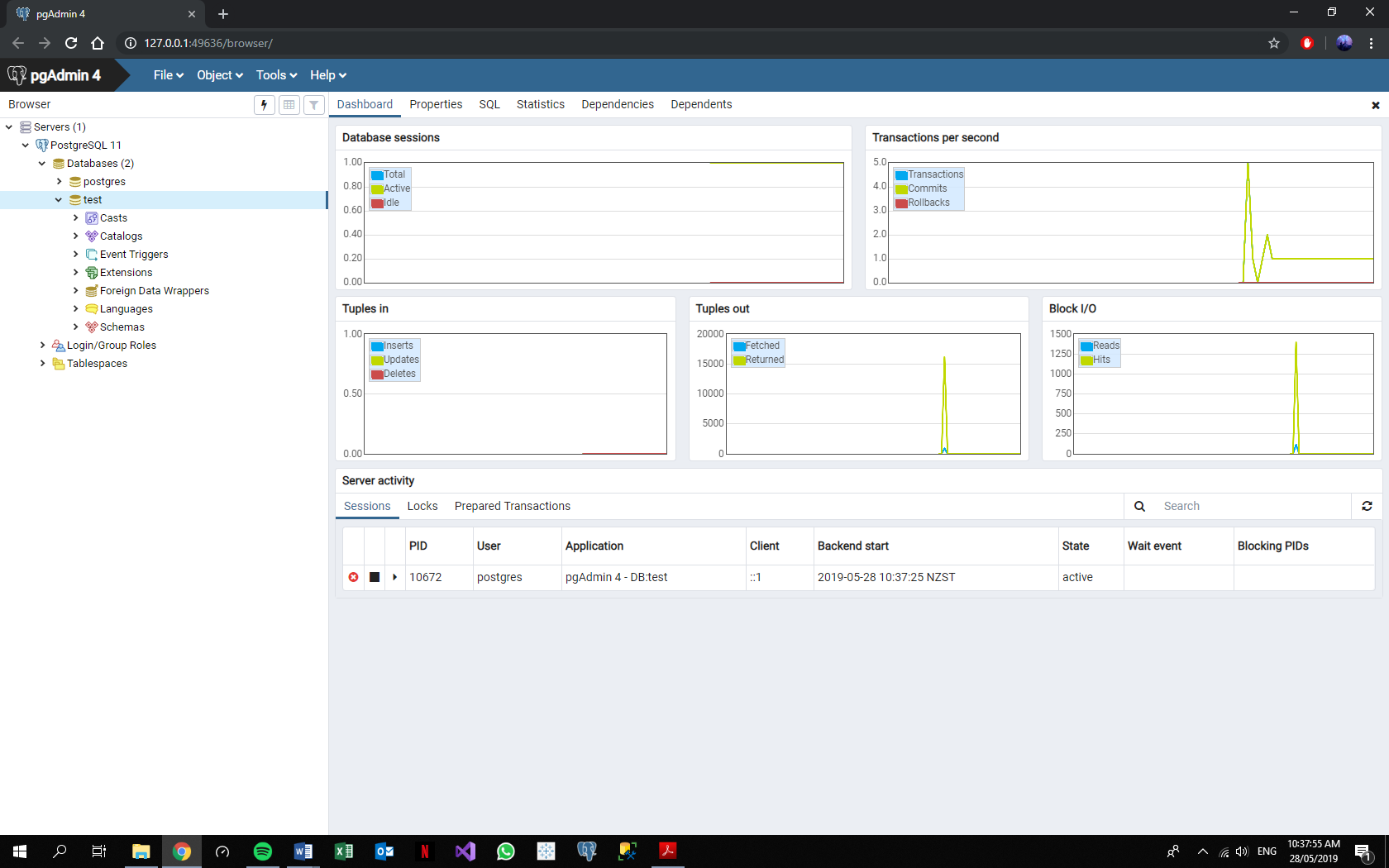


Figure 12: Postgres pgAdmin4 Server Interface

### Create Tpcc Database

Right clicking on “Databases”, hovering over “Create”, then selecting “Database…” allows the user to create the database “tpcc”. Again, this must be named as such for the TPC-C application to recognise it.

### Populate Tpcc

Refer to common Populate “Tpcc” Database section.

### Database Backup

By right clicking on the database “tpcc” and selecting “Backup…” as seen in Figure 13, a backup copy of the database is created, in the pop-up form seen in Figure 14, under an appropriate name such as “tpcc copy 2”. After clicking on “Backup” within the form, Postgres will store this backup file in a default location such as the “Documents” folder on the computer. Once this is completed, a tab appears on the bottom right of the screen, as seen in Figure 13, to confirm the backup has completed. This may be different for other versions of Windows or other operating systems that Postgres is compatible with.

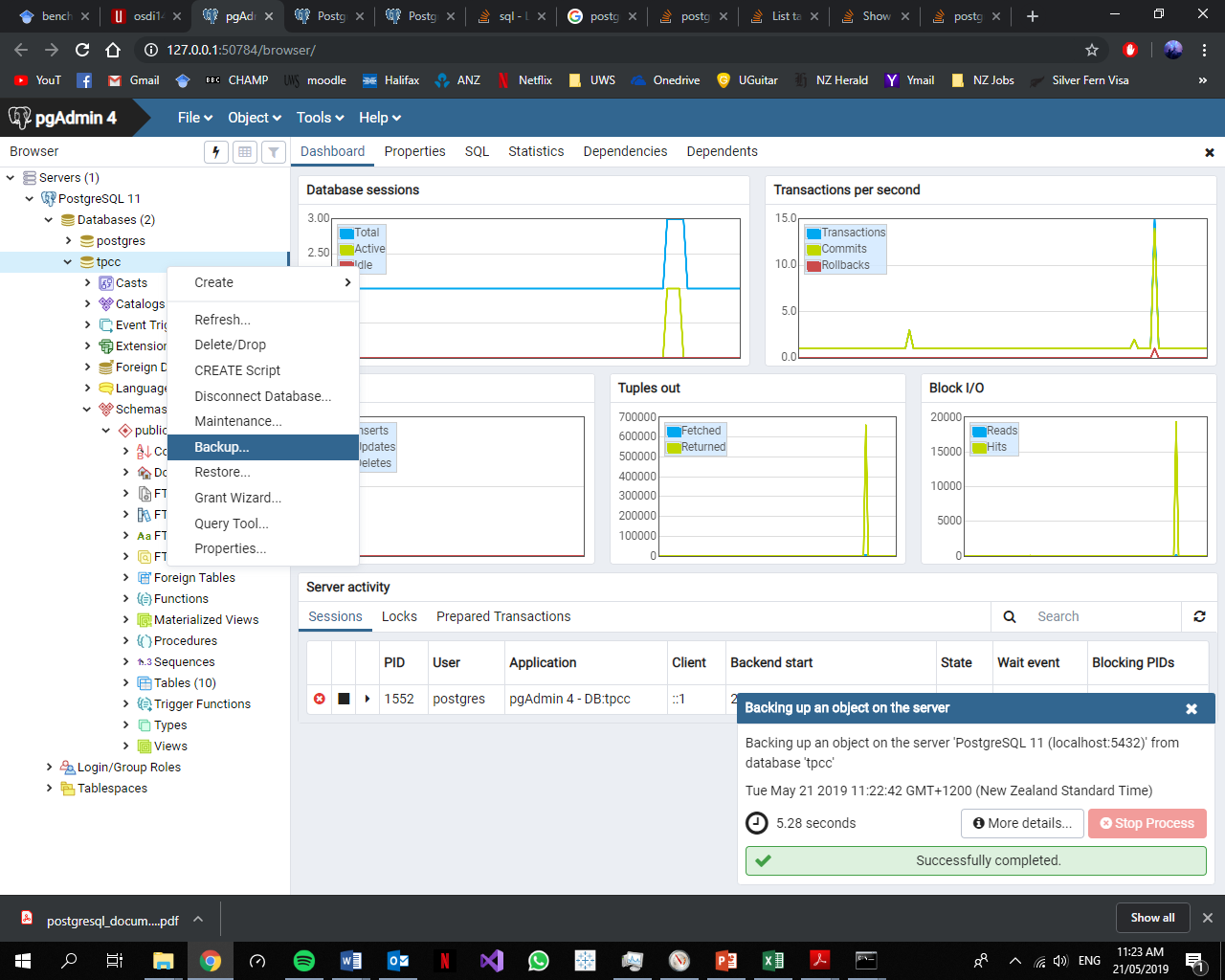


Figure 13: Postgres Backup Step 1

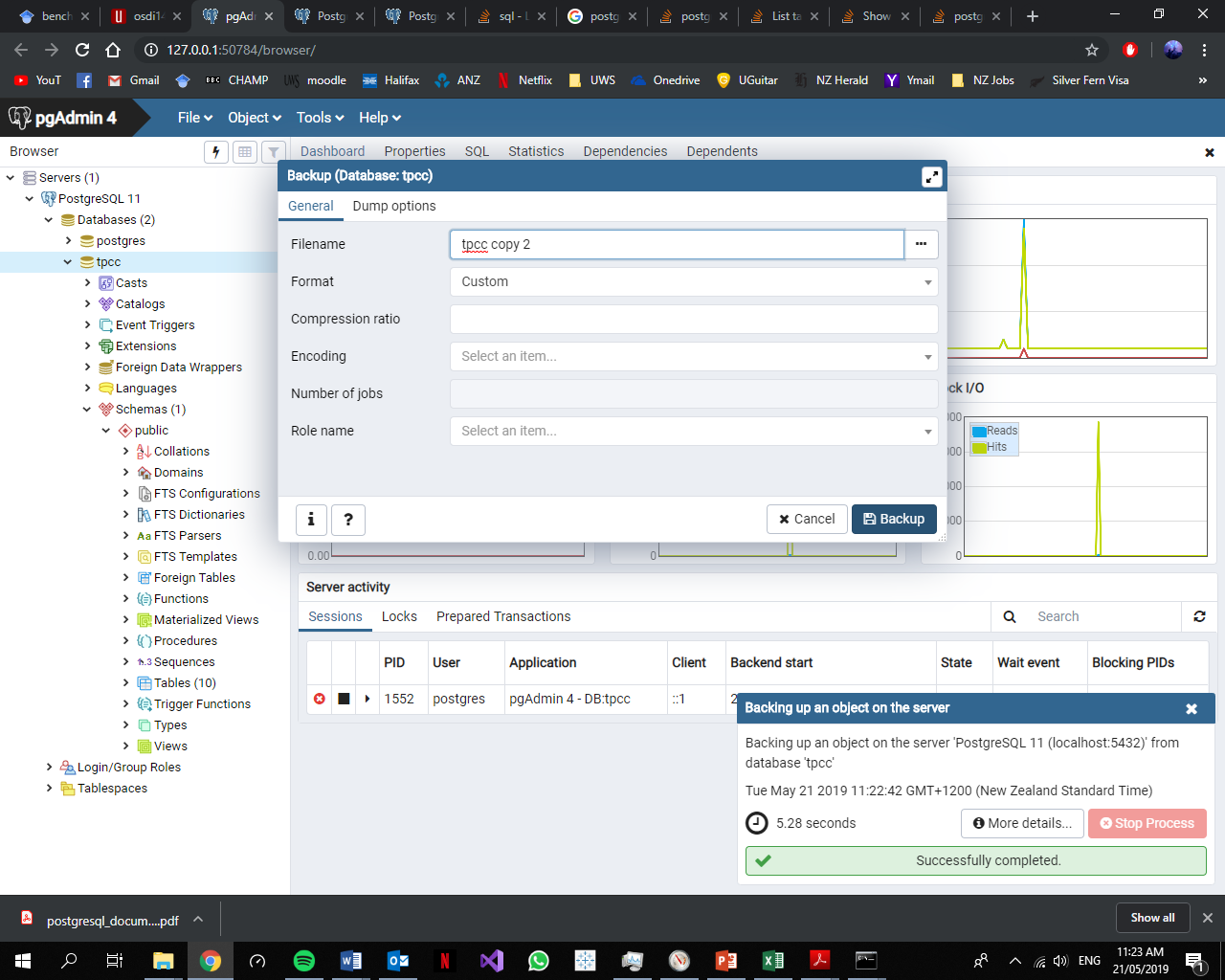


Figure 14: Postgres Backup Step 2

### Query for Table Condition

The original state of the table must be known before testing begins. A Windows cmd window is used as a client to connect to the Postgres server. As seen in Figure 15, this is done by executing the “psql.exe” file, found in the PostgreSQL\bin path created on installation, and writing in the command prompt “-U postgres tpcc”. “-U” tells Postgres that a username is about to follow, “postgres” is the default username created on installation, and “tpcc” is the name of the database to be connected to. To query the initial database state the following query is entered:

select relname, n\_live\_tup from pg\_stat\_user\_tables;

Copying the table is done in the same manner as a Strong query result. This query is only accurate if the TPC-C application has properly terminated. Executing this query during a benchmark will give inaccurate results as it searches for the estimated number of ‘live’ rows (PostgreSQL Global Development Group, 2019). A row will be considered ‘live’ by the DBMS if it is not locked in a transaction.

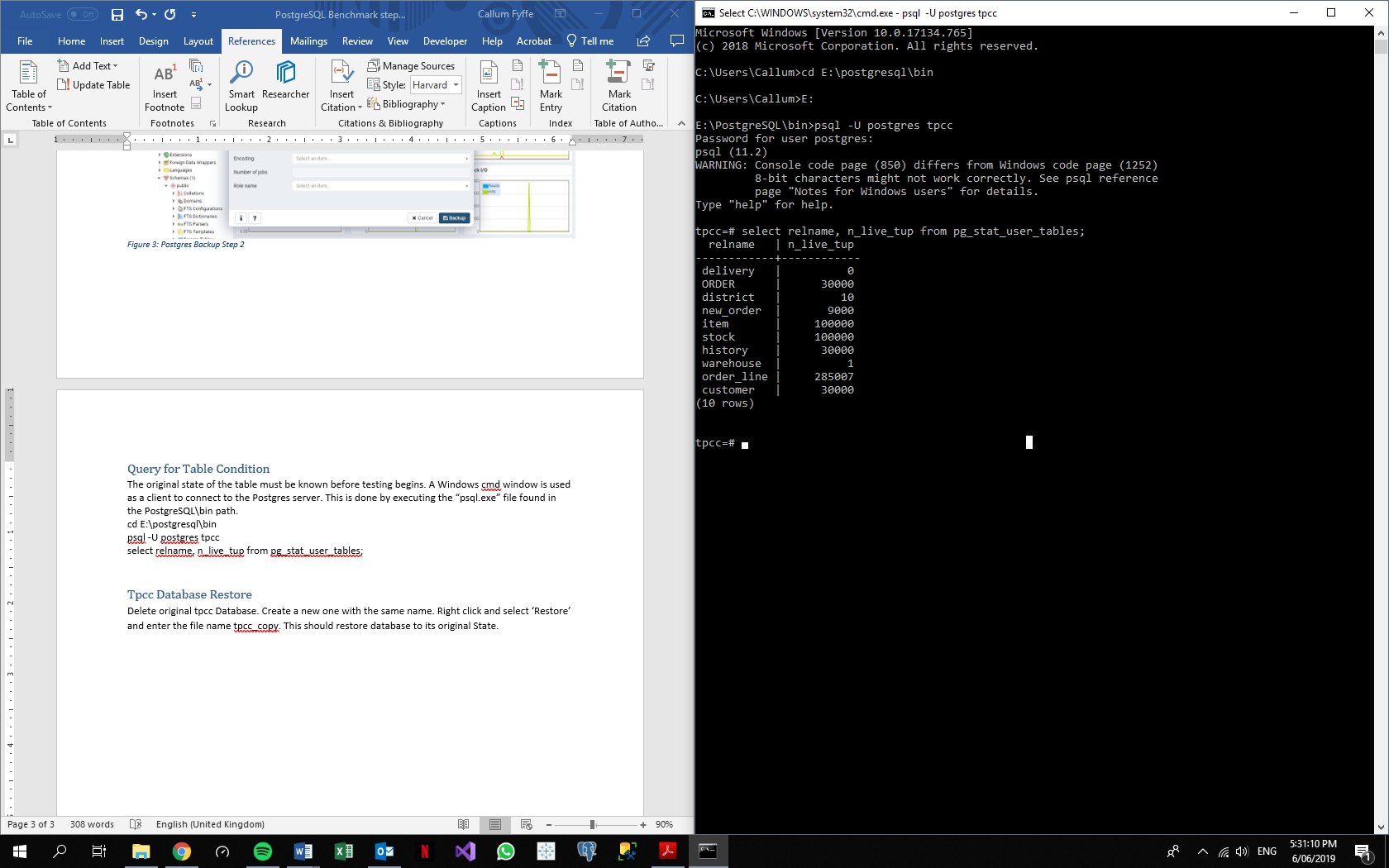


Figure 15: Cmd Prompt Postgres Client Window

### Run TPC-C Application

Refer to common Run TPCC section.

### Result Collection

The query displayed in Figure 15 is run to display the tables’ properties then copied into a word document to be analysed later.

### Restore Database

The original database “tpcc” is deleted. A new one is created with the same name. Right clicking on the database “tpcc”, as seen in Figure 16, and selecting “Restore…” opens a pop-up form. In this form, the file name chosen in the earlier Backup section is entered. This will restore database to its original state.

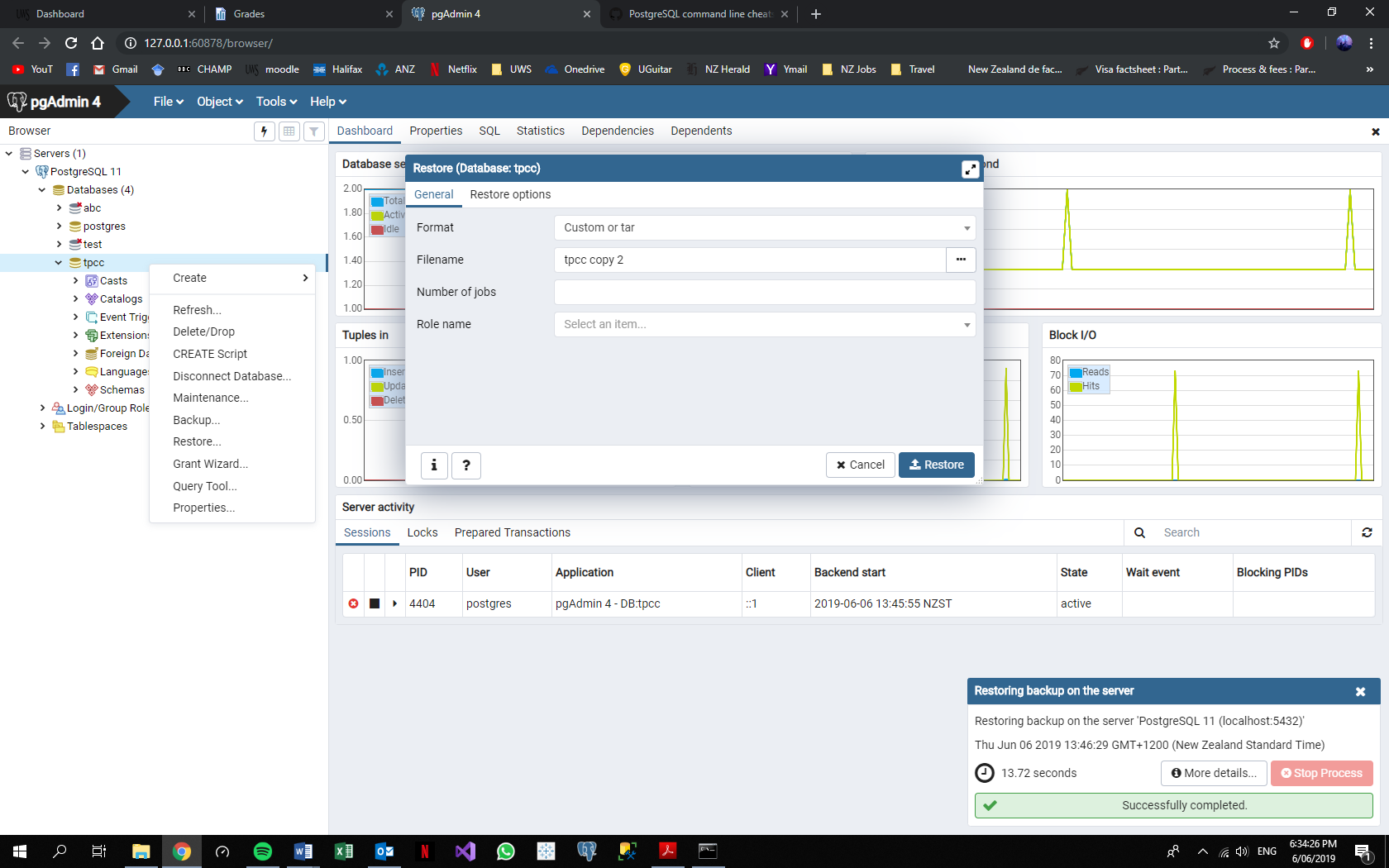


Figure 16: Postgres Database Restoration

## Additional Considerations for Methodology

Research literature has been and will be further acquired through the UWS library, Google Scholar, IEEE, SpringerLink and official documentation of the RDBMSs from their respective websites. Research literature will be revised throughout to uncover any developments in the related fields such as version updates to the RDBMSs that could affect the outcome of TPC-C benchmark tests.

The results of these tests will be compared to provide an evaluation on how far StrongDBMS has come and what future work and research can be conducted to improve it. Depending on the results, further TPC benchmark tests could be suggested, such as TPC-E, which features 12 transaction types instead of 5 (Transaction Processing Performance Council, 2019). It would also be interesting to test the DBMSs against attacks such as ACIDRain (Warszawski & Bailis, 2017) which specifically exploits concurrency-based weaknesses. Based on how StrongDBMS is designed, it should perform well against this kind of attack.

StrongDBMS is free to use by anyone (Crowe, 2019b) and the project will be in collaboration with the efforts of Malcolm Crowe, Santiago Matalonga and UWS.

Before conducting any benchmarking tests on the RDBMSs, the terms and conditions of each licence must be considered.

* As per Strong’s documentation (Crowe, 2019a):

“…available for use by anyone and in any product without fee, provided only that its origin and original authorship is suitably acknowledged.”

* As per Oracle’s License and Service agreements that can be found on their website (Oracle, 2018):

“You may not: … - disclose results of any program benchmark tests without our prior consent.”

* As per Microsoft’s Software License Terms (Microsoft, 2019):

“Customer must obtain Microsoft’s prior written approval to disclose to a third party the results of any benchmark test of any Server Product.”

* As per PostgreSQL’s documentation (PostgreSQL Global Development Group, 2019) there is no mention against disclosing results of benchmark testing.

Due to Oracle and SQL Server having strict policies on disclosing benchmark test, one will be tested but its identity will remain anonymous.

# Results thus far

## Strong (Average of 3 Benchmark Tests)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Clerks | 0 | 1 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 | 110 | 120 | 130 |
| AVE\_Commits | 0 | 16 | 136 | 209 | 259 | 289 | 324 | 329 | 351 | 359 | 361 | 381 | 375 | 385 | 388 |
| AVE\_Conflict L | 0 | 0 | 6 | 14 | 27 | 33 | 31 | 44 | 43 | 55 | 60 | 66 | 70 | 74 | 69 |
| AVE\_Conflict R | 0 | 0 | 103 | 395 | 749 | 1141 | 1529 | 1957 | 2397 | 2796 | 3175 | 3518 | 3697 | 4055 | 4505 |
| AVE\_Exceptions | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| AVE\_ORDER | 0 | 16 | 133 | 203 | 246 | 271 | 304 | 305 | 322 | 323 | 329 | 337 | 334 | 339 | 339 |
| AVE\_NEW\_ORDER | 0 | 16 | 133 | 203 | 246 | 271 | 304 | 305 | 322 | 323 | 329 | 337 | 334 | 339 | 339 |
| AVE\_ORDER\_LINE | 0 | 151 | 1157 | 1700 | 1958 | 2055 | 2267 | 2129 | 2199 | 2177 | 2177 | 2249 | 2194 | 2182 | 2160 |
| AVE\_DELIVERY | 0 | 1 | 14 | 19 | 30 | 30 | 35 | 30 | 40 | 43 | 47 | 41 | 39 | 43 | 44 |
| Clerk Efficiency (%) |  | 100 | 83.1 | 63.4 | 51.3 | 42.3 | 38 | 31.8 | 28.8 | 25.2 | 22.8 | 21.1 | 19 | 17.7 | 16.3 |

## SQL Server (2 Separate Tests)

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Clerks | 0 | 1 | 2 | 4 | 6 | 8 | 10 | 12 | 14 | 20 | 30 |
| A\_Commits | 0 | 16 | 35 | 46 | 72 | 71 | 89 | 92 | 61 | 71 | 31 |
| A\_Conflict L | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| A\_Conflict R | 0 | 0 | 0 | 21 | 6 | 33 | 32 | 47 | 235 | 181 | 1344 |
| A\_Exceptions | 0 |  |  |  |  |  |  |  |  |  |  |
| A\_ORDER | 30000 | 30016 | 30036 | 30067 | 30073 | 30101 | 30112 | 30119 | 30112 | 30144 | 30034 |
| A\_NEW\_ORDER | 9000 | 9016 | 9036 | 9067 | 9073 | 9101 | 9112 | 9119 | 9112 | 9144 | 9034 |
| A\_ORDER\_LINE | 285007 | 285158 | 285349 | 285644 | 285701 | 285979 | 286073 | 286141 | 286113 | 286336 | 285330 |
| A\_DELIVERY | 0 | 1 | 3 | 3 | 6 | 11 | 9 | 18 | 10 | 16 | 10 |
|  |  |  |  |  |  |  |  |  |  |  |  |
| Clerks | **0** | **1** | **2** | **4** | **6** | **8** | **10** | **12** | **14** | **20** |  |
| B\_Commits | 0 | 16 | 35 | 46 | 72 | 59 | 70 | 87 | 63 | 12 |  |
| B\_Conflict L | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |  |
| B\_Conflict R | 0 | 0 | 0 | 21 | 11 | 50 | 41 | 98 | 126 | 1192 |  |
| B\_Exceptions | 0 |  |  |  |  |  |  |  |  |  |  |
| B\_ORDER | 30000 | 30016 | 30036 | 30069 | 30082 | 30103 | 30095 | 30159 | 30114 | 30012 |  |
| B\_NEW\_ORDER | 9000 | 9016 | 9036 | 9069 | 9082 | 9103 | 9095 | 9159 | 9114 | 9012 |  |
| B\_ORDER\_LINE | 285007 | 285158 | 285349 | 285680 | 285743 | 286041 | 285907 | 286502 | 286173 | 285120 |  |
| B\_DELIVERY | 0 | 1 | 3 | 4 | 13 | 12 | 15 | 17 | 14 | 3 |  |

## PostgreSQL

Postgres Results will be done after source code for TPC-C is updated. Currently max performance gets about 18 New Orders at 6-8 clerks which is only an improvement of +2 from 1 clerk.