



PostgreSQL OLTP (Transactional) Load Testing

This guide gives you an introduction to conducting OLTP (Online Transaction Processing) workloads on the PostgreSQL Database. This guide will equip you with the essentials for assessing the ability of any system that runs the PostgreSQL Database for processing transactional workloads. On completion of this guide you will be able to run detailed and comprehensive PostgreSQL load tests. After building a basic skill set, you should be able to take a system from 'bare metal' to generation of a full performance profile within one day. If you have not already done so you should read the [Quick Start tutorial](#) before proceeding with this guide.

EnterpriseDB Postgres Plus Advanced Server and PostgreSQL

[EnterpriseDB](#) produce an enhanced version of the PostgreSQL database termed Postgres Plus Advanced Server. This version of PostgreSQL in particular includes compatibility features with the Oracle database such as PL/SQL support. For this reason the Hammerora OLTP workload for PostgreSQL can operate in 2 modes. Firstly Oracle compatible mode uses PL/SQL and additional Postgres Plus Advanced Server features (such as DRITA snapshots) that will only operate against Enterprise DB Postgres Plus Advanced Server. Secondly by not selecting Oracle compatibility Hammerora can continue to operate against Postgres Plus Advanced Server but additionally against a regular PostgreSQL build using native PostgreSQL features. You must ensure before proceeding with OLTP that you are aware of the version of PostgreSQL you have installed and the features available, if you wish to test Oracle compatibility then you must use Postgres Plus Advanced Server from EnterpriseDB and install in Oracle compatible mode.

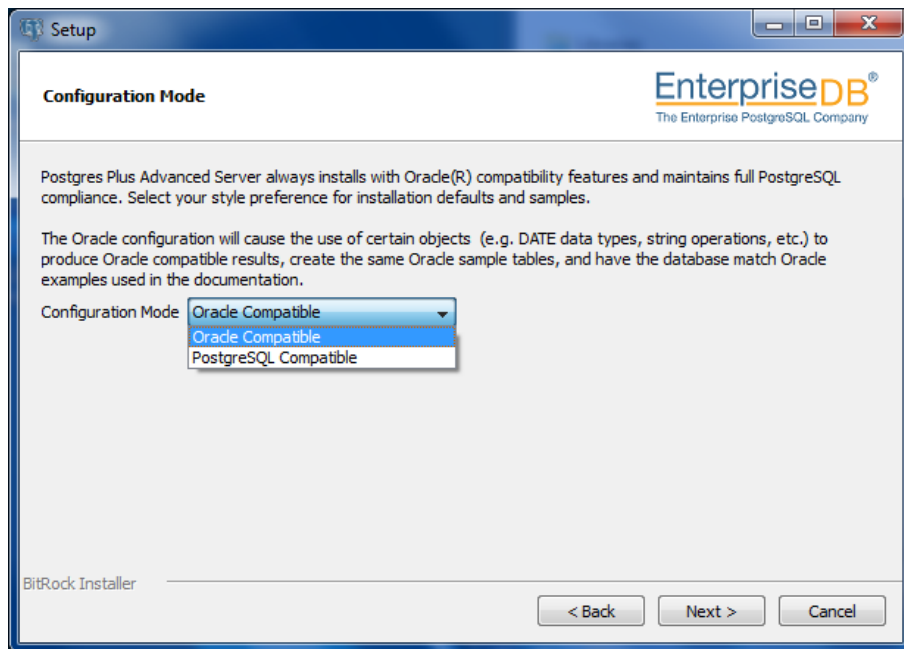


Figure 1 Enterprise DB Oracle Compatible

Database load testing is an advanced skill and therefore familiarity with the PostgreSQL Database and basic PostgreSQL DBA skills are assumed. You should already be able to create, administer and connect to an PostgreSQL database. If you do not have these skills I recommend start with an [Introduction to PostgreSQL](#).

EnterpriseDB Postgres Plus Advanced Server and PostgreSQL	1
Introduction.....	3
Single Threaded Tests.....	3
What is TPC-C?	3
Performance Profiles.....	5
Test Network Configuration.....	6
Load Generation Server Configuration	7
SUT Database Server Configuration	8
Administrator PC Configuration	8
Installation and Configuration	8
Load Generation Server Installation	9
Load Generation Server Configuration	9
SUT Database Server Installation	9
Network Connectivity	9
Creating the Test Schema.....	10
Build Options.....	12
Starting the Schema Build	13
Pre-Testing and Planning	18
Driver Options.....	20
Loading the Driver Script.....	24
Pre-Test 1 Verifying the Schema	25
Pre-Test 2 Single and Multiple Virtual User Throughput	32
Planning and Preparation.....	39
Running Timed Tests with the Timed Test Driver Script.....	40
Automating Tests with Autopilot Mode.....	47

Performance Analysis	52
Conclusion	53
Support and Discussion.....	54

Introduction

In this introduction we give an an overview of the correct approach to take for PostgreSQL load testing and discuss the tests that Hammerora implements.

Single Threaded Tests

Historically user database performance tests were often conducted with simple scripts and using the time to completion for a simple select statement, function or Cartesian join as a prediction of CPU performance , for example:

```
edb=# select count(*) from information_schema.parameters cross join
information_schema.columns cross join information_schema.sql_languages;

count
-----
65732472

(1 row)

Time: 6761.382 ms
```

The challenge was that such a statement runs as single-threaded and therefore was valid in the era of single-core processors to test the performance of one processor but became obsolete in the era of multi-core processors. In other words with an example eight core processor such a test would give indications on the performance of one core and leave the other cores idle testing only a fraction of the performance potential of the CPU as a whole. Additionally such tests focused on CPU performance only without testing any of the storage component. Such a simple approach is flawed and to test a multiple CPU or multicore database environment requires a multithreaded test framework. Fortunately Hammerora is multi-threaded and therefore ready to test your multi-core environments with multiple virtual users all interacting independently with the database simultaneously.

What is TPC-C?

Designing and implementing a benchmark is a significant challenge. Many performance tests and tools experience difficulties in comparing system performance especially in the area of scalability, the ability of a test conducted on a certain system and schema size to be comparable with a test on a larger scale system. When system vendors wish to publish benchmark information about PostgreSQL performance they have long had to access to such sophisticated test specifications to do so. In particular it can be noted that PostgreSQL [recognise](#) the [TPC-C](#) as the standard for Online Transaction Processing, the type of workload we are looking to simulate. Fortunately the [“TPC benchmarks are industry standards. The TPC, at no charge, distributes its benchmark specifications to the public.”](#) For this reason Hammerora includes an implementation of the specification of the TPC-C benchmark that can be run in any PostgreSQL environment. This implementation has the significant advantage that you know that the test is reliable, scalable and tested to produce consistent results. It is important to emphasise that the implementation is

not a full specification TPC-C benchmark and the transaction results cannot be compared with the official published benchmarks in any way. Instead the implementations in Hammerora take the best designed specifications for a database transactional workload available in the world and enable you to run an accurate and repeatable workload against your own PostgreSQL database. Audited TPC-C benchmarks are extremely costly and time consuming to establish and maintain, the Hammerora implementation of the TPC-C benchmarks is designed to capture the essence of TPC-C in a form that can be run at low cost bringing professional load testing to all PostgreSQL environments.

TPC-C implements a computer system to fulfil orders from customers to supply products from a company. The company sells 100,000 items and keeps its stock in warehouses. Each warehouse has 10 sales districts and each district serves 3000 customers. The customers call the company whose operators take the order, each order containing a number of items. Orders are usually satisfied from the local warehouse however a small number of items are not in stock at a particular point in time and are supplied by an alternative warehouse. Figure 2 shows this company structure.

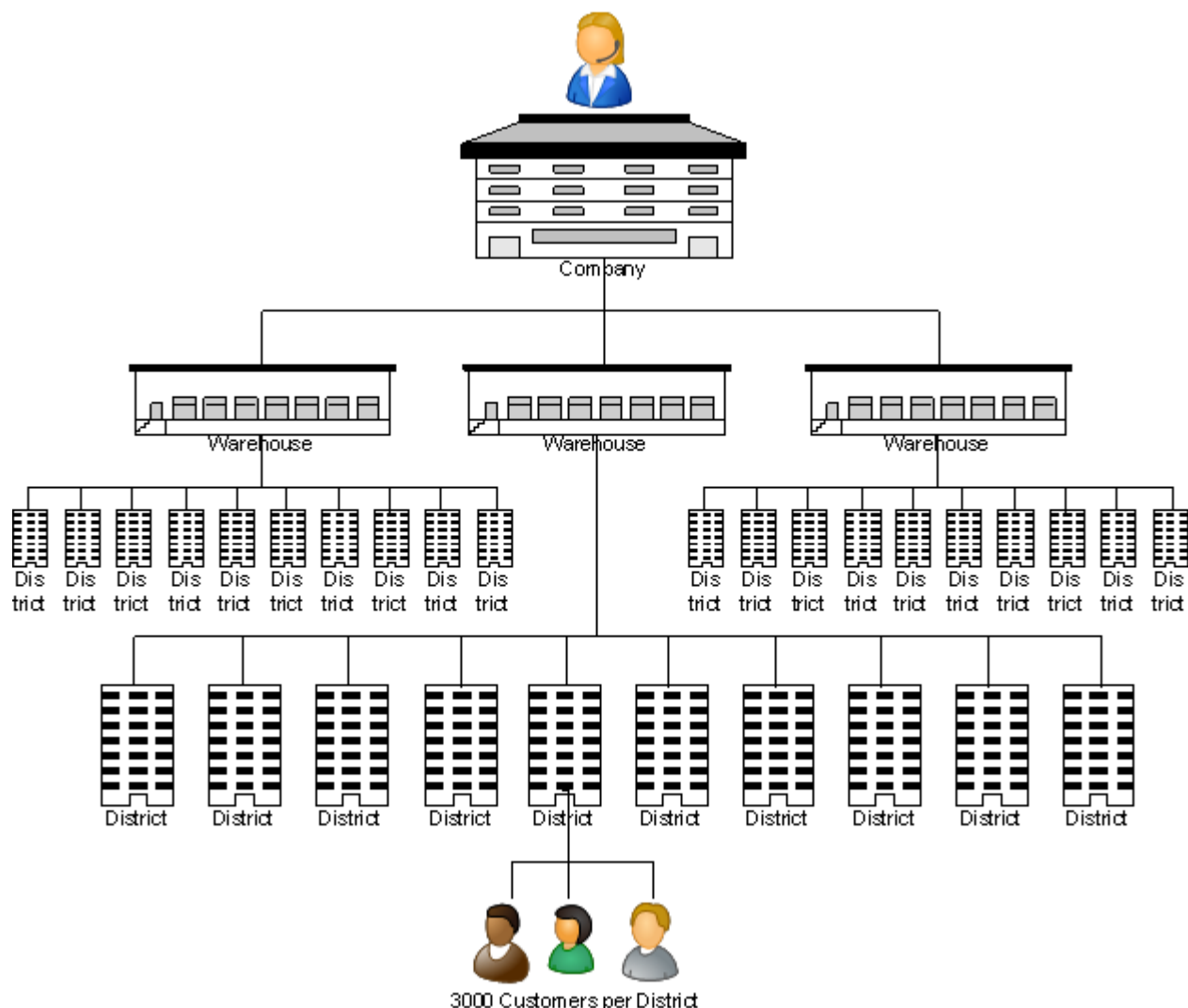


Figure 2 TPC-C Company Structure

It is important to note that the size of the company is not fixed and can add Warehouses and sales districts as the company grows. For this reason your test schema can be as small or large as you wish with a larger schema requiring a more powerful computer system to process the increased level of transactions. Figure 3 shows the TPC-C schema, in particular note how the number of rows in all of the tables apart from the ITEM table which is fixed is dependent upon the number of warehouses you choose to create your schema.

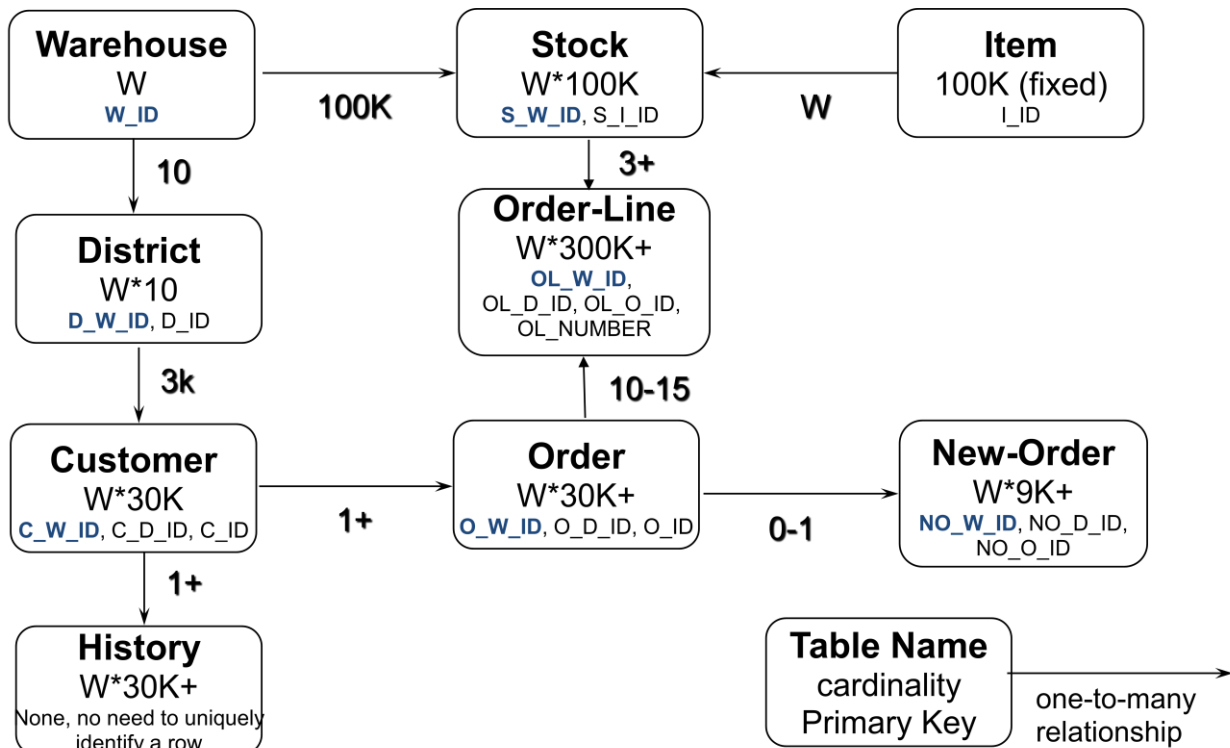


Figure 3 TPC-C Schema

For additional clarity please note that the term Warehouse in the context of TPC-C bears no relation to a Data Warehousing workload, TPC-C defines a transactional based system and not a decision support (DSS) one.

In addition to the computer system being used to place orders it also enables payment and delivery of orders and the ability to query the stock levels of warehouses. Consequently the workload is defined by a mix of 5 transactions as follows:

- New-order: receive a new order from a customer: 45%
- Payment: update the customers balance to record a payment: 43%
- Delivery: deliver orders asynchronously: 4%
- Order-status: retrieve the status of customer's most recent order: 4%
- Stock-level: return the status of the warehouse's inventory: 4%

Performance Profiles

For an official audited TPC-C benchmark the result of the tests is detailed as tpmC which represents the number of New Orders processed only. One particular advantage of Hammerora is the ability to generate a performance profile as the load increases on your system. Whereas an official TPC-C benchmark gives you a single data-point and a typical single-threaded test (such as timing SQL Statements) also gives you a single data-point. Consider the graph shown in figure 4.

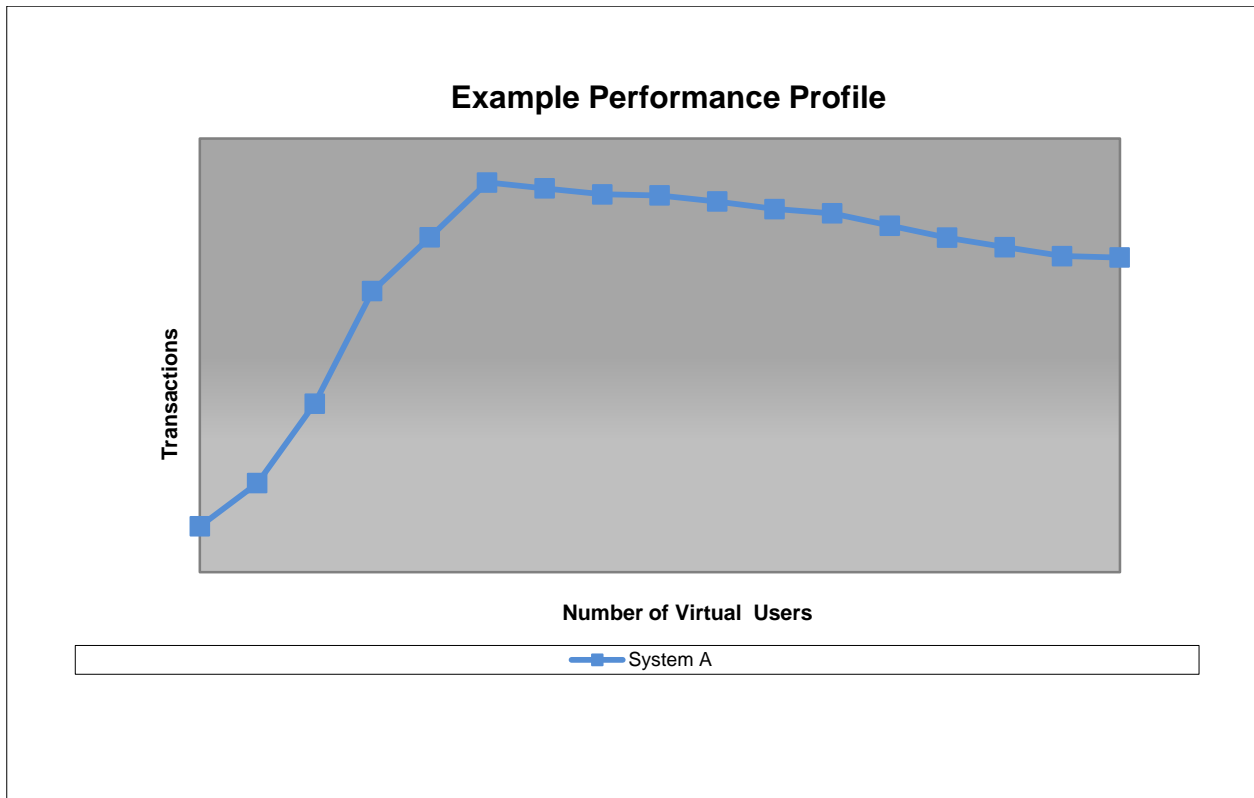


Figure 4 Performance Profile Example

This graph shows the performance of real tests on a PostgreSQL configuration. System A shows good scalability up to a certain level of utilisation, however beyond this point throughput declines as utilisation is increased. It should be clear that your testing goal should be to measure the performance profile of your system across all levels of utilisation to assess whether performance is maintained to the match available system resources.

Test Network Configuration

As shown in figure 5 the network architecture required for Hammerora is both simple and straightforward. You require the database server to be tested known as the system under test (SUT) installed and configured with the PostgreSQL database. You also require a load generation server to run Hammerora installed with the Hammerora software and an PostgreSQL client. Typically the administrator will monitor and manage the load testing from a separate notebook or PC. All systems will be connected across a network. Technically it is possible to run Hammerora on the SUT however this is not recommended. Firstly it makes it comparatively harder to distinguish between the load generation server workload and the database workload. Secondly running the Hammerora workload will skew the results. By eliminating the network component of the workload results for a smaller number of virtual users will be comparatively higher however as the workload increases performance will be comparatively lower. To eliminate this skew in results a dedicated load generation server should be used.

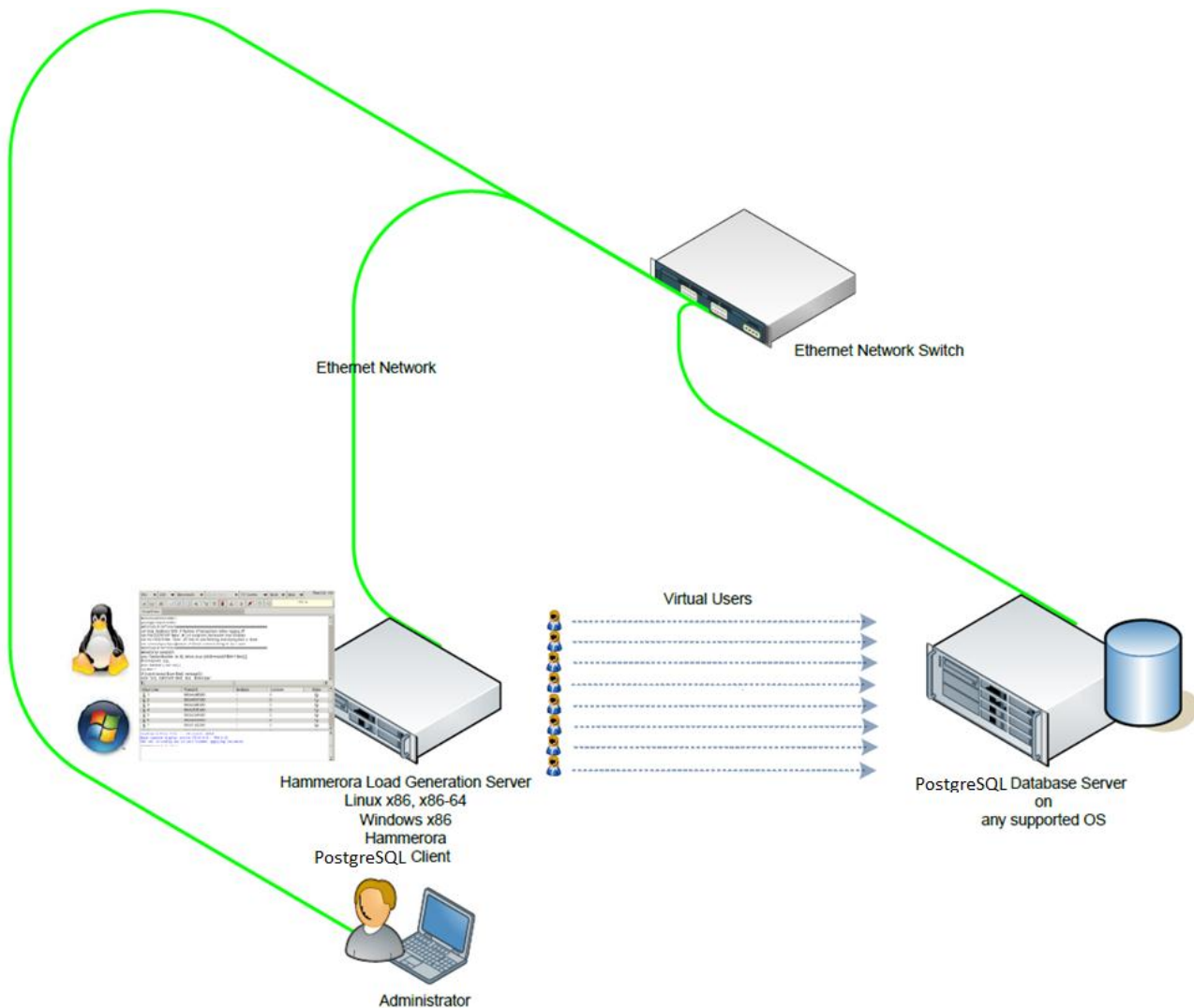


Figure 5 Hammerora Network Architecture

Load Generation Server Configuration

The most important component of the load generation server is the server processor. It is recommended to use an up to date multicore and multithreaded processor. Hammerora is a multithreaded application and implicitly benefits from a multicore server CPU such as the Intel Xeon E5 series range. To determine whether CPU capacity is sufficient for testing you can monitor the CPU utilisation with utilities such as **top** on Linux or **Task Manager** on Windows during testing. CPU utilisation reaching 100% is an indication that the CPU on the load generation server is limiting performance. It is important to note however that Hammerora is highly efficient and a high-performance multicore processor such as the Xeon E5 and upwards will likely only see utilisation in the 10% range to drive a much larger database server to 100% utilisation. For the load generation memory requirement a rough guide is that Hammerora requires approximately 10MB for the application and 2MB per virtual user, for example 64 virtual users will need 138MB of memory. Again this represents a highly efficient load testing environment in comparison to commercial database load testing applications. Consequently it is entirely feasible to load test with a 32-bit x86 operating system on the load generation client with a 64-bit operating system only required when conducting tests in excess of 1000 virtual users. For the load testing operating system, Hammerora is available pre-compiled for 32-bit Windows, 32-bit Linux and 64-bit Linux however you may compile the packages used for Hammerora manually for another operating system if you wish. Hammerora is a graphical application and therefore on Linux the operating system installation must include the X-windows packages. Storage requirements on the load generation server are minimal and all modern servers are likely to meet

the storage required. Hammerora consumes approximately 15MB of disk space and you will also need an PostgreSQL client. All PostgreSQL database installations include an PostgreSQL client. Postgres Plus Advanced Server installations include a “client” directory. If the load generation server is running under the same operating system as the SUT this client directory can be copied from SUT to load generation server as the quickest way to provide the client libraries necessary for Hammerora to connect to the SUT.

SUT Database Server Configuration

The database server architecture to be tested must meet the standard requirements for an PostgreSQL Database Server. PostgreSQL can be installed on any supported operating system, there is no restriction on the version of PostgreSQL that is required. To run a Hammerora transactional load test there are minimum requirements in memory and I/O (disk performance) to prevent these components being a bottleneck on performance. In turn the memory and I/O is determined by the capabilities of the CPUs installed. For a test schema that requires the minimal level of memory and I/O and therefore keying and thinking time is set to FALSE (keying and thinking time is detailed later in this guide) you should aim to create a schema with approximately a factor of 5 warehouses multiplied by the number of cores/threads (threads refers to Hyper Threads, so if a processor has 4 cores but supports 8 Hyper Threads then the value of 8 should be used) supported by system processor rounded up to the nearest 100. For example for a system with 64 threads, $64 * 5 = 320$, rounded up to 400. A system with 24 threads would be configured with 200 warehouses. To calculate the disk storage required by your test schema allow a minimum of 100MB per warehouse plus 50% for additional growth. For the example system that supports 64 threads that would be 60GB of storage space. You should have sufficient memory to cache as much of your test schema in memory with the `shared_buffers` parameter although care should be taken to ensure that this value is not oversized beyond Postgres recommendations. Reductions in memory will place more emphasis on the I/O performance. The database server should be connected to external storage typically in the form of a SAN or NAS. A database server without external storage will not sustain the level of I/O throughput necessary to conduct a load test. I recommend SSD disks to minimise the risk of I/O constraining performance and a RAID configuration of 8 SSD disks is a good minimum configuration. For hard disks a minimum configuration of 12 disk drives is recommended. Where possible RAID 0 should be used to maximise the potential performance of the disk drives available. If your I/O configuration does not meet minimum requirements it is worthwhile taking a moment to question whether to proceed with a performance test that will result in a conclusion that the storage is inadequate. For transactional tests I/O should be configured in particular to maximize sequential write performance to the WAL files. There is less emphasis on the storage performance of the datafiles provided that enough memory is available to cache the test schema and database checkpoints are not frequent during the test. It is worth reiterating that these guidelines define the minimum requirements for testing and you should aim for your SUT to exceed these minimum standards where possible. In particular where you are using keying and thinking times the number of virtual users required and the memory and I/O requirements will increase considerably.

Administrator PC Configuration

The administrator PC has the minimal requirement to display the graphical output from the load generation server. for example for a Linux load generation server the ability to display X windows. The PC should also have the ability to connect to the SUT.

Installation and Configuration

This sections describes the procedure to install and configure the Load Generation Server and the SUT Database Server.

Load Generation Server Installation

On the Load Generation Server refer to the dedicated [Hammerora Installation Guide](#) for details on installing Hammerora for your environment.

Load Generation Server Configuration

All of Hammerora's working data can be set using menu options. However if you wish in the Hammerora home directory there is a configuration file called config.xml that is read on startup. In this file you can preset your schema build and driver configurations by editing the xml file without having to change the data manually. If your xml file is well formed your variables will be applied to Hammerora when you selected the menu options.

```
= <postgresql>
= <connection>
  <pg_host>localhost</pg_host>
  <pg_port>5432</pg_port>
</connection>
= <tpcc>
= <schema>
  <pg_count_ware>1</pg_count_ware>
  <pg_num_threads>1</pg_num_threads>
  <pg_superuser>postgres</pg_superuser>
  <pg_superuserpass>postgres</pg_superuserpass>
  <pg_defaultdbase>postgres</pg_defaultdbase>
  <pg_user>tpcc</pg_user>
  <pg_pass>tpcc</pg_pass>
  <pg_dbase>tpcc</pg_dbase>
  <pg_vacuum>false</pg_vacuum>
  <pg_dritasnap>false</pg_dritasnap>
  <pg_oracompat>false</pg_oracompat>
</schema>
= <driver>
  <pg_total_iterations>1000000</pg_total_iterations>
  <pg_raiseerror>false</pg_raiseerror>
  <pg_keyandthink>false</pg_keyandthink>
  <pg_driver>standard</pg_driver>
  <pg_rampup>2</pg_rampup>
  <pg_duration>5</pg_duration>
</driver>
</tpcc>
</postgresql>
```

SUT Database Server Installation

Installation and configuration of the PostgreSQL Database on your chosen operating system is beyond the scope of this document. We recommend using one of the many tutorials available on the web. You should have the PostgreSQL database software installed and a test database created and running. During the installation make a note of your postgres superuser password, you will need it for the test schema creation.

Network Connectivity

You must be able to connect from your load generation server to your SUT database server across the network. This will involve successful configuration of your pg_hba.conf on the SUT database server. For example the following extract from a pg_hba.conf file from a PostgreSQL 9.1.3 installation shows trusted

local connections on the SUT permitting connection without a password and remote connections from the Load Generation server with IP address 172.17.1.40 with the remote connections showing multiple options of either entering specific usernames or the 'all' keyword.

```
# TYPE    DATABASE      USER          ADDRESS          METHOD
# "local" is for Unix domain socket connections only
local     all           all           trust
# IPv4 local connections:
hostnossl postgres postgres 172.17.1.40/32 md5
hostnossl tpcc      tpcc          172.17.1.40/32 md5
hostnossl all        all           172.17.1.40/32 md5
```

Note that the syntax of `pg_hba.conf` has changed for different versions of PostgreSQL and you should therefore consult the PostgreSQL documentation to troubleshoot connectivity issues.

Creating the Test Schema

To create the OLTP test schema based on the TPC-C specification you will need to select which benchmark and database you wish to use by choosing select benchmark from under the Options menu or under the benchmark treeview. The initial settings are determined by the values in your `config.xml` file. If the benchmark treeview shows another database then double-click on PostgreSQL to show the option to switch to PostgreSQL and TPC-C.

Select PostgreSQL and TPC-C and press OK as shown in Figure 6.

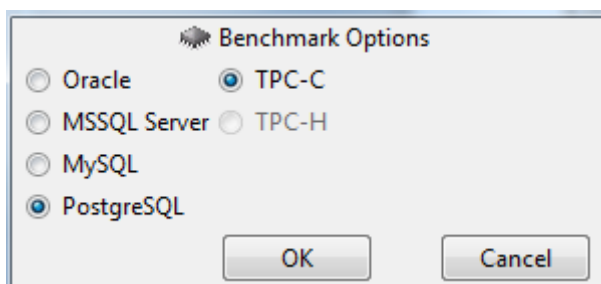


Figure 6 Select Benchmark

To create the TPC-C schema select the TPC-C schema options menu tab from the benchmark treeview or the options menu. This menu will change dynamically according to your chosen database.

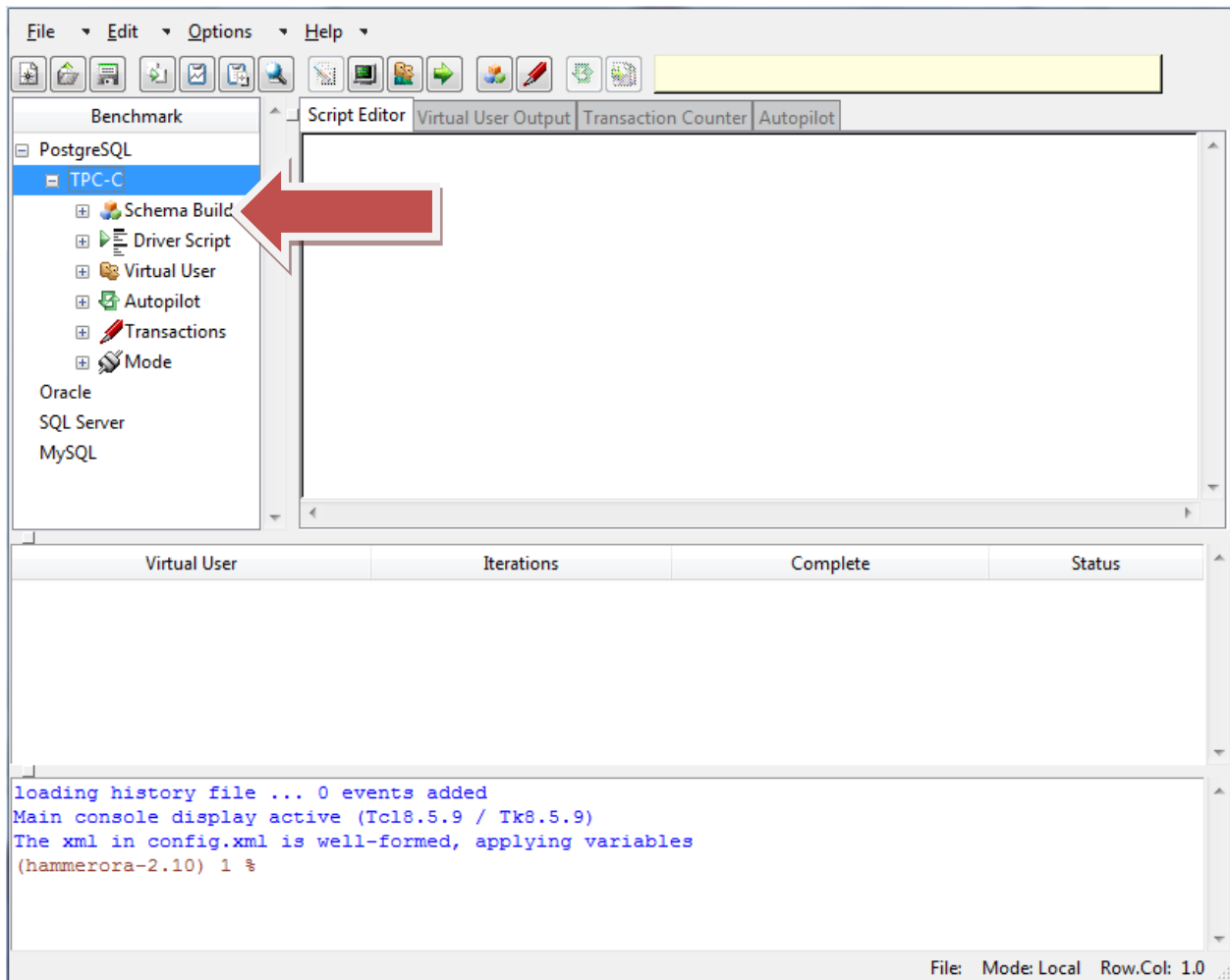
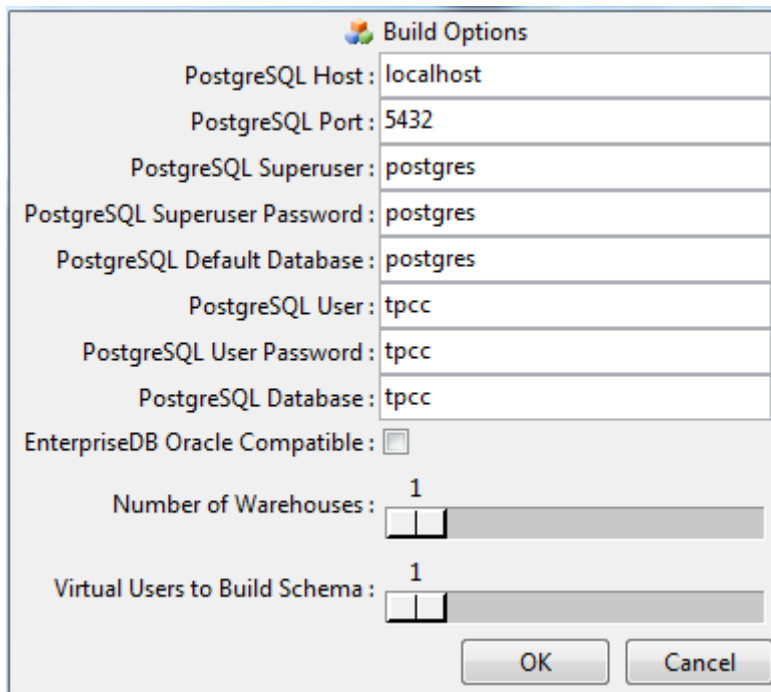


Figure 7 Select Schema Options

If selected from the Options menu the schema options window is divided into two sections. The “Build Options” section details the general login information and where the schema will be built and the “Driver Options” for the Driver Script to run after the schema is built. If selected from the benchmark treeview only the “Build Options” are shown and these are the only options of importance at this stage. Note that in any circumstance you don’t have to rebuild the schema every time you change the “Driver Options” , once the schema has been built only the “Driver Options” may need to be modified. For the “Build Options” fill in the values according to the database where the schema will be built.



The image shows a 'Build Options' dialog box with the following fields and values:

Field	Value
PostgreSQL Host	localhost
PostgreSQL Port	5432
PostgreSQL Superuser	postgres
PostgreSQL Superuser Password	postgres
PostgreSQL Default Database	postgres
PostgreSQL User	tpcc
PostgreSQL User Password	tpcc
PostgreSQL Database	tpcc
EnterpriseDB Oracle Compatible	<input type="checkbox"/>
Number of Warehouses	1
Virtual Users to Build Schema	1

At the bottom right are 'OK' and 'Cancel' buttons.

Figure 8 Build Options

Build Options

The Build Option values have the following meanings.

PostgreSQL Host

The host name of the SUT running PostgreSQL which the load generation server running Hammerora will connect to.

PostgreSQL Port

The port of the PostgreSQL service. By default this will be 5432 for a standard PostgreSQL installation or 5444 for EnterpriseDB.

PostgreSQL Superuser

The PostgreSQL Superuser is a user with sufficient privileges to create both new users (roles) and databases to enable the creation of the test schema.

PostgreSQL Superuser Password

The PostgreSQL Superuser Password is the password for the PostgreSQL superuser which will have been defined during installation. If you have forgotten the password it can be reset from a psql prompt that has logged in from a trusted connection therefore requiring no password.

```
postgres=# alter role postgres password 'postgres';
```

PostgreSQL Default Database

The PostgreSQL default databases is the database to specify for the superuser connection. Typically this will be postgres for a standard PostgreSQL installation or edb for EnterpriseDB.

PostgreSQL User

The PostgreSQL User is the user (role) that will be created that owns the database containing the TPC-C schema.

PostgreSQL User Password

The PostgreSQL User Password is the password that will be specified for the PostgreSQL user when it is created.

PostgreSQL Database

The PostgreSQL Database is the database that will be created and owned by the PostgreSQL User that contains the TPC-C schema.

EnterpriseDB Oracle Compatible

Choosing EnterpriseDB Oracle compatible creates a schema using the Oracle compatible features of EnterpriseDB in an installation of Postgres Plus Advanced Server. This build uses Oracle PL/SQL for the creation of the stored procedures.

Number of Warehouses

The Number of Warehouses is selected by a slider. For fine-tuning you may click either side of the slider to move the value by 1. You should set this value to number of warehouses you have chosen for your test based on the guidance given previously in the section SUT Database Server Configuration.

Virtual Users to Build Schema

The Virtual Users to Build Schema is the number of Virtual Users to be created on the Load Generation Server that will complete your multi-threaded schema build. You should set this value to either the number of warehouses you are going to create (You cannot set the number of threads lower than the number of warehouses value) or the number of cores/Hyper-Threads on your Load Generation Server.

Starting the Schema Build

When you have completed your Build Options click OK to store the values you have entered. For a permanent record the values can be entered directly into the config.xml file. On starting Hammerora the schema options will already contain the values you have entered in the corresponding fields. To begin the schema creation at the buttons in the top level window click the "Create TPC Schema" button or select build from the treeview. Both are shown as three coloured boxes and "Create TPC Schema" appears in the information box when moused over.

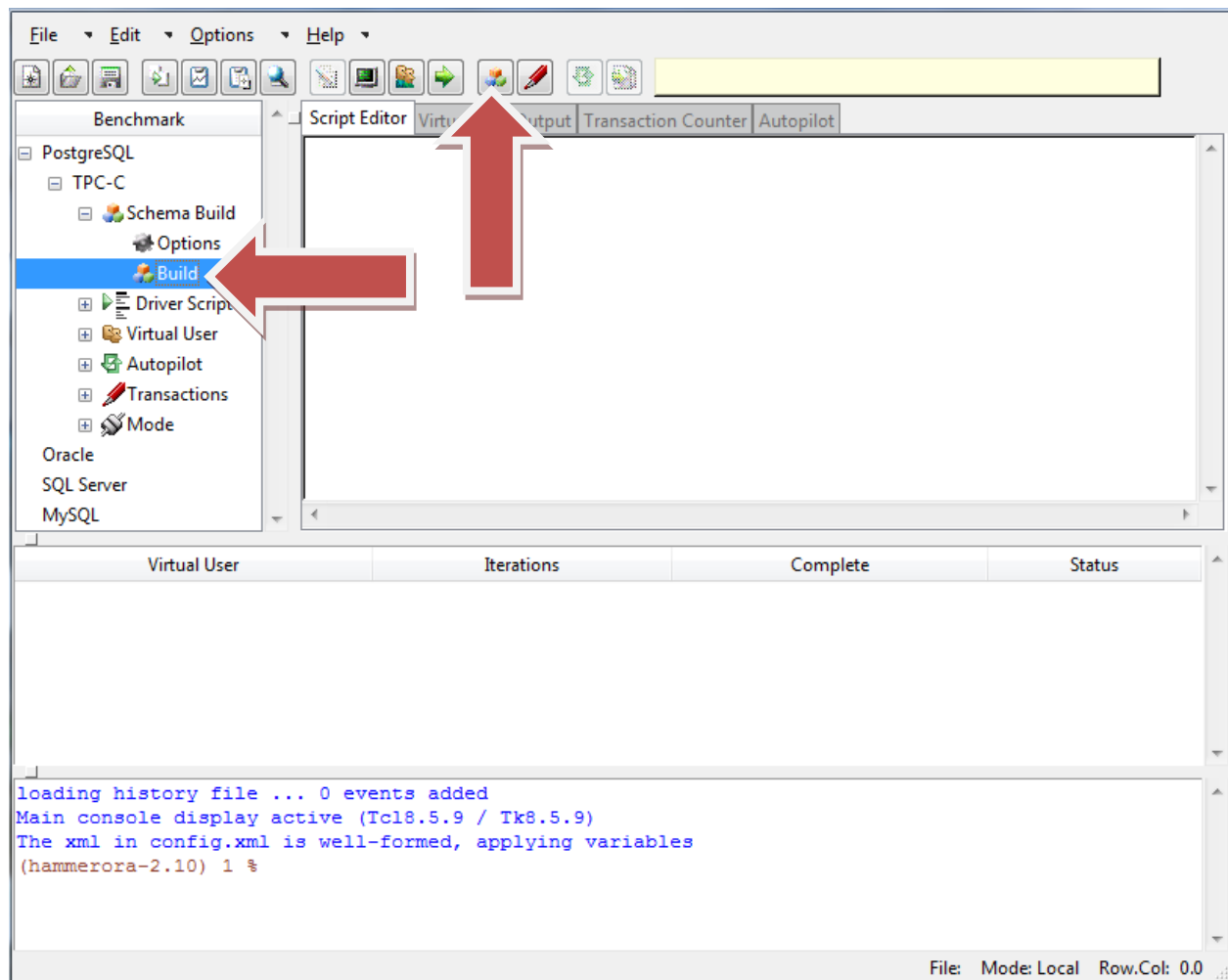


Figure 9 Create Schema

On clicking this button a dialogue box such as the one shown in Figure 10 appears.

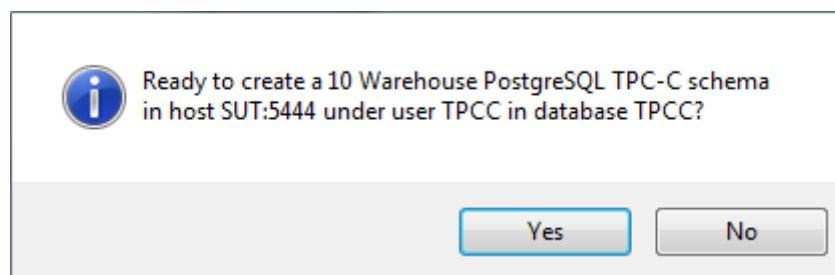


Figure 10 Confirm Schema

When you click Yes Hammerora will login to your chosen PostgreSQL service with a monitor thread as the PostgreSQL superuser and create the user with the password you have chosen. It will then log out and log in again as your chosen user, create the tables and then load the item table data before waiting and monitoring the other threads. The worker threads will wait for the monitor thread to complete its initial work. Subsequently the worker threads will create and insert the data for their assigned warehouses as shown in figure 11. There are no intermediate data files or manual builds required, Hammerora will both create and load your requested data dynamically. Data is inserted in a batch format for optimal network performance.

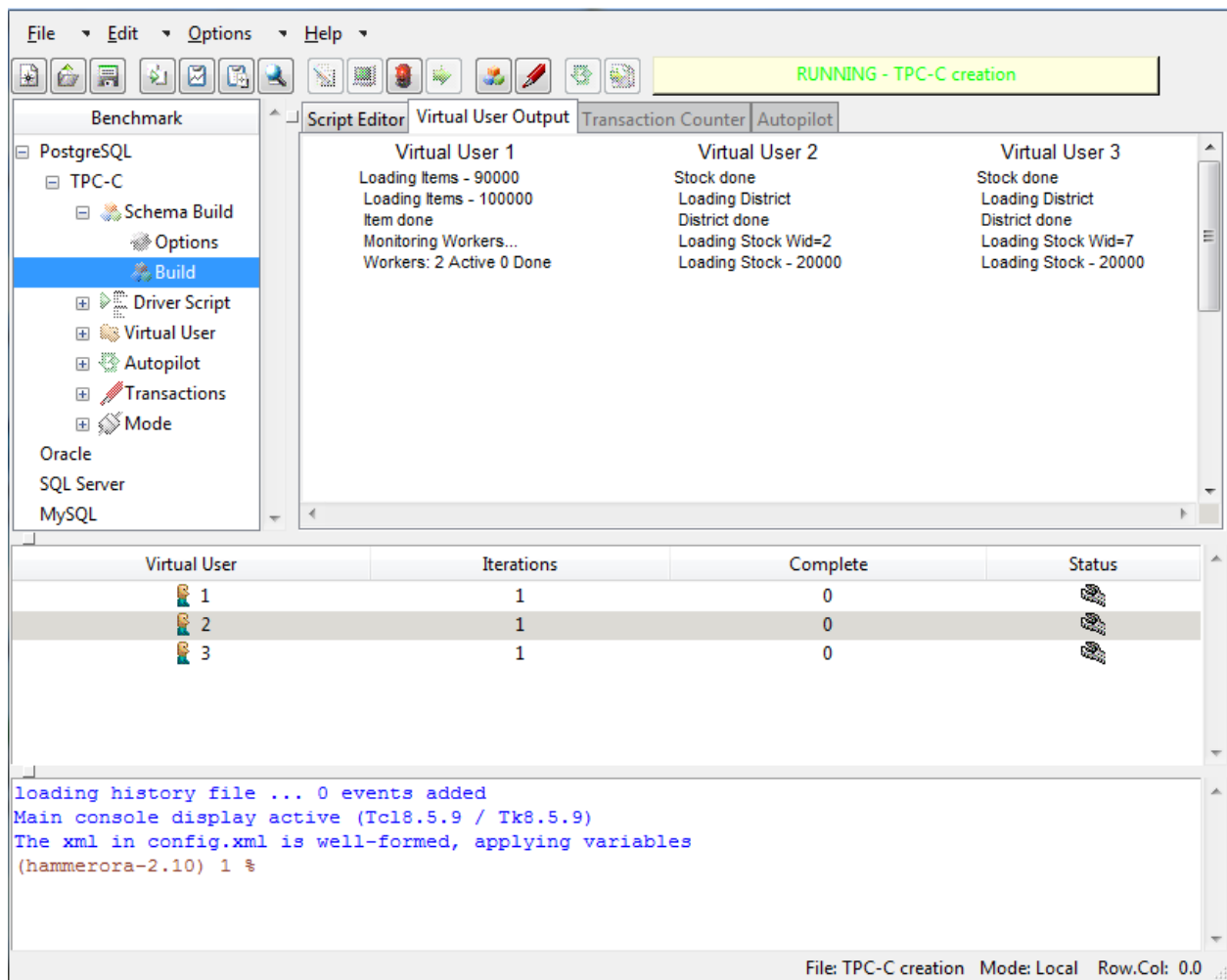


Figure 11 Schema Building

When the workers are complete the monitor thread will create the indexes, stored procedures and gather the statistics. When complete Virtual User 1 will display the message TPCC SCHEMA COMPLETE and all virtual users will show that they completed their action successfully as shown in figure 12.

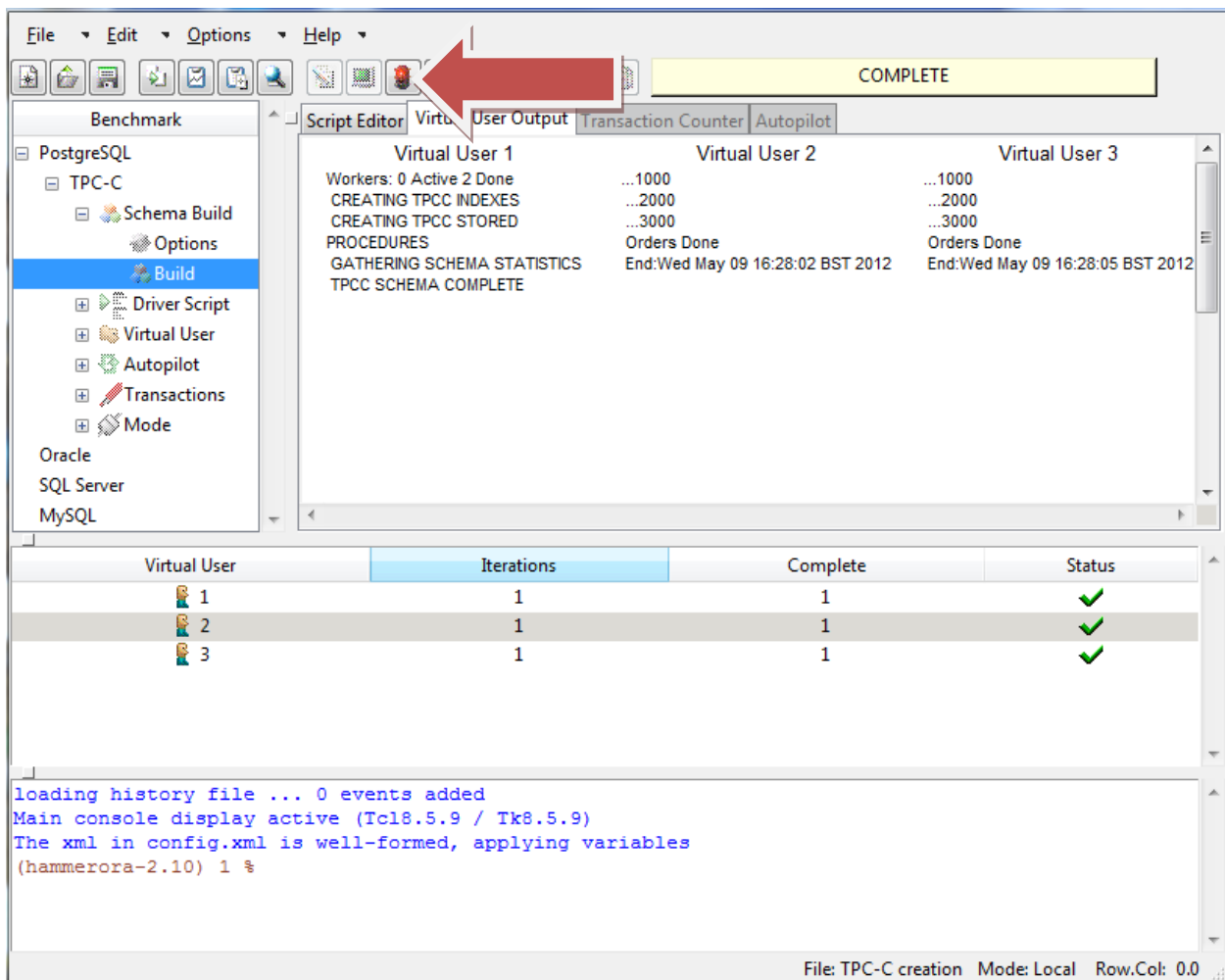


Figure 12 Schema Build Complete

Press the button to destroy the virtual users as shown in figure 12 and clear the script editor as shown in figure 13.

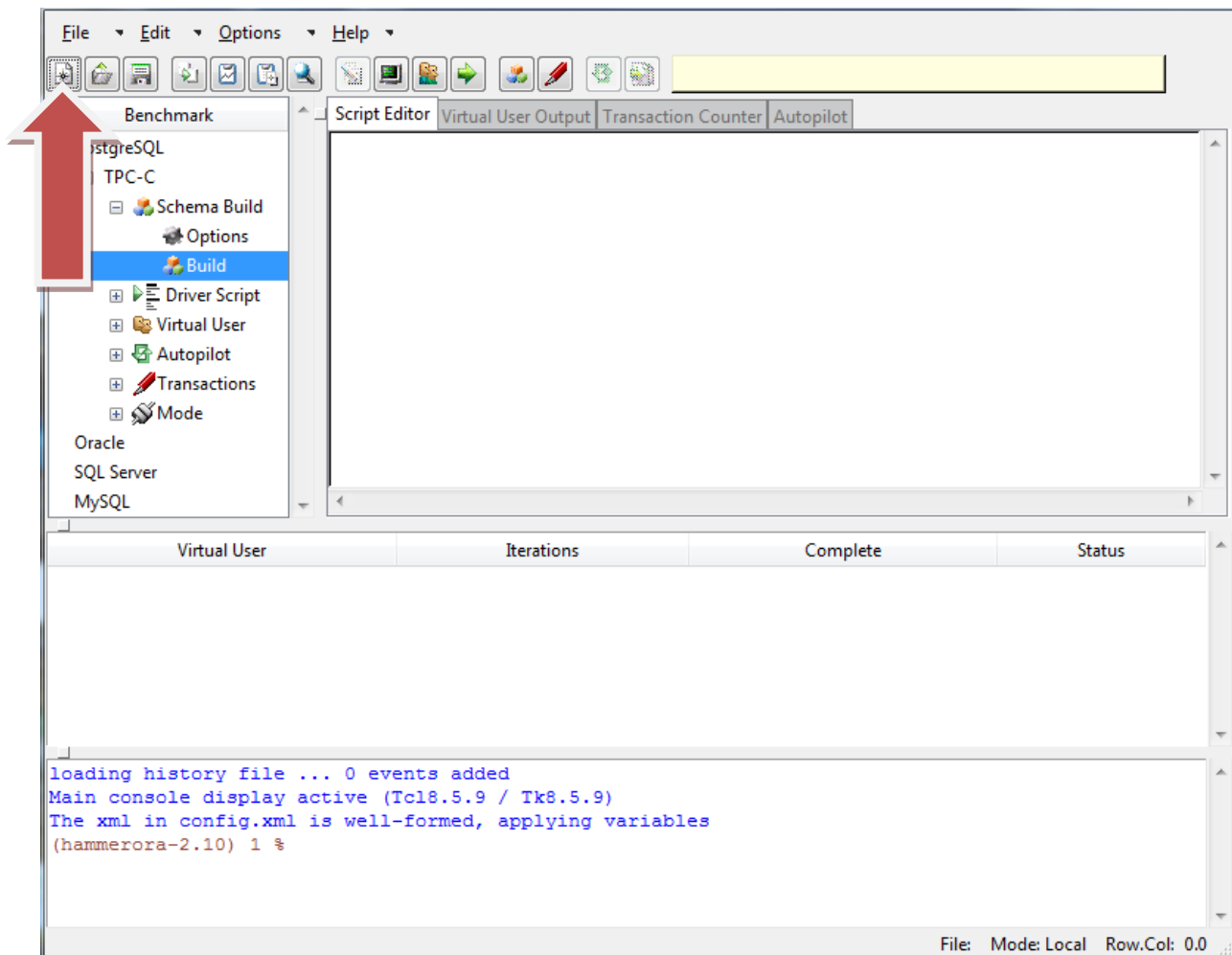


Figure 13 Schema Creation End

The schema build is now complete as an example a 200 warehouse build as follows with nearly 100 million rows should take approximately half an hour or less to create and insert on an up to date 2 socket Linux server.

```
tpcc=#
select relname, n_tup_ins - n_tup_del as rowcount from pg_stat_user_tables;
 relname      | rowcount
-----+-----
stock         | 20000000
history       | 27286657
warehouse     | 200
customer      | 6000000
new_order     | 1866247
item          | 100000
district      | 2000
orders        | 27275059
order_line    | 272749752
(9 rows)
```

The TPC-C schema creation script is a standard Hammerora script like any other so you can save it, modify it and re-run it just like any other Hammerora script. For example if you wish to create more than the 1-5000 warehouses available in the GUI you may notice that the last line in the script calls a procedure with all of the options that you gave in the schema options. Therefore change the second value to any number you like to create more warehouses, for example the following will create 10000 warehouses.

```
do_tpcc SUT 5444 10000 enterprisedb postgres edb tpcc tpcc tpcc true 10
```

Similarly change any other value to modify your script. If you have made a mistake simply close the application and run the following SQL to undo the user you have created.

```
postgres=# drop database tpcc;  
postgres=# drop role tpcc;
```

When you have created your schema you can verify the contents with pgsq or your favourite admin tool as the newly created user. You can also browse the stored procedures you have created by looking in the creation script. At this point the data creation is complete and you are ready to start running a performance test. Before doing so it is worth noting that the schema has been designed in order that you can run multiple tests and it will return the same results. You therefore do not need to recreate your schema after every run for consistent results. Conversely if you do wish to recreate your schema for such a reason as you have exhausted your available disk space the results of tests against different sizes are comparable.

Pre-Testing and Planning

After schema creation but before you start running measured tests an important phase is pre-testing and planning. Pre-testing is a phase also known as 'testing the tests', in this phase you verify that you have the optimal system, operating system and PostgreSQL configuration which you then document and hold consistent for a series of tests. Pre-testing enables you to ensure that your configuration is suitable for testing and the time invested will generate valid results. Pre-testing also enables you to gain familiarity with the Hammerora driver script settings and finally to 'warm the cache' by having your schema data cached in the shared buffers before beginning an extended sequence of tests. Once you are satisfied with your testing configuration you should then thoroughly plan your measured tests to ensure that all of your tests and results are fully documented.

To begin pre-testing select the TPC-C schema options menu tab from the top level Benchmark menu or the Driver Script Options from the treeview as shown in Figure 14.

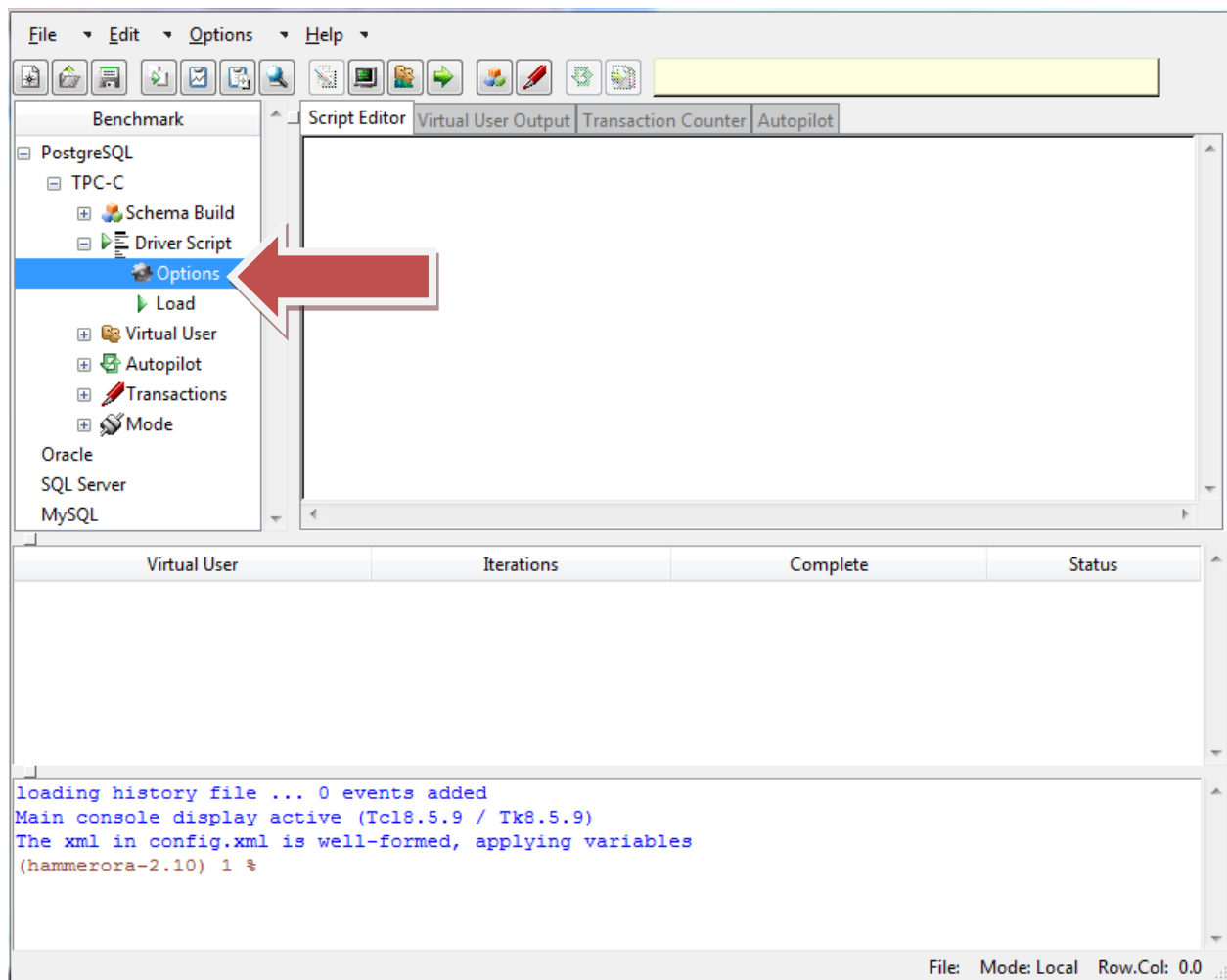


Figure 14 Select Schema Options

At this stage your focus is now on the options given under the section Driver Options as shown in Figure 14.

Driver Options

PostgreSQL Host : localhost

PostgreSQL Port : 5444

PostgreSQL Superuser : enterprisedb

PostgreSQL Superuser Password : postgres

PostgreSQL Default Database : edb

PostgreSQL User : tpcc

PostgreSQL User Password : tpcc

PostgreSQL Database : tpcc

EnterpriseDB Oracle Compatible : ☒

☐ Standard Driver Script

TPC-C Driver Script : ☒ Timed Test Driver Script

Total Transactions per User : 1000000

Exit on PostgreSQL Error : ☐

Keying and Thinking Time : ☐

Vacuum when complete : ☐

EnterpriseDB DRITA Snapshots : ☐

Minutes of Rampup Time : 2

Minutes for Test Duration : 5

OK Cancel

Figure 15 Driver Options

Driver Options

Under the Driver Options section you have the following choices:

PostgreSQL Host

The host name of the SUT running PostgreSQL which the load generation server running Hammerora will connect to.

PostgreSQL Port

The port of the PostgreSQL service. By default this will be 5432 for a standard PostgreSQL installation or 5444 for EnterpriseDB.

PostgreSQL Superuser

The PostgreSQL Superuser is a user with sufficient privileges to create both new users (roles) and databases to enable the creation of the test schema.

PostgreSQL Superuser Password

The PostgreSQL Superuser Password is the password for the PostgreSQL superuser which will have been defined during installation. If you have forgotten the password it can be reset from a psql prompt that has logged in from a trusted connection therefore requiring no password.

```
postgres=# alter role postgres password 'postgres';
```

PostgreSQL Default Database

The PostgreSQL default database is the database to specify for the superuser connection. Typically this will be postgres for a standard PostgreSQL installation or edb for EnterpriseDB.

PostgreSQL User

The PostgreSQL User is the user (role) that will be created that owns the database containing the TPC-C schema.

PostgreSQL User Password

The PostgreSQL User Password is the password that will be specified for the PostgreSQL user when it is created.

PostgreSQL Database

The PostgreSQL Database is the database that will be created and owned by the PostgreSQL User that contains the TPC-C schema.

EnterpriseDB Oracle Compatible

Choosing EnterpriseDB Oracle compatible creates a schema using the Oracle compatible features of EnterpriseDB in an installation of Postgres Plus Advanced Server. This build uses Oracle PL/SQL for the creation of the stored procedures and is shown as the parameter ora_compatible in the editable options section of the script.

TPC-C Driver Script

Under TPC-C Driver script you have the option of choosing either the Standard Driver Script or the Timed Test Driver Script. This choice will dynamically change the Driver Script that is loaded when the TPC-C Driver Script menu option is chosen. The Standard Driver Script as shown in figure 16 is a script run by all virtual users. This script should be chosen where you wish to create a load against the database and view the transaction rate but do not wish to run a timed test or wish to time the tests manually yourself. The Standard Driver Script may be run with Virtual User Output turned on, which will display all of the information each virtual users processes or with Virtual User Output turned off to be able to observe the transaction rate only. Your additional Driver Options choices are populated in the EDITABLE OPTIONS section.

```
#!/usr/local/bin/tclsh8.5
if [catch {package require Pgctl} ] { error "Failed to load Pgctl - Postgres Library Error" }
#EDITABLE OPTIONS#####
set total_iterations 1000000 ;# Number of transactions before logging off
set RAISEERROR "false" ;# Exit script on PostgreSQL (true or false)
set KEYANDTHINK "false" ;# Time for user thinking and keying (true or false)
set ora_compatible "true" ;#Postgres Plus Oracle Compatible Schema
set host "SUT" ;# Address of the server hosting PostgreSQL
set port "5444" ;# Port of the PostgreSQL Server
set user "tpcc" ;# PostgreSQL user
set password "tpcc" ;# Password for the PostgreSQL user
set db "tpcc" ;# Database containing the TPC Schema
#EDITABLE OPTIONS#####
#RANDOM NUMBER
proc RandomNumber {m M} {return [expr {int($m+rand()*($M+1-$m))}]}
```

Figure 16 Standard Driver Script

Instead of the Standard Driver Script you can select the Timed Test Driver Script. As shown in Figure 17 this produces a number of additional options. You should select the Timed Test Driver Script when you wish to

run timed tests and have Hammerora time these tests, measure the results, report on an average transaction rate for a period of time and optionally generate DRITA information for that test. With the Timed Test Driver Script the first virtual user will do the timing and generate the results with the additional virtual users running the workload, therefore you should always select the number of desired virtual users + 1 when running the Timed Test Driver Script. For example if you wish to measure a load generated by two virtual users you should select three virtual users before running the script. Additionally the Timed Test Driver Script is designed to be run with Virtual User Output enabled, this ensures that the information gathered by the first virtual user on the transaction rates and AWR report numbers are correctly reported. Whilst running the Timed Test Driver Script virtual user output for the virtual users generating the load is suppressed.

```
#!/usr/local/bin/tclsh8.5
if [catch {package require Pgtcl} ] { error "Failed to load Pgtcl - Postgres Library Error" }
#EDITABLE OPTIONS#####
set total_iterations 1000000 ;# Number of transactions before logging off
set RAISEERROR "false" ;# Exit script on PostgreSQL (true or false)
set KEYANDTHINK "false" ;# Time for user thinking and keying (true or false)
set rampup 2; ;# Rampup time in minutes before first Transaction Count is taken
set duration 5; ;# Duration in minutes before second Transaction Count is taken
set mode "Local" ;# Hammerora operational mode
set VACUUM "false" ;# Perform checkpoint and vacuum when complete (true or false)
set DRITA_SNAPSHOTS "false";#Take DRITA Snapshots
set ora_compatible "true" ;#Postgres Plus Oracle Compatible Schema
set host "SUT" ;# Address of the server hosting PostgreSQL
set port "5444" ;# Port of the PostgreSQL server
set superuser "enterisedb" ;# Superuser privilege user
```

Figure 2 AWR Snapshot Driver Script

For both the Standard Driver Script and AWR Driver Script the further options selected within the Schema Options window are entered automatically into the EDITABLE OPTIONS section of the driver script as follows:

Total Transactions per User

Total transactions per user is reported as total_iterations within the EDITABLE OPTIONS section of the driver script. This value will set the number of transactions each virtual user will process before logging off. You can use this value to determine how long the virtual user will remain active for. The length of time for activity will depend upon the performance of the Database Server under test. A higher performing server will process the defined number of transactions more quickly than a lower performing one.

It is important to draw the distinction between the total_iterations value and the Iterations value set in the Virtual User Options window. The Iterations value in the Virtual User Options window determines the number of times that a script will be run in its entirety. The total_iterations value is internal to the TPC-C driver script and determines the number of times the internal loop is iterated ie

```
for {set it 0} {$it < $total_iterations} {incr it} { ... }
```

In other words if total_iterations is set to 1000 then the executing user will log on once execute 1000 transactions and then log off. If on the other hand Iterations in the Virtual User Options window is set to 1000 and total_iterations in the script set to 1 then the executing user will log on execute one transaction and then log off 1000 times. For the TPC-C driver script I recommend only modifying the total_iterations value.

When running the Timed Test Driver Script as the test is timed you should ensure that the number of transactions is set to a suitably high value to ensure that the virtual users do not complete their tests before the timed test is complete, doing so will mean that you will be timing idle virtual users and the results will be invalid. Consequently it is acceptable when running timed tests to set the Total Transactions per User to a

high value such as 1000000 (now the default value from Hammerora 2.5) or more to ensure that the virtual users continue running for a long period of time, When the test is complete you can stop the test running by stopping the virtual users.

Exit on PostgreSQL Error

Exit on PostgreSQL Error is shown as the parameter RAISEERROR in the Driver Script. RAISEERROR impacts the behaviour of an individual virtual user on detecting an PostgreSQL error. If set to TRUE on detecting an PostgreSQL error the user will report the error into the Hammerora console and then terminate execution. If set to FALSE the virtual user will ignore the error and proceed with executing the next transaction. It is therefore important to be aware that if set to FALSE firstly if there has been a configuration error resulting in repeated errors then the workload might not be reported accurately and secondly you may not be aware of any occasional errors being reported as they are silently ignored. I recommend running pre-tests with RAISEERROR set to TRUE to ensure a configuration is valid before setting it to FALSE for a measured test run.

Keying and Thinking Time

Keying and Thinking Time is shown as KEYANDTHINK in the Driver Script. A good introduction to the importance of keying and thinking time is to read the TPC-C specification. This parameter will have the biggest impact on the type of workload that your test will take.

TIP: The most common configuration error is to run a test with Keying and Thinking Time set to False with too many virtual users for the schema created. One virtual user without keying and thinking time will generate a workload equivalent to many thousands of users with keying and thinking time enabled. Without keying and thinking time you are likely to see peak performance at or around the number of cores/Hyper Threads on your Database Server.

Keying and thinking time is an integral part of an official TPC-C test in order to simulate the effect of the workload being run by a real user who takes time to key in an actual order and think about the output. If KEYANDTHINK is set to TRUE each user will simulate this real user type workload. An official TPC-C benchmark implements 10 users per warehouse all simulating this real user experience and it should therefore be clear that the main impact of KEYANDTHINK being set to TRUE is that you will need a significant number of warehouses and users in order to generate a meaningful workload. and hence an extensive testing infrastructure. The positive side is that when testing hundreds or thousands of virtual users you will be testing a workload scenario that will be closer to a real production environment. Whereas with KEYANDTHINK set to TRUE each user will execute maybe 2 or 3 transactions a minute you should not underestimate the radical difference that setting KEYANDTHINK to FALSE will have on your workload. Instead of 2 or 3 transactions each user will now execute tens of thousands of transactions a minute. Clearly KEYANDTHINK will have a big impact on the number of virtual users and warehouses you will need to configure to run an accurate workload, if this parameter is set to TRUE you will need at least hundreds of virtual users and warehouses, if FALSE then you will need to begin testing with 1 or 2 threads, building from here up to a maximum workload with the number of warehouses set to a level where the users are not contending for the same data. A common error is to set KEYANDTHINK to FALSE and then create hundreds of users for an initial test, this form of testing will only exhibit a massive contention for data between users and nothing about the potential of the system. If you do not have an extensive testing infrastructure and a large number of warehouses configured then I recommend setting KEYANDTHINK to FALSE (whilst remembering that you are not simulating a real TPC-C type test) and beginning your testing with 1 virtual user building up the number of virtual users for each subsequent test in order to plot a transaction profile.

Vacuum when Complete

The Vacuum when Complete option is shown as VACUUM in the Driver Script. This option is only available in the Timed Test Driver Script and causes the Virtual Users to finish running the test and a PostgreSQL Checkpoint and Vacuum on the database being tested when a test is complete. For consistency between tests this option to do a checkpoint and vacuum is strongly recommended. This value is of particular use when running the Timed Test script in Autopilot Mode. To implement multi version concurrency control (MVCC) PostgreSQL will maintain multiple versions of rows to represent a snapshot of the data at the time a transaction started. Performing a vacuum after each tests ensures that snapshot data generated during a test that is no longer needed is removed.

Minutes of Rampup Time

The Minutes of Ramup Time is shown as rampup in the Driver Script. The rampup time defines the time in minutes for the monitoring virtual user to wait for the virtual users running the workload to connect to the database and build up the transaction rate by caching data in the database buffer cache before taking the first snapshot and timing the test. The rampup time should be sufficiently long enough for a workload to reach a steady transaction rate before the first snapshot is taken.

Enterprise DB DRITA Snasphots

If you have Enterprise DB installed and DRITA functionality enabled, by selecting this option Hammerora will automatically take DRITA snapshots for performance analysis of the workload between tests.

Minutes for Test Duration

The Minutes for Test Duration is shown as duration in the Driver Script. The test duration defines the time of the test measured as the time the monitor thread waits after the first snapshot before taking the second one to signal the test is complete and the active virtual users to complete their workload.

Mode Options

The mode value is taken from the operational mode setting set under the Mode Options menu tab under the Mode menu. If set to Local or Master then the monitor thread takes snapshots, if set to Slave no snapshots are taken. This is useful if multiple instances of Hammerora are running in Master and Slave mode to ensure that only one instance takes the snapshots.

When you have completed defining the Schema Options click OK to save your values. As noted previously under the section Load Generation Server Configuration you can also enter these values into the config.xml file to save a permanent record of your values for pre-populating the values after restarting Hammerora.

Loading the Driver Script

Once you have selected and saved your driver options under the Benchmark Menu select TPC-C and TPC-C Driver Script or select Load from the treeview as shown in Figure 18.

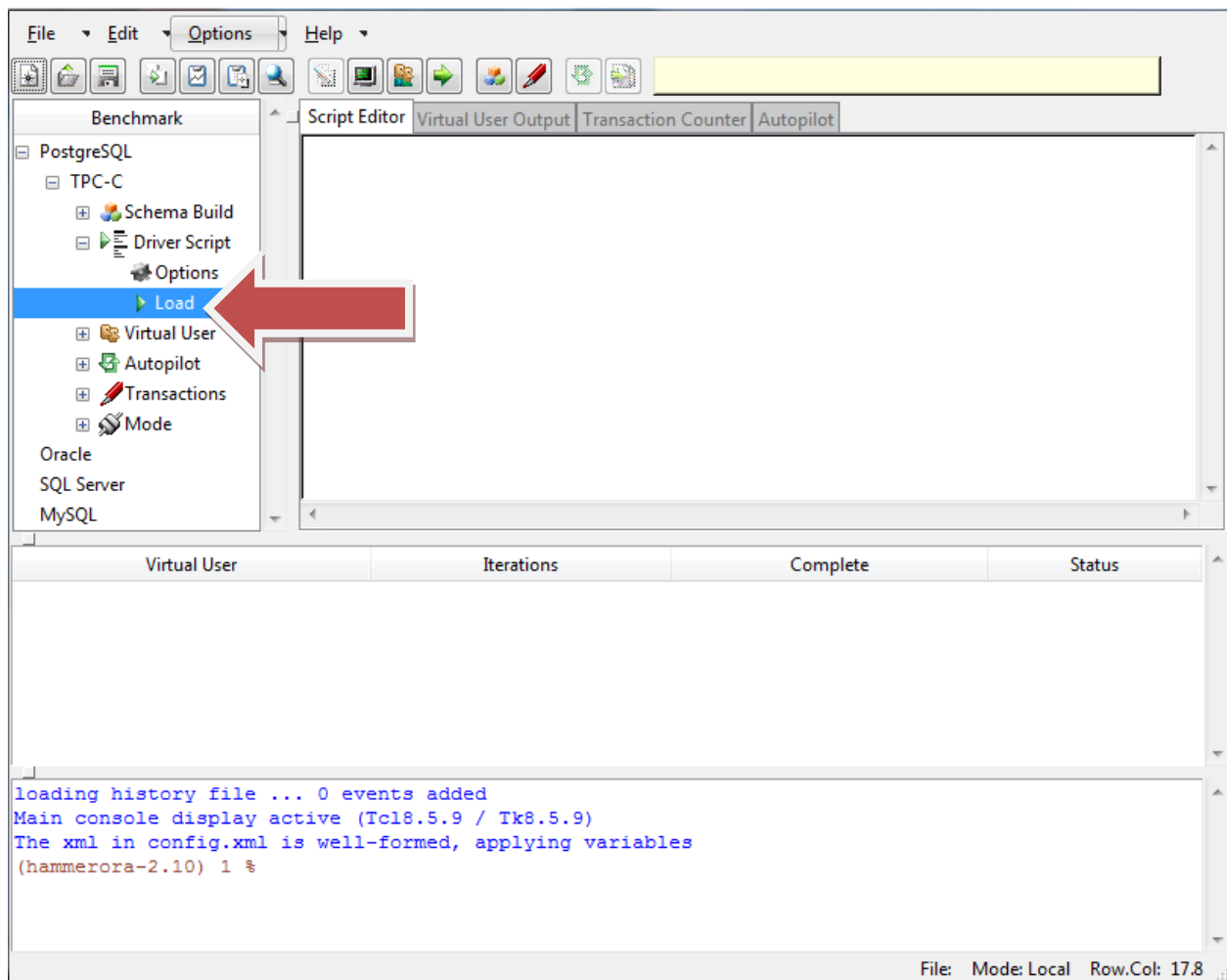


Figure 18 Select Driver Script

This will populate the Script Editor window with the driver script shown in Figure 16 or 17 according to whether the standard or timed test driver script is chosen. These scripts provide the interaction from the Load Generation Server to the schema on the SUT Database Server. If you have correctly configured the parameters in the Driver Options section you do not have to edit in the script. If you so choose however you may also manually edit the the values given in the EDITABLE OPTIONS section. Additionally the driver scripts are regular Hammerora scripts and a copy may be saved externally and modified as you desire for a genuinely Open Source approach to load testing.

Pre-Test 1 Verifying the Schema

Figure 19 shows a successfully loaded Standard Driver Script which provides a useful first test against a newly created TPC-C Schema.

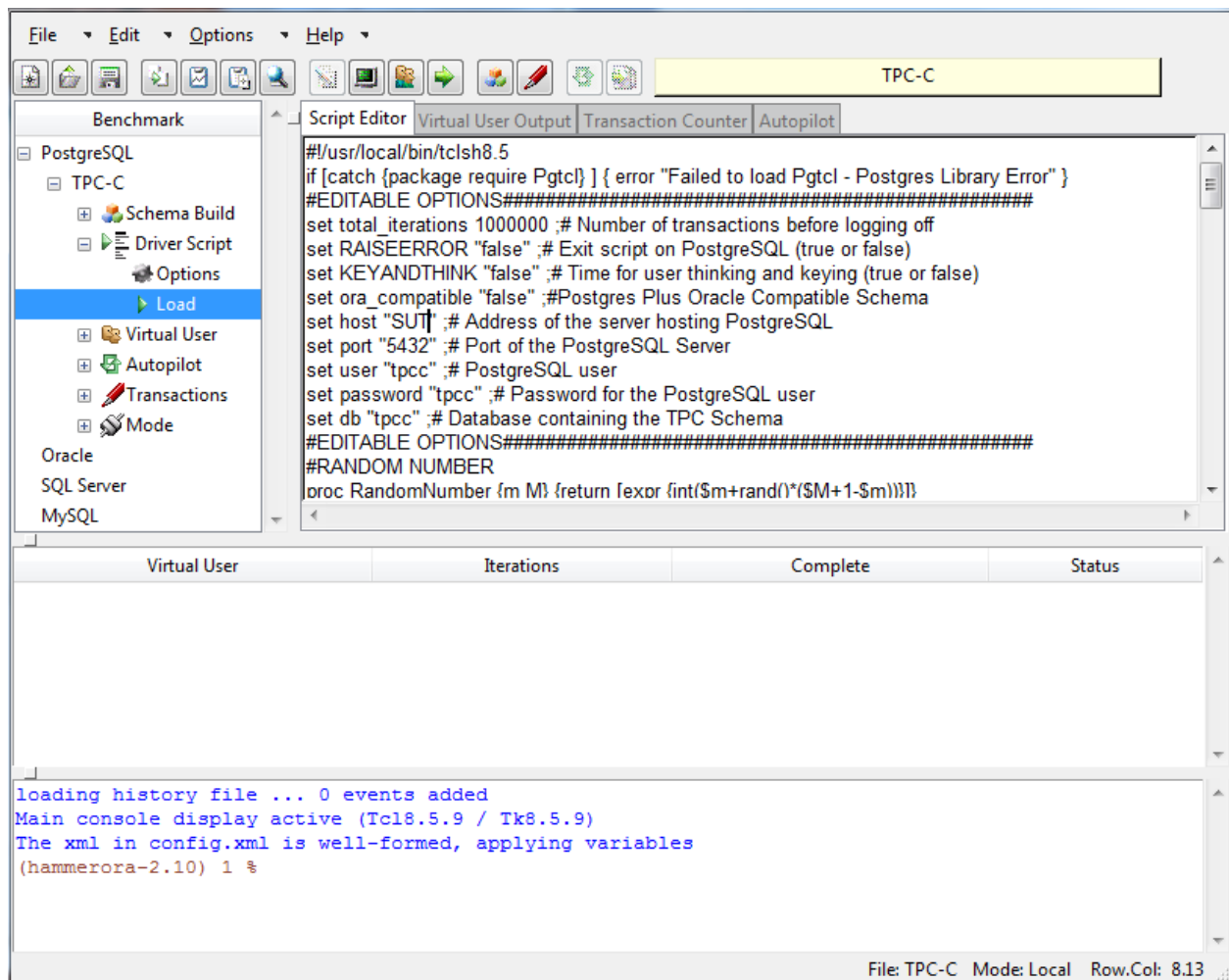


Figure 19 TPC-C Driver Script

In this example we will create two virtual users and choose to display their output to verify the schema and database configuration. To do this Under the Options menu or from the treeview as shown in Figure 20 select the Virtual User Options and enter the number 2. Also check the Show Output button to see what your users are doing whilst the test is running. Note that displaying the output will reduce the overall level of performance (although Hammerora is multi-threaded many Window display systems are not and a display can only be updated by a single thread thereby limited performance) and click OK. Showing output is OK here as it is running a pre-test and not a performance test.

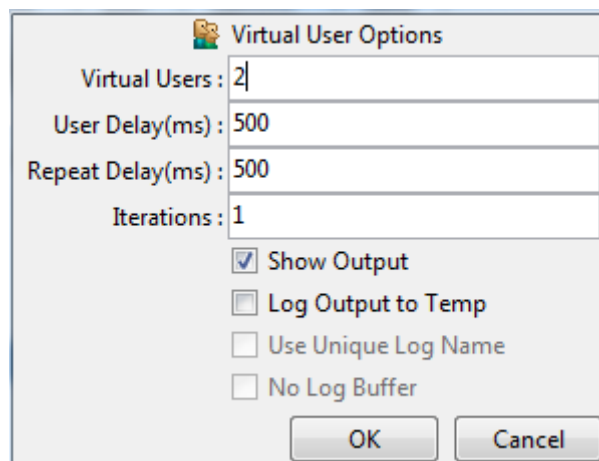


Figure 20 Select Virtual Users

There are three other related options under the Virtual User Options dialogue, namely User Delay(ms),

Repeat Delay(ms) and Iterations. Iterations defines the number of times that Hammerora should execute a script in its entirety. With regards to running the TPC-C driver script this can be thought of as the number of times a Virtual User logs on to the database, runs the number of transactions you defined in Total Transactions per User and logs off again. For example if Total Transactions per User was set to 1000 and the Virtual Users Iterations was set to 10, the Virtual User would complete 10000 transactions in total logging off and on between each run. Setting Total Transactions per User to 10000 and Virtual User Iterations to 1 would also complete 10,000 transactions per virtual user but all in one session. User Delay(ms) defines the time to wait between each Virtual User starting its test and the Repeat Delay(ms) is the time that each Virtual User will wait before running its next Iteration. For the TPC-C driver script the recommended approach is to leave the Iterations and User and Repeat Delays at the default settings and only modify the Total Transactions per User or total_iterations value inside the Driver Script. When you have completed the selection press OK. Click the Create Virtual Users button or the Create Virtual User treeview option as shown in Figure 21 to create the virtual users, they will be created but not start running yet.

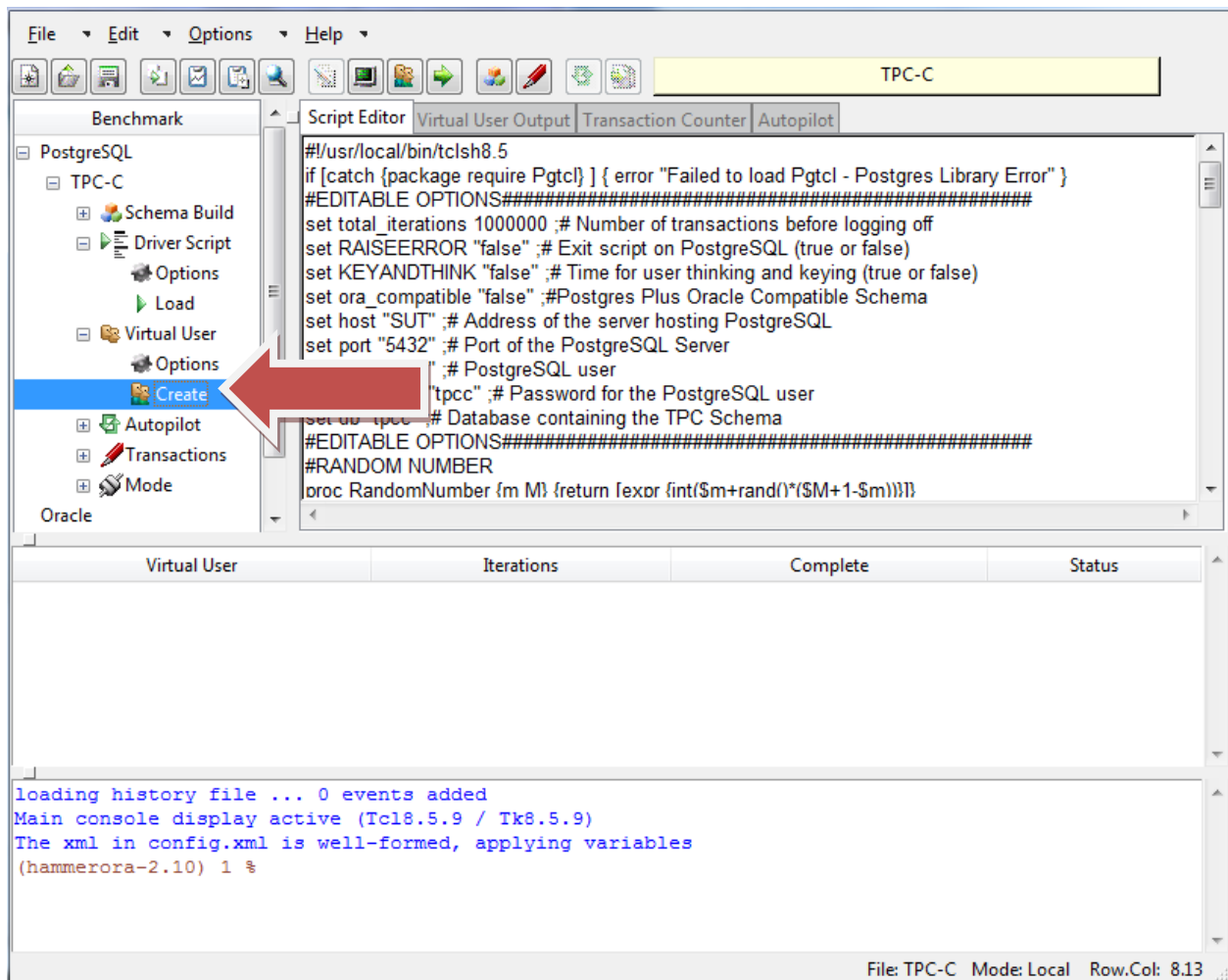


Figure 21 Create Virtual Users

You can observe as shown in Figure 22 that the virtual users have been created but are showing a status of idle. You can destroy the Virtual Users by pressing the Red Traffic light icon that has appeared in place of the Create Virtual Users button.

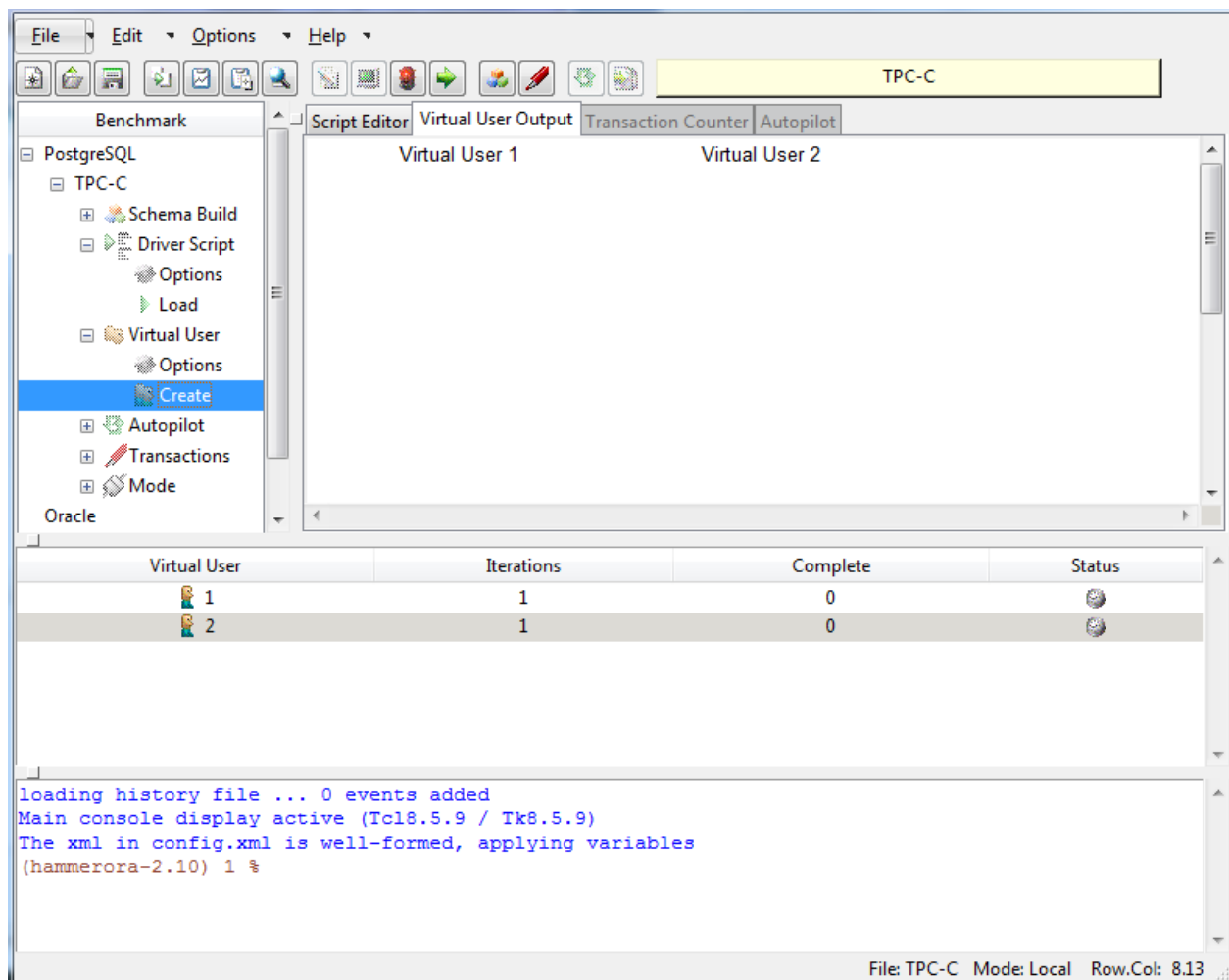


Figure 22 Virtual Users Created

To begin the test press the button Run Virtual Users button as shown in Figure 23, the name of the button will appear in the information pane.

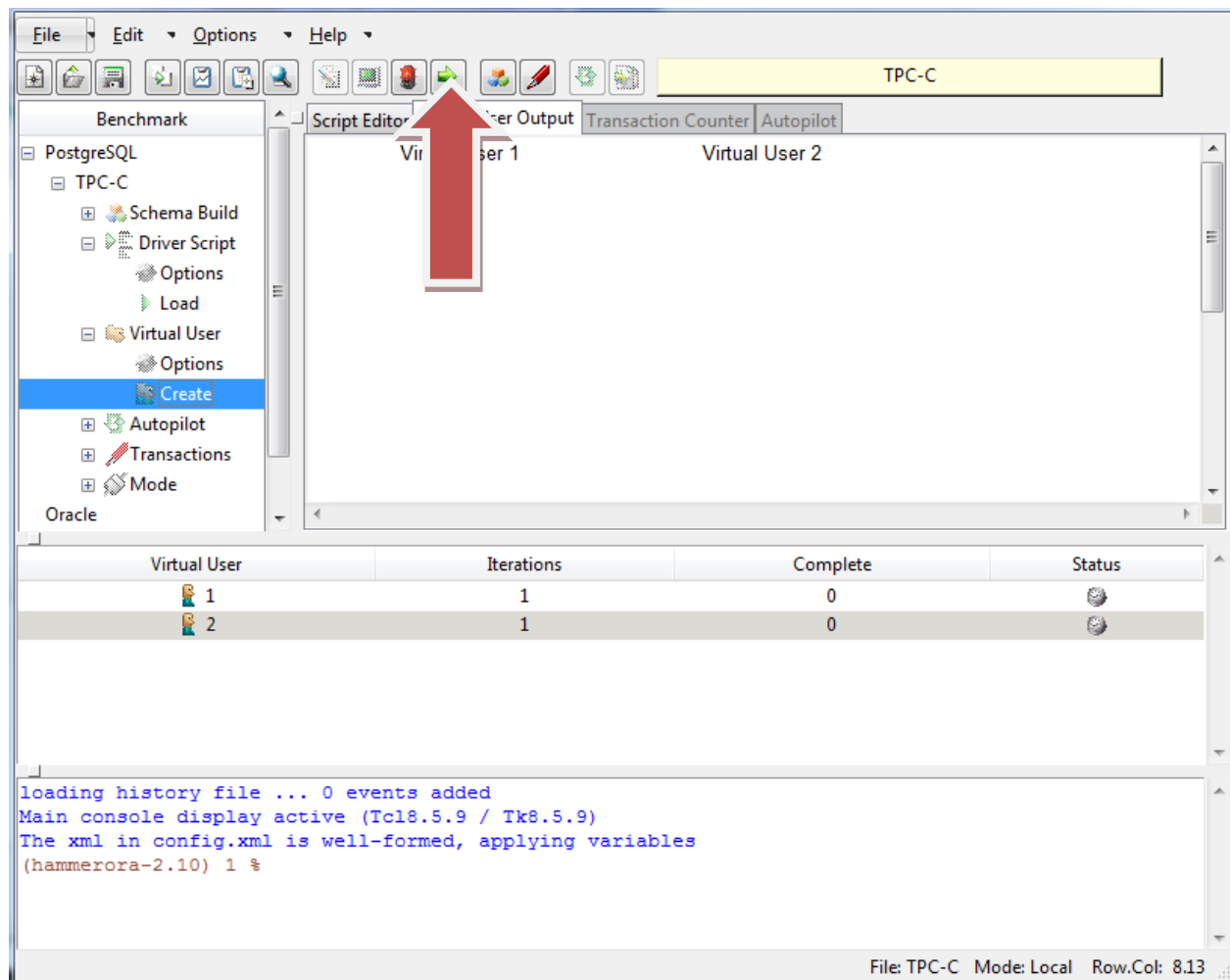


Figure 23 Run Hammerora Load Test

You can observe the Virtual User icon change to signify activity. The Virtual Users have logged on to the database, you will be able to see their presence in V\$SESSION for example

```
edb=# select * from pg_stat_activity;
```

...

```
16585 | tpcc      | 7912 | 16584 | tpcc      | 127.0.0.1 |
      |          |      |      |          |          |
      |          |      |      |          |          | 10-MAY-12 21:52:55.472 +01:00 |
      |          |      |      |          |          | 10-MAY-12 11:52:58.982 +01:00 | f          | <IDLE>

16585 | tpcc      | 6064 | 16584 | tpcc      | 127.0.0.1 |
      |          |      |      |          |          |
      |          |      |      |          |          | 10-MAY-12 21:52:55.987 +01:00 |
      |          |      |      |          |          | 10-MAY-12 21:52:58.982 +01:00 | f          | <IDLE>
```

are running transactions as can be observed in the Virtual User Output as shown in Figure 24.

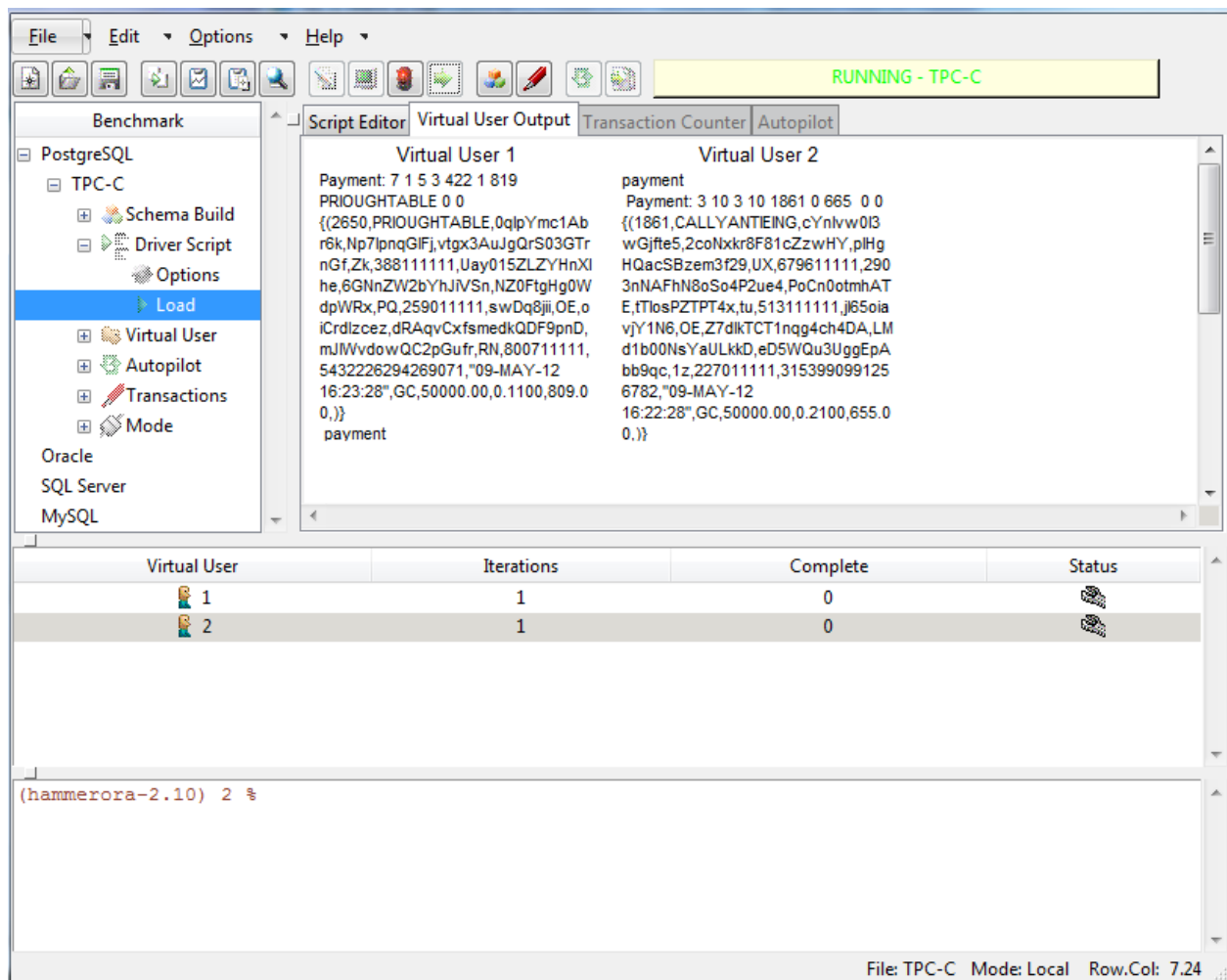


Figure 24 Load Testing Running

When the Virtual Users have completed all of their designated transactions they will exit showing a positive status as shown in Figure 25. Once the Virtual User is displaying this positive status it has logged off the database and will not be seen in `pg_stat_activity`. The Virtual User is once again idle and not running transactions. The Virtual User does not need to be destroyed and recreated to re-run the test from this status. The Virtual Users can be destroyed to stop a running test.

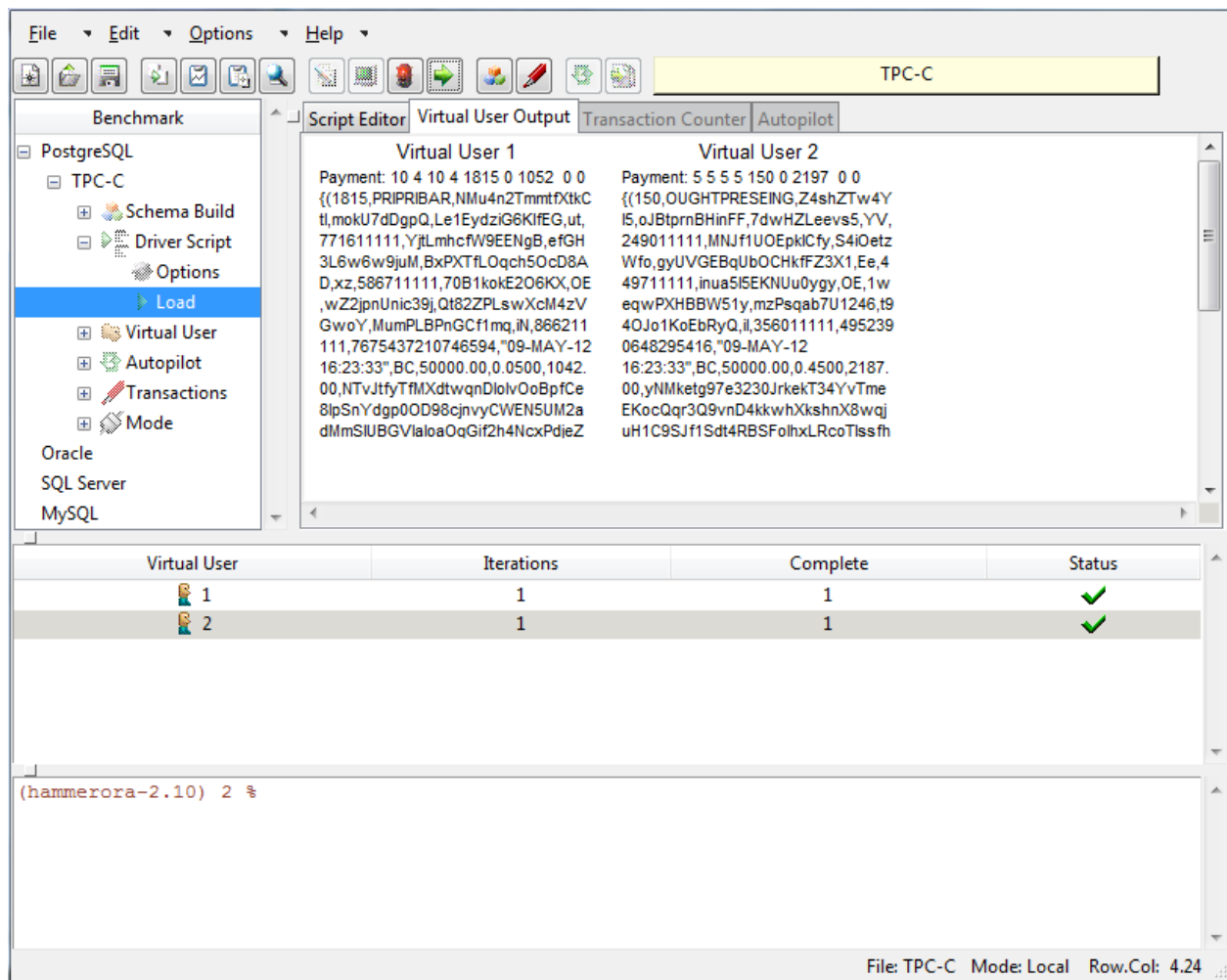


Figure 25 Virtual Users Complete

If there is an error when running the Driver Script it will be reported in the Virtual User icon with the detail of the error shown in the Console window. Figure 26 shows an example of an error, in this case multiple failures have been recorded from attempting to run a non-Oracle compatible test script against an EnterpriseDB Oracle compatible schema . Setting the RAISEERROR parameter will display further information. The Virtual User is once again idle and not running transactions. The Virtual User does not need to be destroyed and recreated to re-run the test from this status.

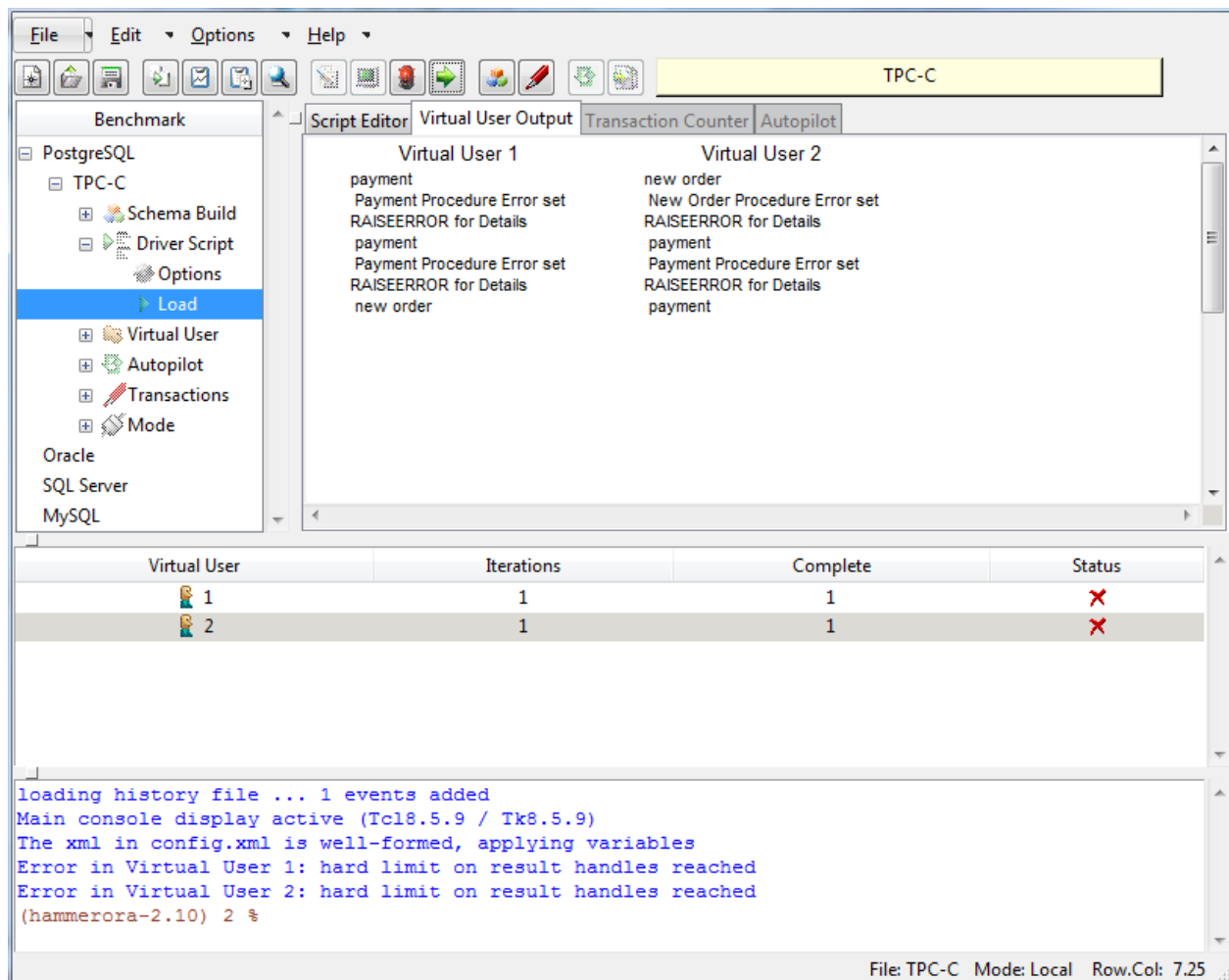


Figure 26 Virtual User Error

At this stage in pre-testing the test configuration has been verified and it has been demonstrated that the load generation server can log on to the SUT Database Server and run a test.

Pre-Test 2 Single and Multiple Virtual User Throughput

Once the configuration has been verified the next stage is to focus upon performance. The best place to start with verifying performance is to monitor the workload of a single Virtual User. To do this follow all of the steps for Pre-Test 1 ensuring that you select the Standard Driver Script. Note that the Timed Test Driver Script is designed for multiple users with one Virtual User providing the monitoring capabilities for the other Virtual Users. Consequently if one Virtual User is configured to run the Timed Test Driver Script it will result in one Virtual monitoring an idle workload which is almost certainly not the desired outcome. Once the Standard Driver Script has been loaded configure a single Virtual User as shown in Figure 27. Configure One Virtual user without selecting the Show Output check box (The reason for suppressing output is described under Pre-Test 1).

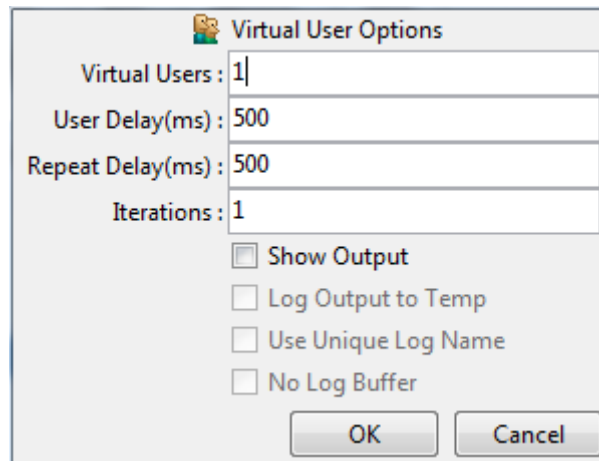


Figure 27 One Virtual User

Note that a single Virtual User without output is the default configuration if you have not modified the config.xml file and therefore creating the Virtual Users will give you this single Virtual Configuration without specifically configuring the Virtual Users as shown in Figure 27. Figure 28 shows the single Virtual User created and the Standard Driver script loaded.

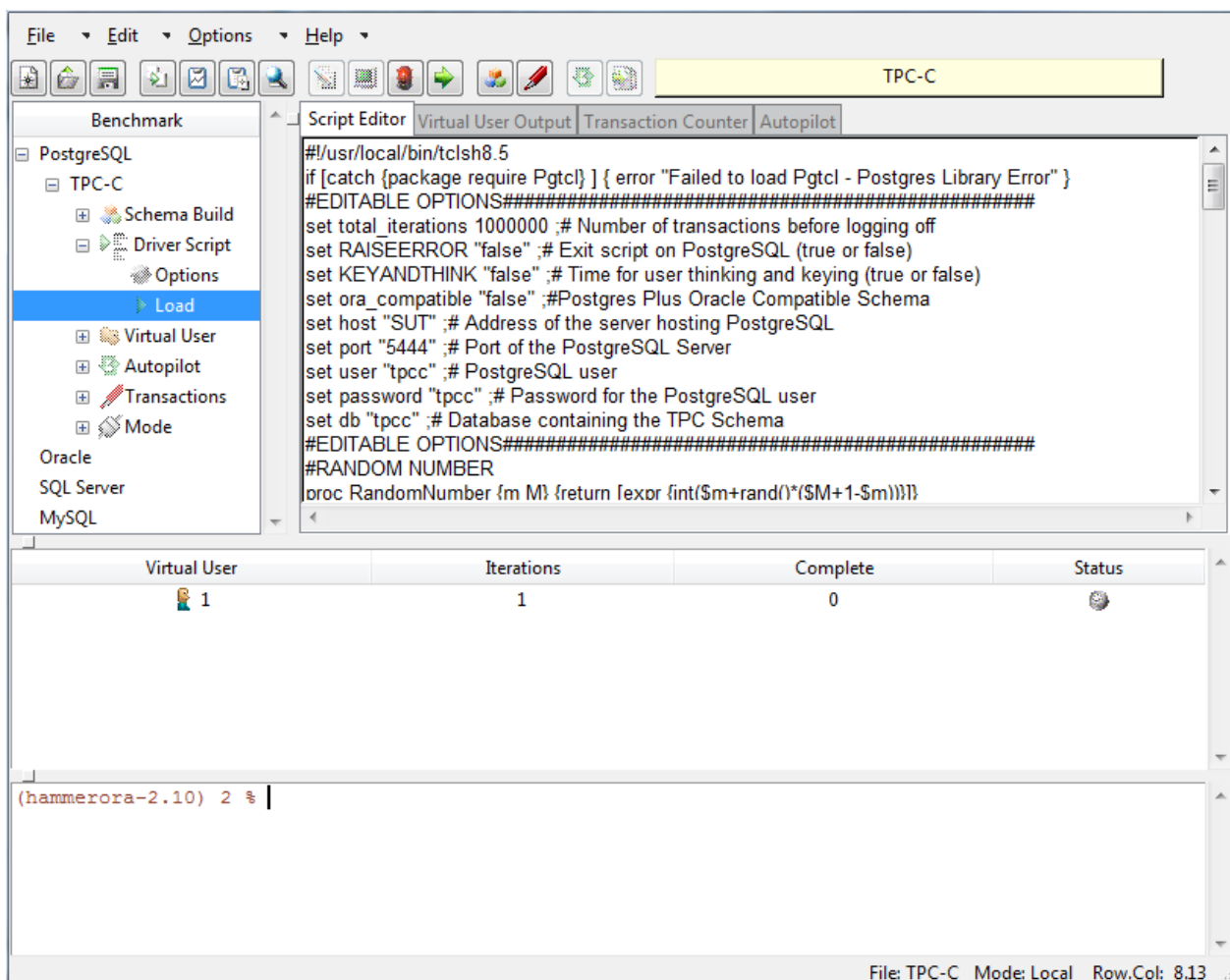


Figure 28 Single Virtual User

Press the Run Virtual Users button as described previously to begin generating the Single User Throughput test. As shown in figure 29 the Virtual User icon has been updated to signify that the workload is running.

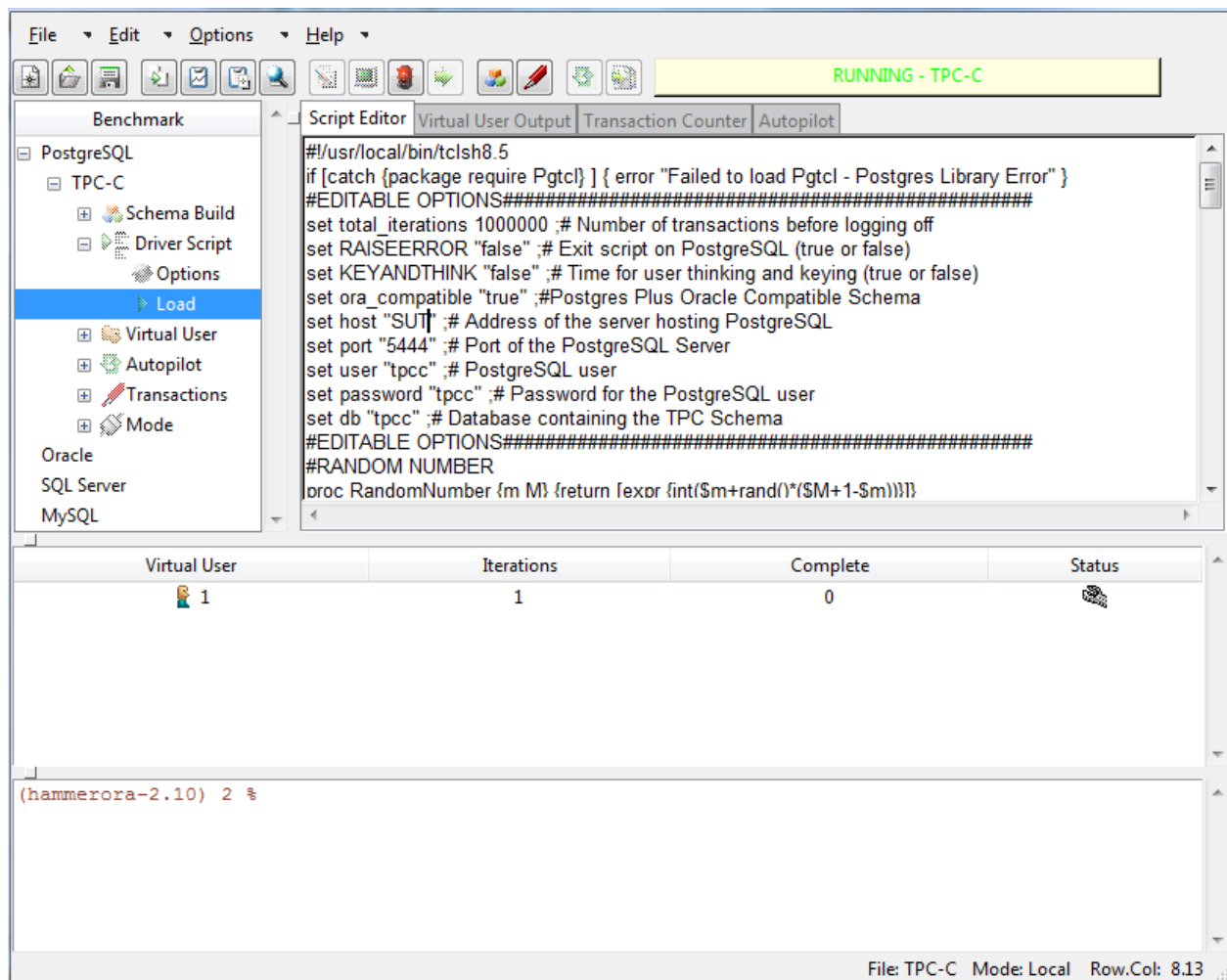


Figure 29 Single User Throughput Test

To observe performance during the test you can use the Transaction Counter. The Transaction Counter options can be selected from the treeview or the options menu as shown in Figure 30.

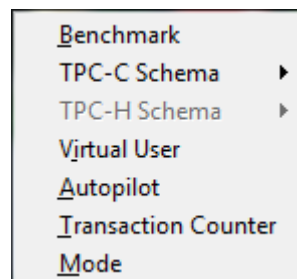
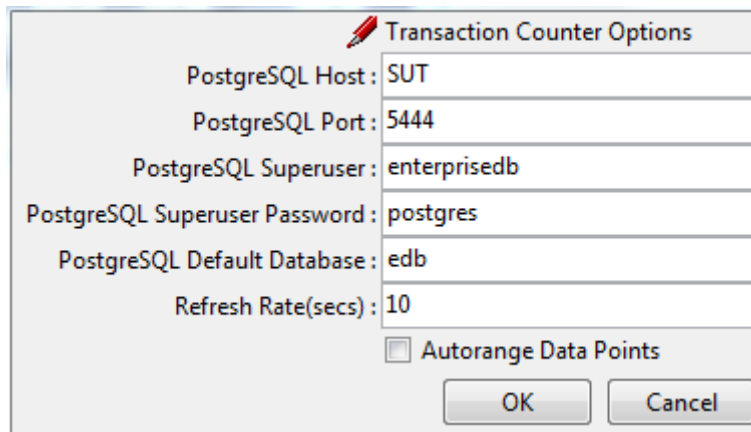


Figure 30 TX Counter Options

This displays the Transaction Counter Options as shown in Figure 31.

A screenshot of a Windows-style dialog box titled "Transaction Counter Options" with a red pencil icon. It contains several text input fields: "PostgreSQL Host" with "SUT", "PostgreSQL Port" with "5444", "PostgreSQL Superuser" with "enterprisedb", "PostgreSQL Superuser Password" with "postgres", and "PostgreSQL Default Database" with "edb". There is also a "Refresh Rate(secs)" field with "10". Below these is a checkbox labeled "Autorange Data Points" which is currently unchecked. At the bottom right are "OK" and "Cancel" buttons.

PostgreSQL Host :	SUT
PostgreSQL Port :	5444
PostgreSQL Superuser :	enterprisedb
PostgreSQL Superuser Password :	postgres
PostgreSQL Default Database :	edb
Refresh Rate(secs) :	10
<input type="checkbox"/> Autorange Data Points	
OK Cancel	

Figure 31 Transaction Counter Options

Transaction Counter Options

Under the Transaction Counter Options section you have the following choices:

PostgreSQL Host

The host name of the SUT running PostgreSQL which the load generation server running Hammerora will connect to.

PostgreSQL Port

The port of the PostgreSQL service. By default this will be 5432 for a standard PostgreSQL installation or 5444 for EnterpriseDB.

PostgreSQL Superuser

The PostgreSQL Superuser is a user with sufficient privileges to create both new users (roles) and databases to enable the creation of the test schema.

PostgreSQL Superuser Password

The PostgreSQL Superuser Password is the password for the PostgreSQL superuser which will have been defined during installation.

PostgreSQL Default Database

The PostgreSQL default databases is the database to specify for the superuser connection. Typically this will be postgres for a standard PostgreSQL installation or edb for EnterpriseDB.

Refresh Rate

The refresh rate defines the time in seconds between when the transaction counter will refresh its values. Setting this value too low may impact the accuracy of the data reported by the PostgreSQL database and the default value of 10 seconds is a good choice for an accurate representation.

Autorange Data Points

By default the Data Points in the transaction counter will be anchored to the data point Zero. By selecting Autorange data points you enable the transaction counter to zoom in to show a finer detail of peaks and troughs in your transaction Data

When you have completed the transaction counter options press OK to save your values and press the Transaction Counter button as shown in Figure 32 to begin observing the transaction rate.

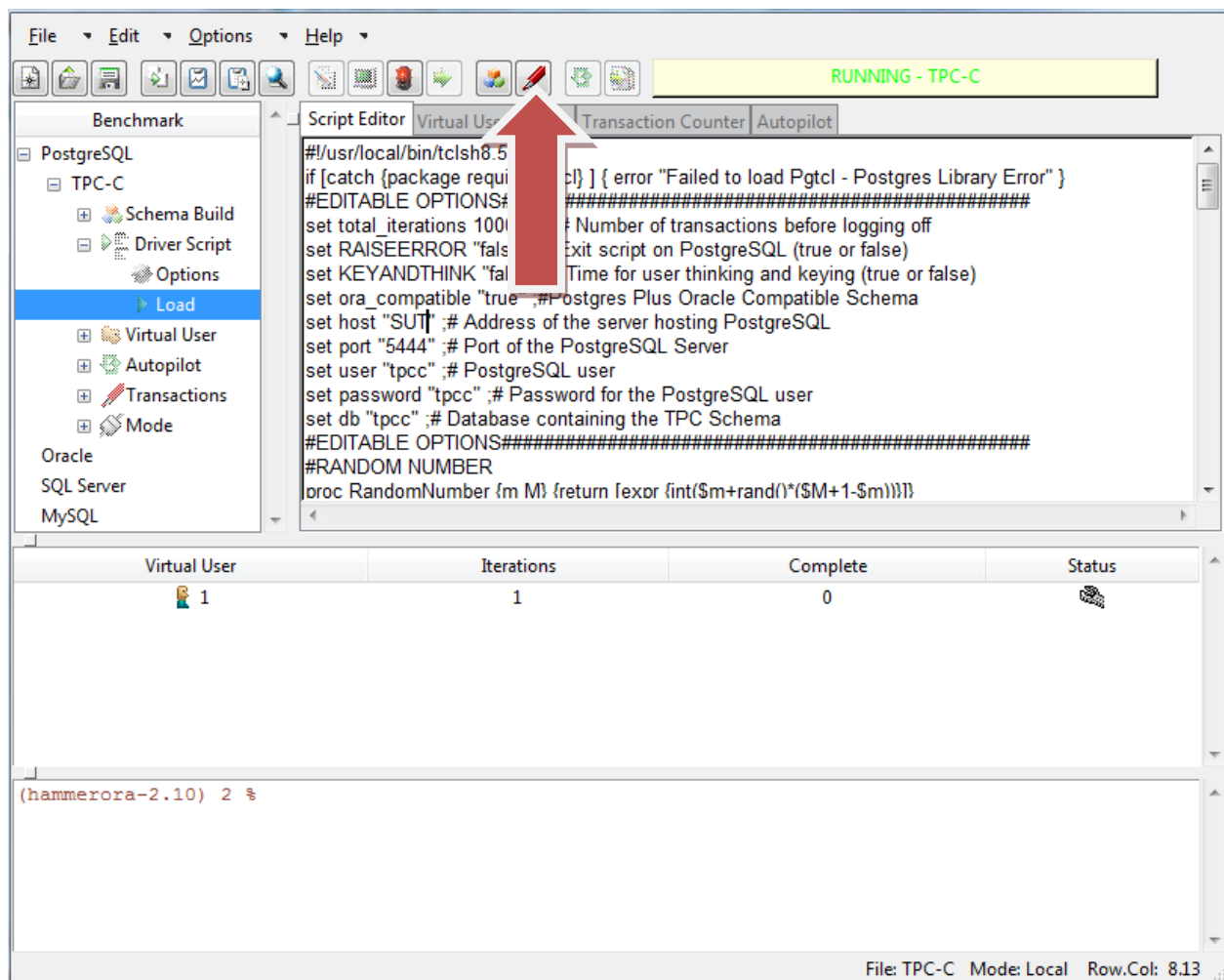


Figure 32 Start Transaction Counter

The transaction Counter will become active and start collecting throughput data as shown in figure 33.

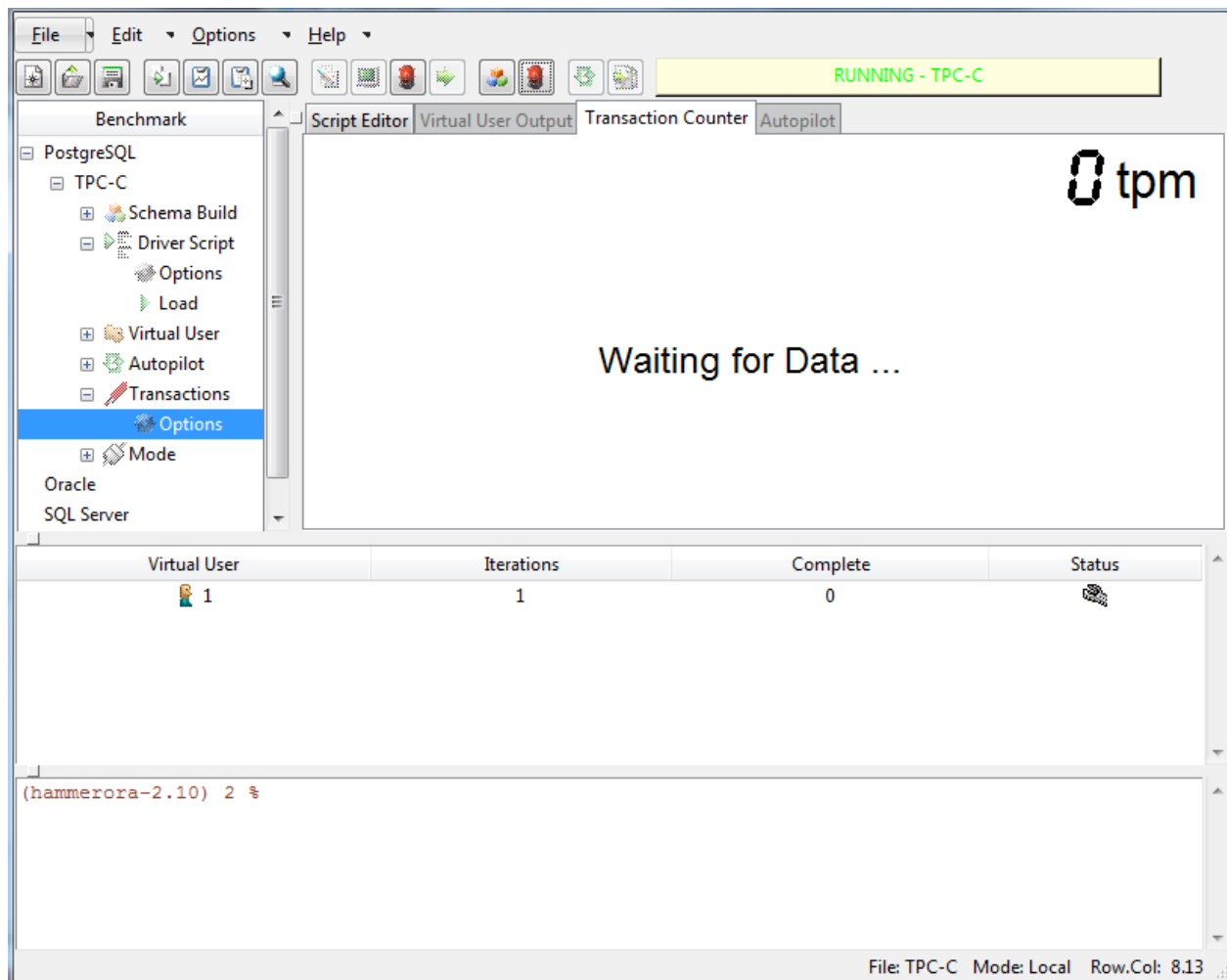


Figure 33 Waiting for Data

After the first refresh time interval you will be able to observe the transaction counter updating according to the throughput of your system. The actual throughput you observe for a single Virtual User will vary according to the capabilities of your system, however typically you should be looking for values in the low tens of thousands. Additionally once the transaction rate reaches a steady state you should observe the transaction counter maintaining a reasonably flat profile. Low transaction rates or excessive peaks and troughs in the transaction counter should be investigated for system bottlenecks on throughput.

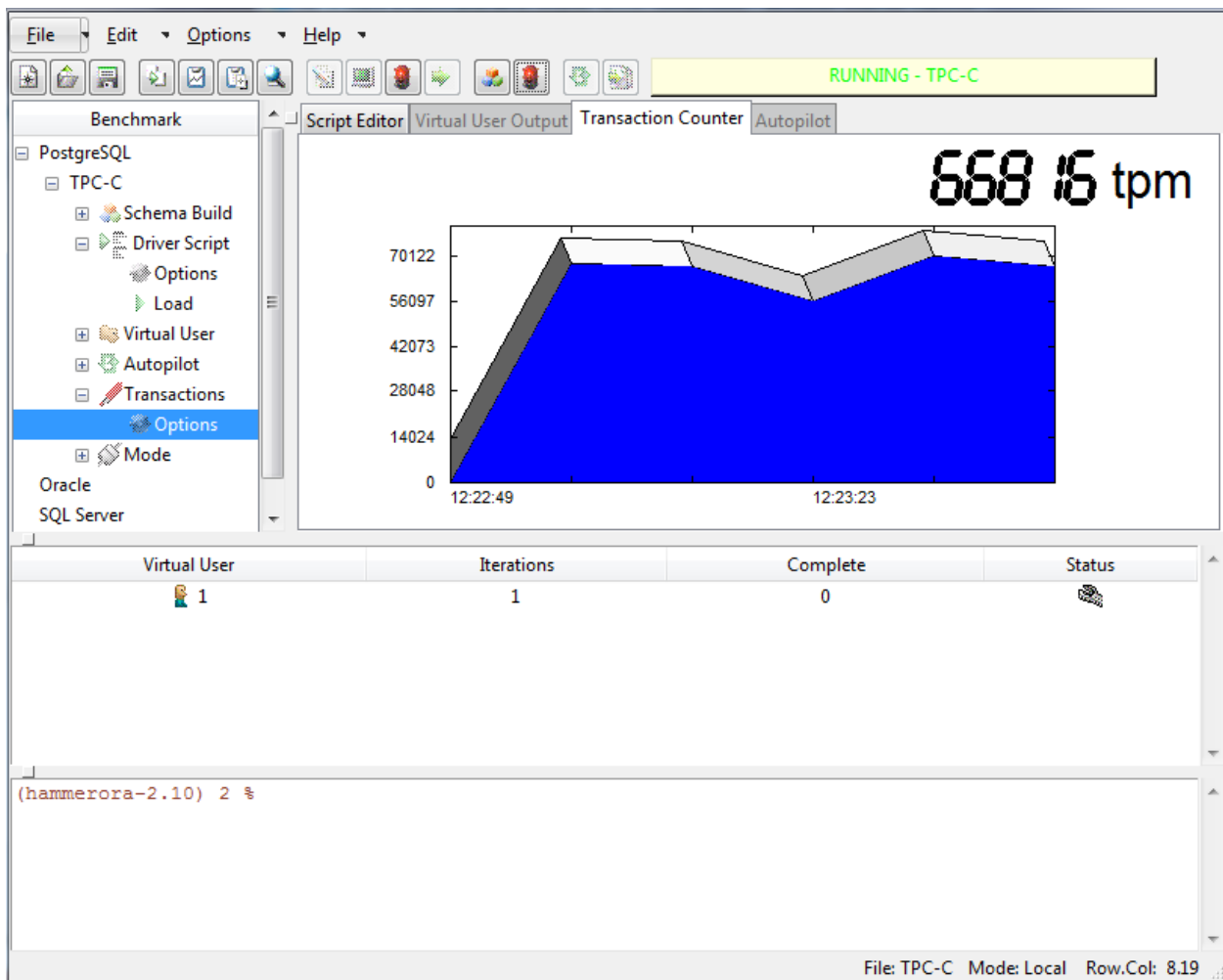


Figure 34 Virtual user Throughput

Once you are satisfied with the single Virtual User throughput close both the Transaction Counter and destroy the Virtual Users also stopping the test by pressing both Red Traffic Light icons. You should also proceed to pre-testing the throughput multiple Virtual Users. To do so repeat the testing you have done for a single Virtual User however instead increase the value for the number of Virtual Users to run the test in the Virtual User Options as shown in Figure 35.

The screenshot shows the "Virtual User Options" dialog box. It contains the following fields and options:

- Virtual Users: 2
- User Delay(ms): 500
- Repeat Delay(ms): 500
- Iterations: 1
- ☐ Show Output
- ☐ Log Output to Temp
- ☐ Use Unique Log Name
- ☐ No Log Buffer
- OK button
- Cancel button

Figure 35 Configuring Multiple Virtual Users

Similarly monitor the throughput for a higher number of Virtual Users as shown in Figure 36.

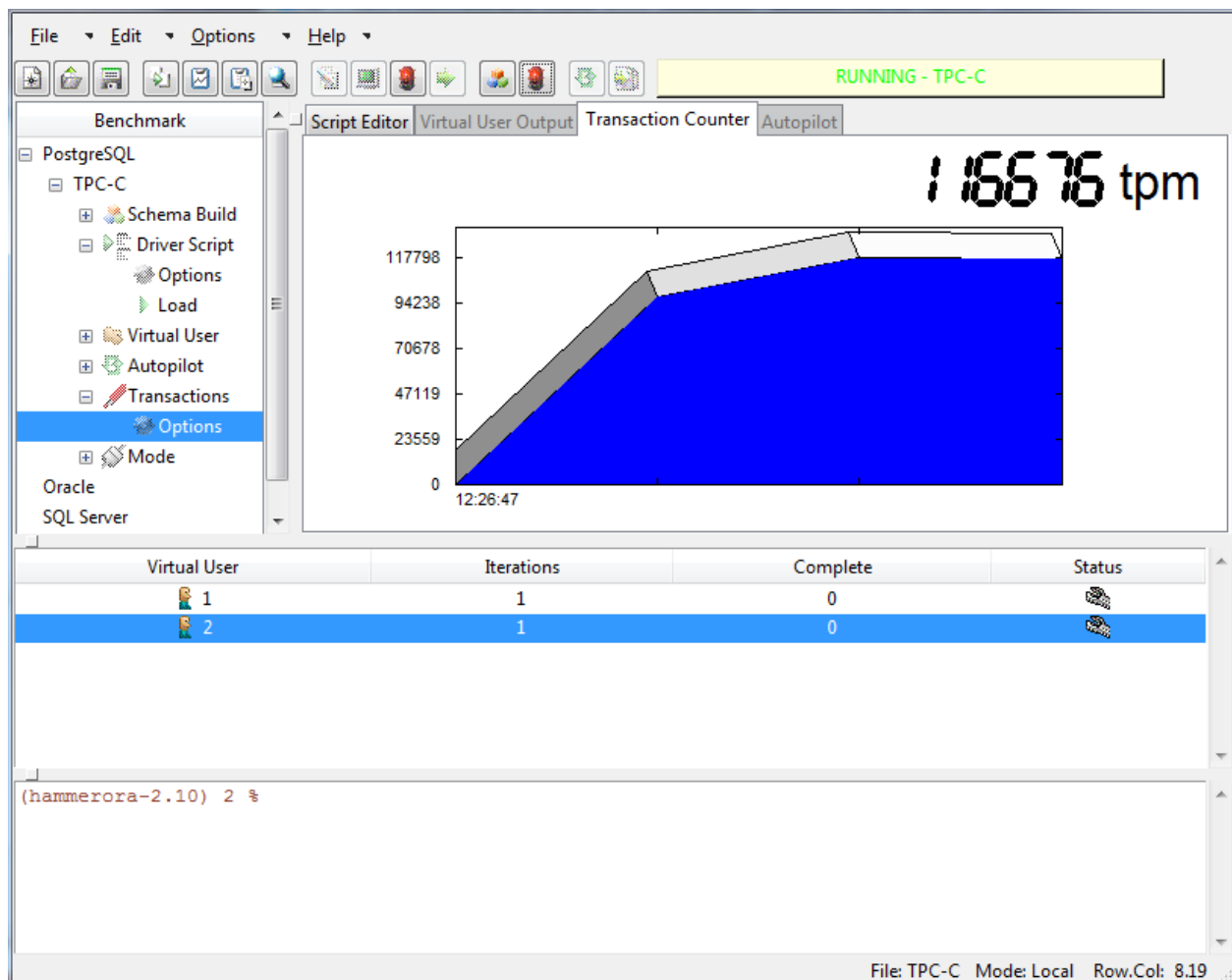


Figure 36 Running Multiple Virtual Users

Recommended Multiple Virtual Users for throughput testing are on an exponential scale from the single Virtual User test i.e. 2,4,8,16,32 Virtual Users should be tested up to double the number of Cores or Hyper-Threads on the SUT Database Server. If the observed transaction rate has numerous peaks and troughs or the consistent throughput is lower than expected you should examine the system configuration to diagnose the reasons why performance is limited. Once you have completed your pre-testing and are satisfied with your configuration you should move to planning and preparing to run a series of measured tests. You do not have to restart the database or rebuild the schema to conduct your performance tests. In fact having run a series of pre-tests and to have data resident in the buffer cache is the ideal starting point for conducting measured tests.

Planning and Preparation

Planning and Preparation is one of the most crucial stages of successful testing but is often overlooked. Firstly you should fully document the configuration of your entire load testing environment including details such as hardware, operating system versions and settings and PostgreSQL version and parameters. Once you have fully documented your configuration you should ensure that the configuration is not changed for an entire series of measured tests. This takes discipline but is an essential component of conducting accurate and measured tests. If you wish to change the configuration between tests to improve performance you should do so as part of the pre-test phase and not for the measured tests. If you change any aspect of the configuration you should conduct another full series of measured tests.

To plan your measured tests you should have a defined aim for what you wish to achieve and plan the tests accordingly. Often a test project can fail for having an unclear definition for the aim of what is desired to be

achieved. Typically this aim will take the form of determining the performance characteristics of a server (or server) however this can have many forms, for example generating a performance profile, determining the maximum throughput, measuring transaction response times or determining the maximum number of supported virtual users. The tests will vary according to the aim, for example it is relatively meaningless to use a test without keying and thinking to determine the maximum number of supported virtual users (because each virtual user can use the maximum performance of one core or thread), similarly enabling keying and thinking time is not applicable to determining a performance profile. Alternative testing aims can be to compare multiple configurations on the same platform, for example looking at the impact on throughput of Virtualization or changing OS and PostgreSQL parameters, the scope in this area for testing is limitless.

In this guide we will focus upon one of the most common testing scenarios, to generate a performance profile for server. This aim is used to identify for a given configuration of CPU, memory and I/O on a defined OS and PostgreSQL configuration the maximum number of transactions that the system can support. This is tested for a given number of virtual users, starting with one virtual user scaling up to the maximum number that the system can support. This approach ensures that the full capabilities of a multithreaded server are tested. With this approach we will define our Virtual Users without keying and thinking time. The number of cores/threads in this example on the SUT Database Server is 16, therefore we will prepare a simple tracking spreadsheet to record the results of our tests as shown in figure 37.

My Development Server				
Vusers	Run	DRITA	TPM	NOPM
1	1			
2	1			
3	1			
4	1			
8	1			
12	1			
16	1			
20	1			
24	1			
28	1			
32	1			

Figure 37 Planning Spreadsheet

With the configuration documented, the aim defined and a method to track the results of the tests prepared for our performance profile test project it is now possible to proceed to running timed tests with the Timed Test Driver Script.

Running Timed Tests with the Timed Test Driver Script

To run a timed and measured test there is an additional script to the Standard Driver Script called the Timed Test Driver Script that automates this functionality for you. With EnterpriseDB DRITA also provides functionality within the PostgreSQL Database to take snapshots of PostgreSQL performance metrics and report the change delta of these metrics taken between snapshots. Hammerora leverages DRITA to

automatically calculate the average transaction rate over a period of time and provide a comprehensive performance report for the test corresponding to that transaction rate. Note that the DRITA is a feature of EnterpriseDB Postgres Plus Advanced Server only. To select the Timed Test driver script, open the TPC-C Driver Options Window as described previously in this guide. Your focus is upon the Driver Options in this Window, and it is important to reiterate that you do not need to recreate the schema to modify the driver options or to change from using the Standard Driver Script to the Timed Test Driver Script or Vice Versa. Within the Driver Options shown in Figure 38, select the Timed Test Driver Script radio button.

Driver Options

PostgreSQL Host : SUT

PostgreSQL Port : 5444

PostgreSQL Superuser : enterisedb

PostgreSQL Superuser Password : postgres

PostgreSQL Default Database : edb

PostgreSQL User : tpcc

PostgreSQL User Password : tpcc

PostgreSQL Database : tpcc

EnterpriseDB Oracle Compatible : ☒

TPC-C Driver Script : ☐ Standard Driver Script ☒ Timed Test Driver Script

Total Transactions per User : 1000000

Exit on PostgreSQL Error : ☐

Keying and Thinking Time : ☐

Vacuum when complete : ☐

EnterpriseDB DRITA Snapshots : ☐

Minutes of Rampup Time : 2

Minutes for Test Duration : 5

OK Cancel

Figure 38 Timed Test Options

Once the Timed Test Driver Script is selected this activates the options to choose to Vacuum when complete, take EnterpriseDB DRITA snapshots and to select the Minutes of Rampup Time and Minutes for Test Duration as described previously in this guide. For a performance profile test you should plan to keep the Minutes of Rampup Time and the Minutes for Test Duration consistent for a number of tests with an increasing number of Virtual Users. For this reason you should plan to allocate sufficient rampup time for the higher number of Virtual Users at the end of your test sequence as well as the smaller number at the start. When you have selected your options click OK.

From under the Benchmark and TPC-C Menu select TPC-C Driver Script, this populates the Script Editor Window as shown in Figure 39 with the Timed Test Driver Script configured with your chosen options.

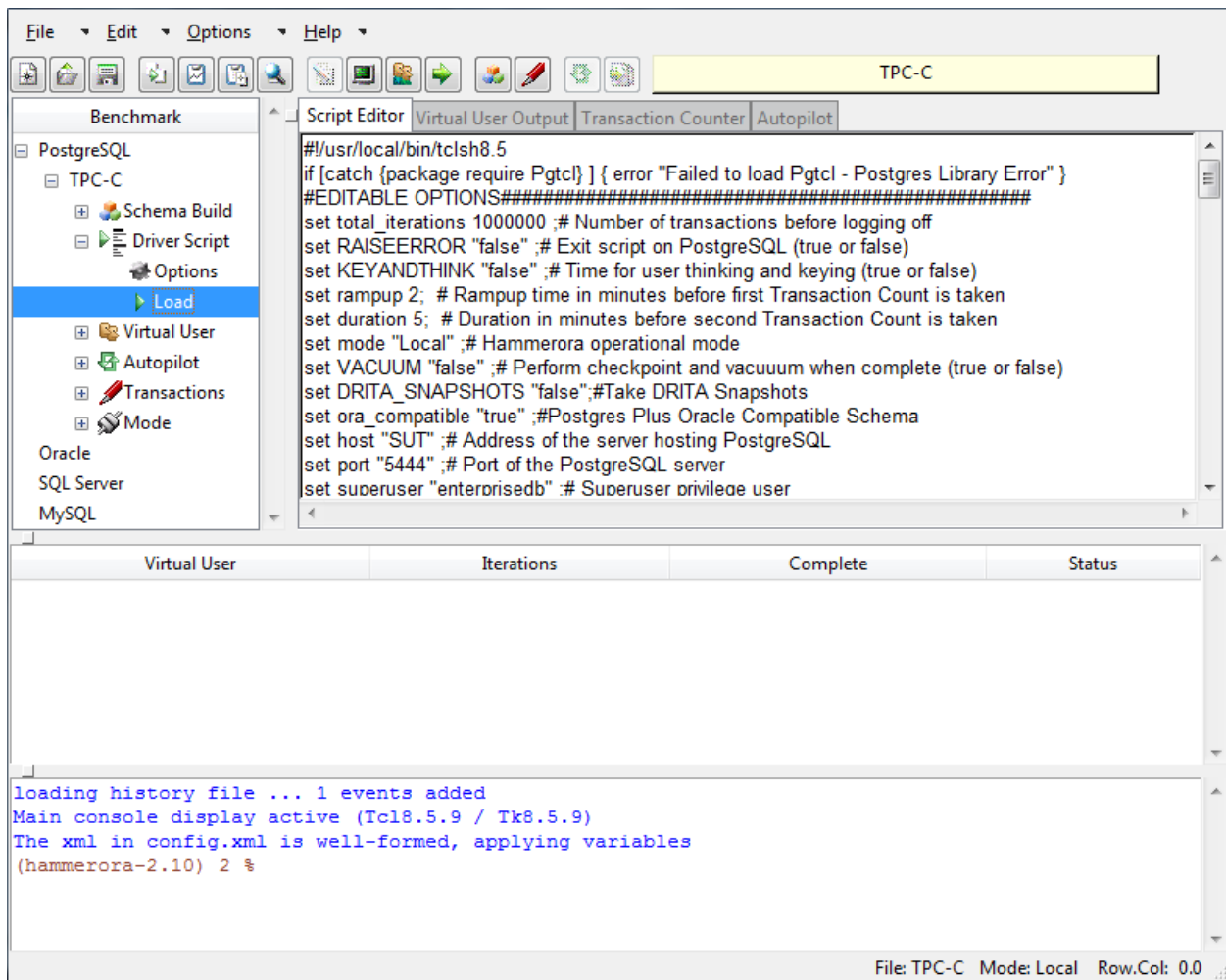


Figure 39 Timed Test Driver Script

To change these options you can either change them in the Schema Options window and reload the driver script or more advanced users can also change them directly in the Driver Script itself. To run the Timed Test Driver Script you must configure the Virtual Users as you did with the Standard Driver Script however there are two notable differences to observe. Firstly when running the Timed Test Driver Script one Virtual user will not run the Driver Script workload, instead this one Virtual User will monitor the timing of the test, take the Timed Tests and return the results. For this reason you should configure your Virtual Users with a Virtual User + 1 approach. ie to measure the workload for 1 Virtual User you should configure 2 Virtual Users, to measure the workload for 2 virtual Users you should configure 3 and so on. Additionally the Timed Test Driver Script is designed to be run with the Virtual User output enabled in order that you can view the Output from the Virtual User doing the monitoring, consequently the output for the Virtual Users running the workload is suppressed. The Virtual User configuration for the first test will look as Figure 40.

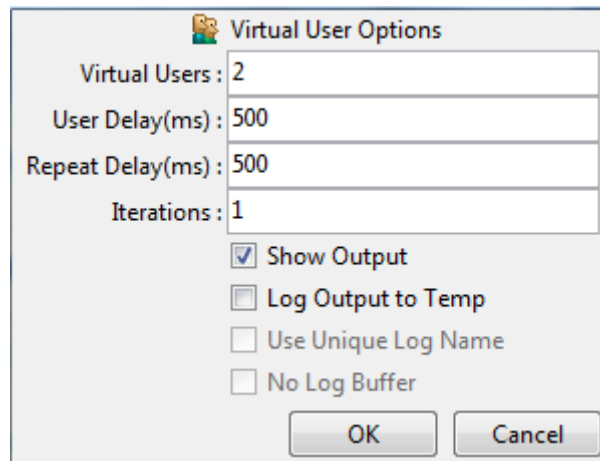


Figure 40 Timed Test Virtual Users

Click OK to save the configuration. Click the Create Virtual Users button as shown previously in this guide and Start the Virtual Users running. Note that the Virtual User output is now different as shown in Figure 41.

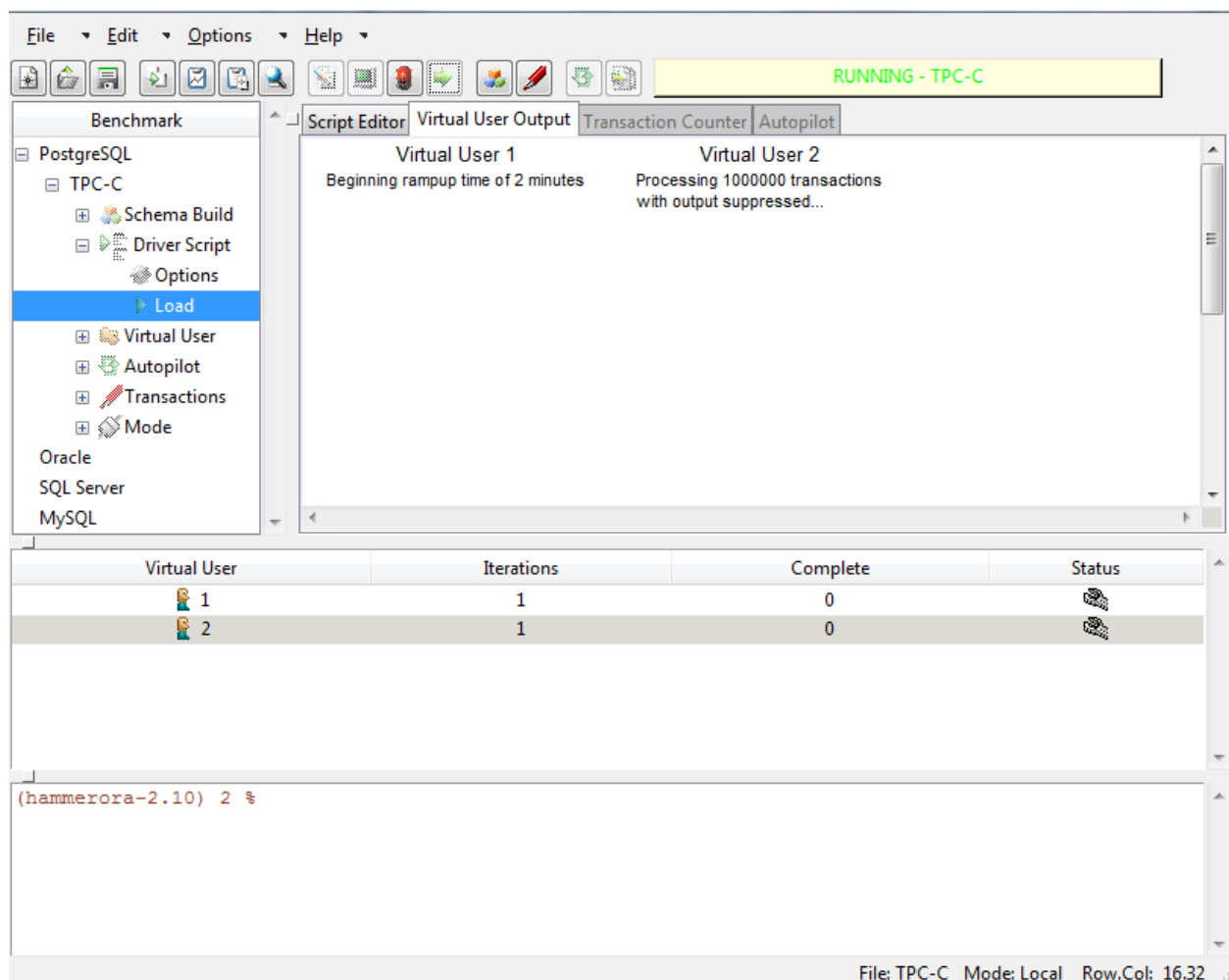


Figure 41 Timed Test Driver Script Running

The output shows that rather than reporting the outcome of every transaction the worker Virtual User in this example Virtual User 2 reports that it is processing transactions, however the output is suppressed. The Virtual User will print its message AFTER it has logged on and immediately BEFORE it runs its first

transaction. If this message has not been printed the session is still in the process of logging into the database. Increasing the User Delay(ms) value in the virtual user options can on some systems prevent a "login storm" and have all users logged on and processing transactions more quickly. Your rampup time should allow enough time for all of the users to be fully connected.

You will also be able to observe that in this example this single virtual User has logged on to the database and is running the workload. You can also observe that the monitor Virtual User, in this example Virtual User 1 is not running a workload but instead has logged on to measure the rampup time followed by taking the first Timed Test, measuring the timed test, taking the second Timed Test and reporting the outcome before logging off and ending the monitor script. It is worthwhile reiterating therefore that for the Timed Test Driver Script you need to configure and run n+1 Virtual Users with the additional Virtual User doing the monitoring and measuring. The sample output of this monitoring Virtual User is shown in figure 42.

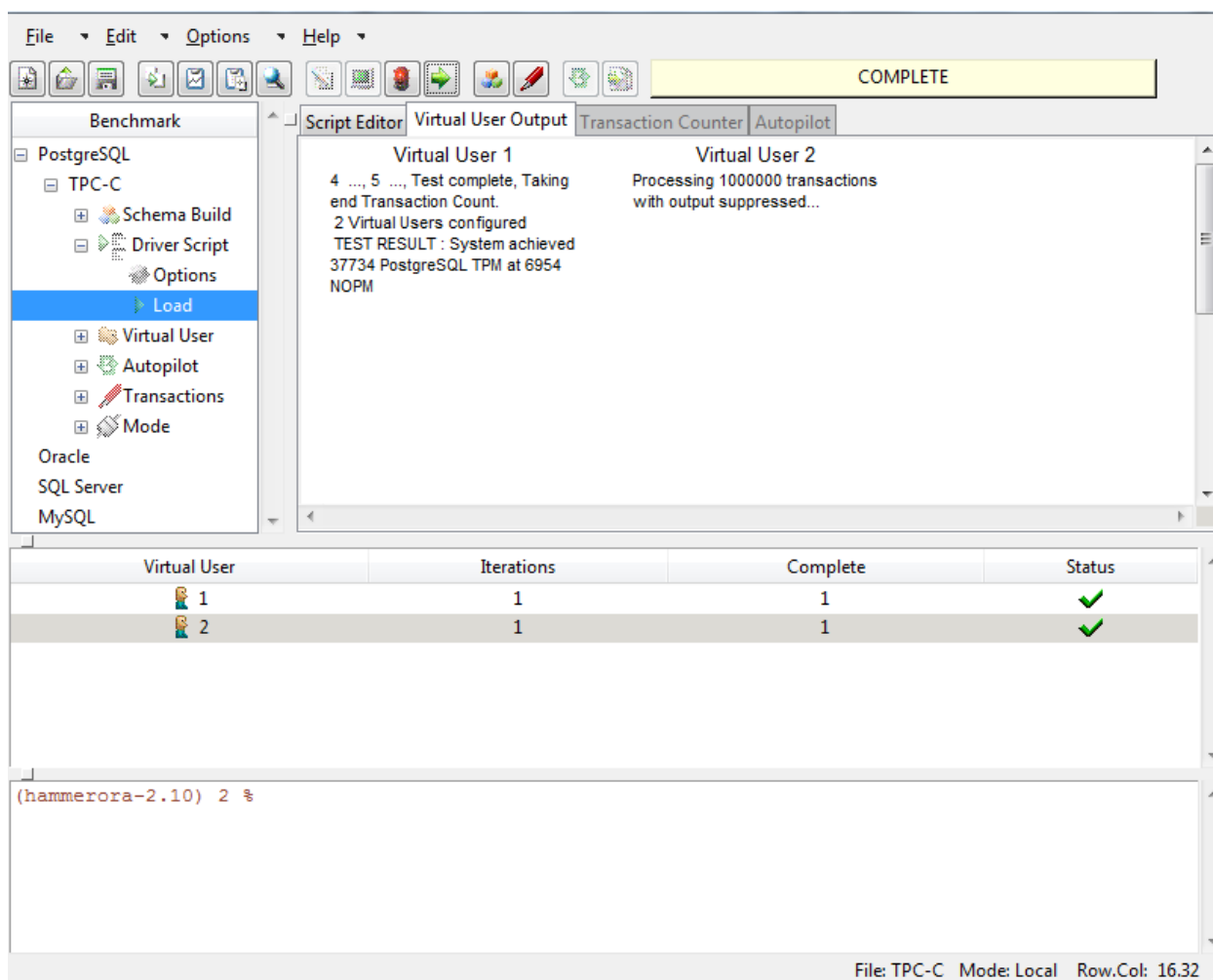


Figure 42 Timed Test Result

The monitoring user reports the TEST RESULT of TPM and NOPM. TPM measures the number of PostgreSQL Transactions per minute and is not to be confused with the tpmC value from an official TPC-C benchmark. NOPM reports the number of New Orders per minute and is used as a database independent statistic. Consequently for example TPM cannot be used to compare the performance results of PostgreSQL with Oracle, SQL Server or MySQL but NOPM can.

In addition to the test report if the DRITA option has been chosen the monitoring user also reports the SNAPIDs that can be used to generate a DRITA performance report for the workload. If you have not chosen the Checkpoint when Complete option you should manually press the red traffic light icon to stop the Virtual User workload. You may if you wish also run the Transaction Counter during an Timed Test Driver

Script test. When you have stopped the test enter your data into your reporting spreadsheet as shown in Figure 43.

My Development Server				
Vusers	Run	DRITA	TPM	NOPM
1	1		37734	6954
2	1			
3	1			
4	1			
8	1			
12	1			
16	1			
20	1			
24	1			
28	1			
32	1			

Figure 43 Updated Reporting Spreadsheet

Note that the Run column enables you to record multiple tests for the same number of Virtual Users. You should run two or three tests at the same number of Virtual Users and take the average value of all of the tests as your final value for the workload. Your aim should be for consistent performance as shown in Figure 44.

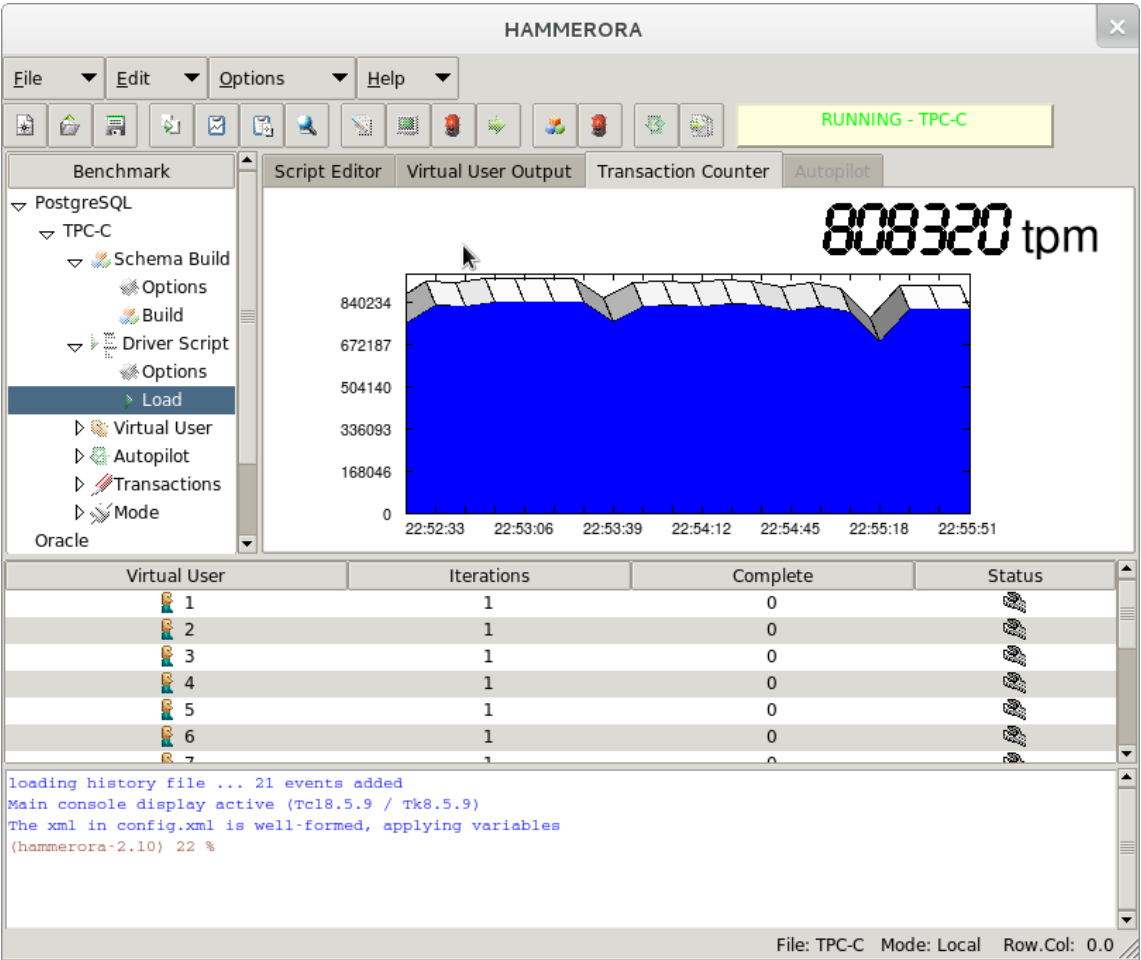


Figure 44 Consistent Performance

With the test complete and the values you recorded if you selected the DRITA option you should next generate the DRITA report that corresponds to the reported SNAPIDs to show the PostgreSQL wait events, in the example below snapshots 19 and 20.

```
SQL> select * from sys_rpt(19,20,1000);
```

SYS_RPT

WAIT NAME		COUNT
WAIT TIME	% WAIT	
proc array lock acquire		99548
132520043050237	100.00	
wal write		182176
529.963593	0.00	
wal flush		152316
528.913145	0.00	
wal insert lock acquire		599941
480.952203	0.00	
wal file sync		83990
32.792558	0.00	
db file extend		100566
2.353239	0.00	
db file read		153642
1.756448	0.00	
sinval lock acquire		92891
1.028955	0.00	
query plan		8016
0.196485	0.00	
clog control lock acquire		2831
0.171418	0.00	
db file write		5833

```

0.105992      0.00

buffer free list lock acquire      1355
0.066343      0.00

bgwriter communication lock acquire      83
0.002002      0.00

15 rows retrieved.

```

Once you are satisfied with the test results, repeat the test with the next value in the number of Virtual Users in your sequence remembering to add one for the monitor thread. Once this test is complete either repeat the process with the next value in the sequence or automate your testing with autopilot mode as detailed in the following section. With either method do this until you have completed your spreadsheet with all of the desired values for database performance.

Automating Tests with Autopilot Mode

If you prefer to run all of your tests manually you do not need to use the Autopilot Mode. However if you wish to run your entire sequence of tests unattended then Autopilot Mode enables you to use your time most productively. It can help to understand Autopilot Mode as a feature that simulates the presence of a DBA instructed to run your desired sequence of tests at specified time intervals and report the entire results of all tests in one batch. To begin configuring Autopilot mode follow the steps described in the previous section for Running Timed Tests with the Timed Test Driver Script up to the steps illustrated in Figures 39 and 40. You only need to configure the correct driver script but not configure the Virtual Users, they will be configured automatically. To do this select Autopilot Options from the either the Options menu or the treeview as shown in Figure 45.

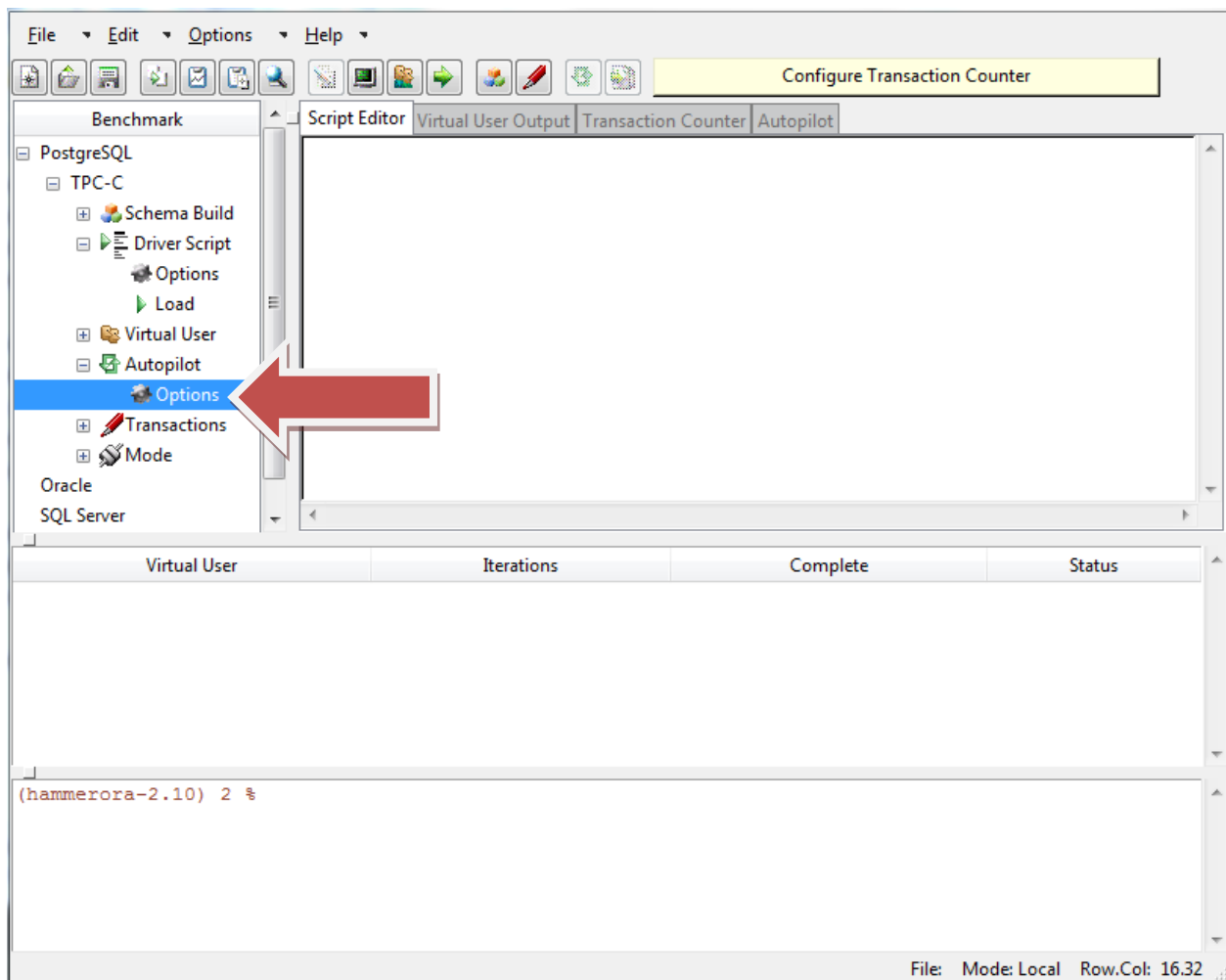


Figure 3 Autopilot menu

This shows the Autopilot Options menu as shown in Figure 46.

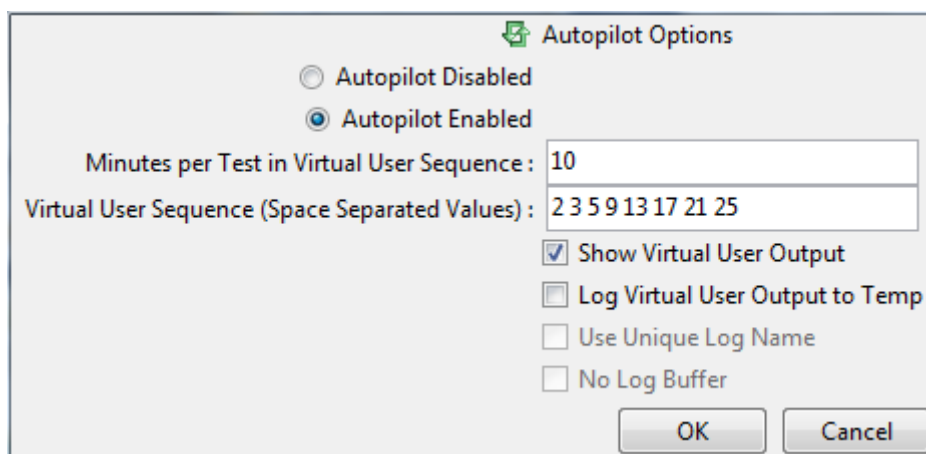


Figure 4 Autopilot Options

Configure the Autopilot options precisely in the same manner as you would use to instruct your Virtual DBA as follows:

Autopilot Disabled/Autopilot Enabled

This Autopilot Disabled/Autopilot Enabled Radio buttons give you the option to select whether the Autopilot button is enabled on the main window.

Minutes per Test in Virtual User Sequence

The minutes for test duration defines the time interval between which your virtual DBA will create the Virtual Users, stop the test and create the next Virtual Users in the sequence. You should configure this value in relation to the Minutes for Ramup Time and Minutes for Test Duration given in the Timed Test options shown in Figure 39. For example if the values in the test script are 2 and 5 minutes respectively then 10 minutes for the Autopilot Options is a good value to allow the test to complete before the next test in the sequence is run. If the test overruns the time interval and the Virtual Users are still running the sequence will wait for the Virtual Users to complete before proceeding however note any pending output will be discarded and therefore for example if the TPM and NOPM values have not been reported by the time the test is stopped they will not be reported at all.

Virtual User Sequence (Space Separated Values)

The Virtual User Sequence defines the number of Virtual Users to be configured in order for a sequence of tests separated by the Minutes for Test Duration. For example as shown in Figure 46, firstly a test with 2 Virtual Users will be run, then after 10 minutes a test with 3 Virtual Users will be run, then 5 Virtual Users and so on to the end of the sequence. Note that the default Values are given as odd numbers to account for the Monitoring Virtual User when running the Timed Test Driver Script. Therefore in this example the actual Users running the workload will be 1, 2, 4, 8, 12, 16, 20 and 24.

Show Virtual User Output/Log Virtual User Output to Temp/Use Unique Log Name/no Log Buffer

These values are exactly the same as set when defining the Virtual Users, the Autopilot Options gives you the opportunity to set them when configuring Autopilot Mode to ensure that you have a permanent record of the output of the tests that you run.

TIP: When running the Timed Test Driver Script in Autopilot Mode it is recommended to enable Vacuum when complete in the Timed Test options as part of the Driver Script options.

Once your Autopilot Options are defined, press OK to save the values. Close down all running virtual Users and the transaction counter and press the Autopilot button as shown in Figure 47.

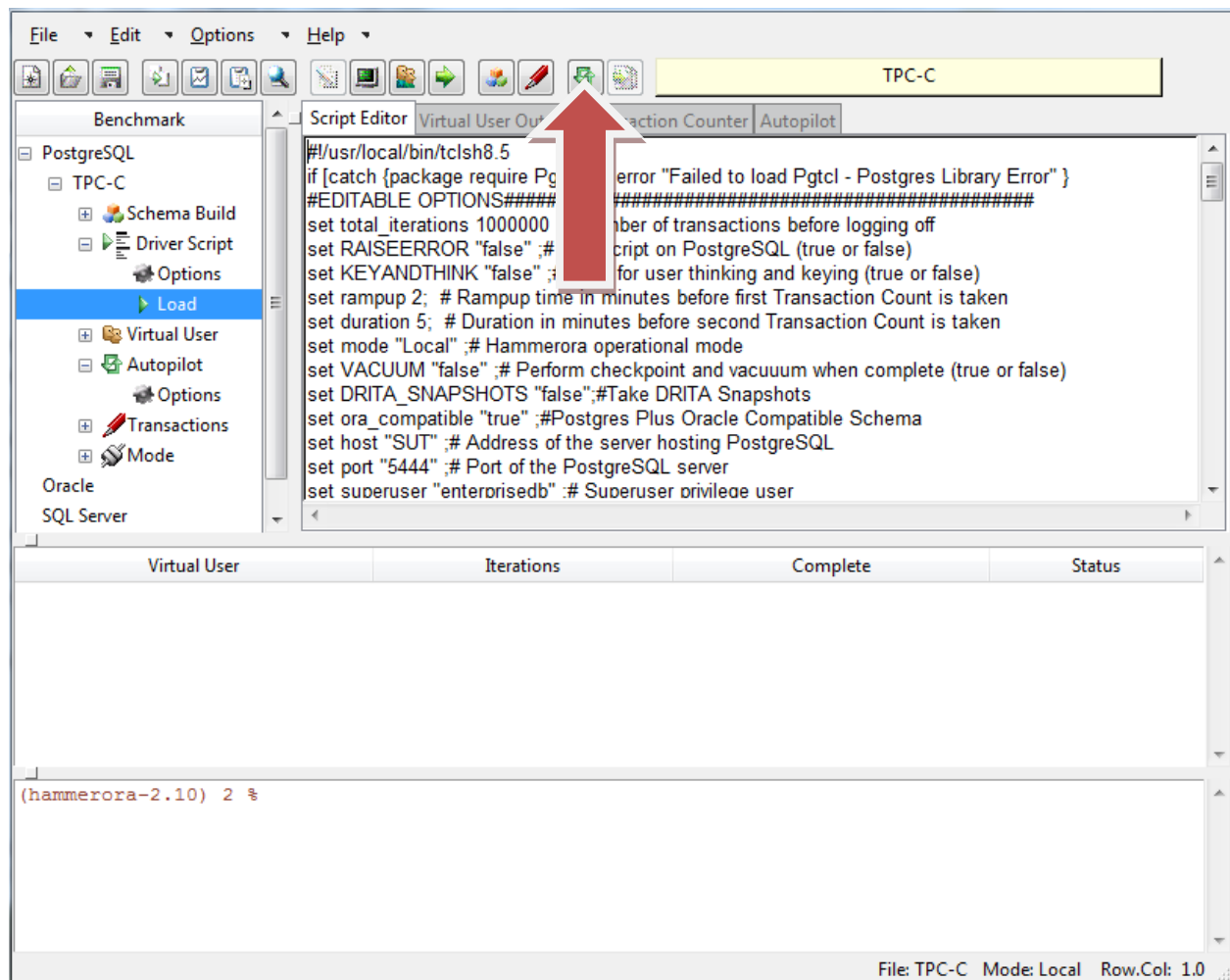


Figure 5 Start Autopilot

You can now leave the autopilot mode to run your chosen sequence of tests without any further intervention. The Autopilot screen as shown in Figure 48 becomes active and reports your progress. In particular note the timer in the top right hand corner tracking the interval times at which your tests should be run.

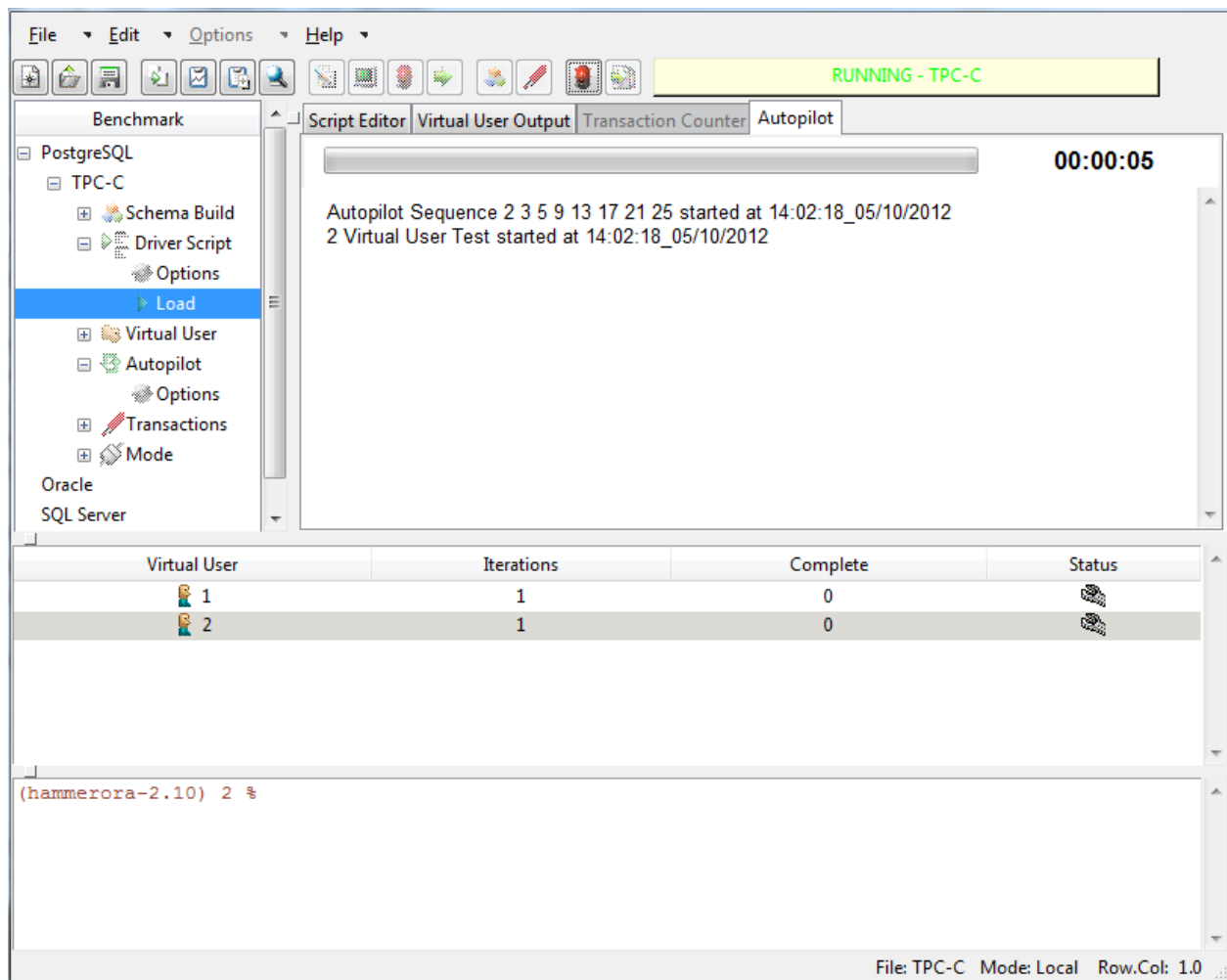


Figure 6 Autopilot Screen

The Autopilot will continue to run through your chosen sequence, creating virtual users and running the test in the test script as shown in Figure 49.

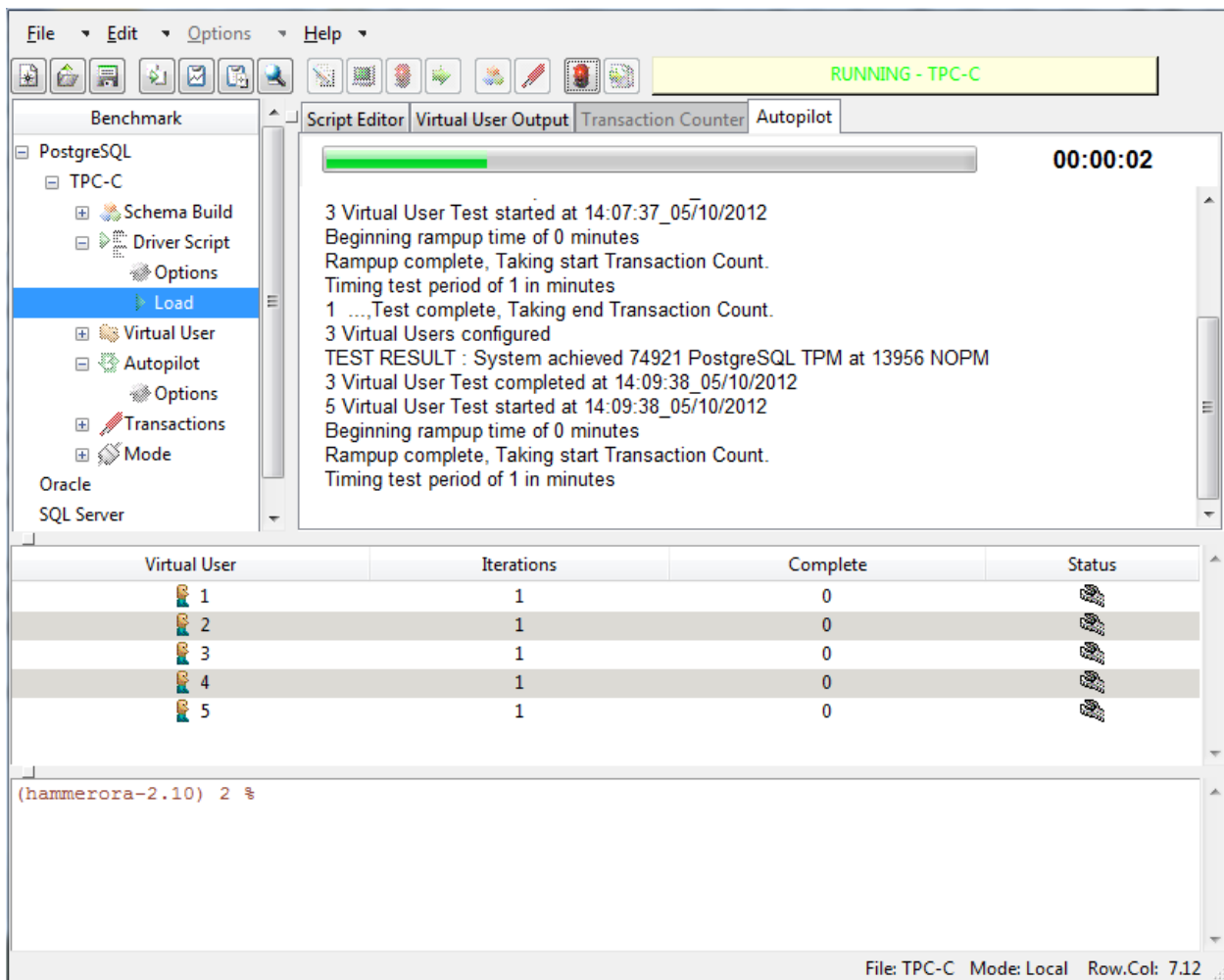


Figure 7 Autopilot Continuing

The output of the first virtual user will also be reported to the autopilot output screen, additionally all output will be reported to the log file if you have chosen to write the output to one. If you have not chosen to use the Hammerora log file and have closed the autopilot output then it is necessary to use the data Timed Tests to report the performance recorded during your tests.

When you have finished your test sequence press the traffic light icon to end Autopilot Mode. This data is available for all of the tests you performed allowing you to collect all of your results and DRITA reports at a single point in time after all tests are complete.

You should now be equipped with all of the performance data you need to begin your analysis.

Performance Analysis

When you have finished your sequence of tests you will be equipped with a spreadsheet containing a number of data points and optionally a DRITA report corresponding to each data point.

Performance Comparisons

Use your spreadsheet to generate a graph of a performance profile with the TPM value for the y axis and the number of Virtual Users for the x axis. The performance profile should resemble the figure as shown in Figure 50 with an increasing level of transactions as the number of Virtual Users increases up to a maximum point of system utilisation. (Note that the actual values of virtual users and transactions per minute have

been removed from this example as it is the relative performance that is important as opposed to the absolute values.)

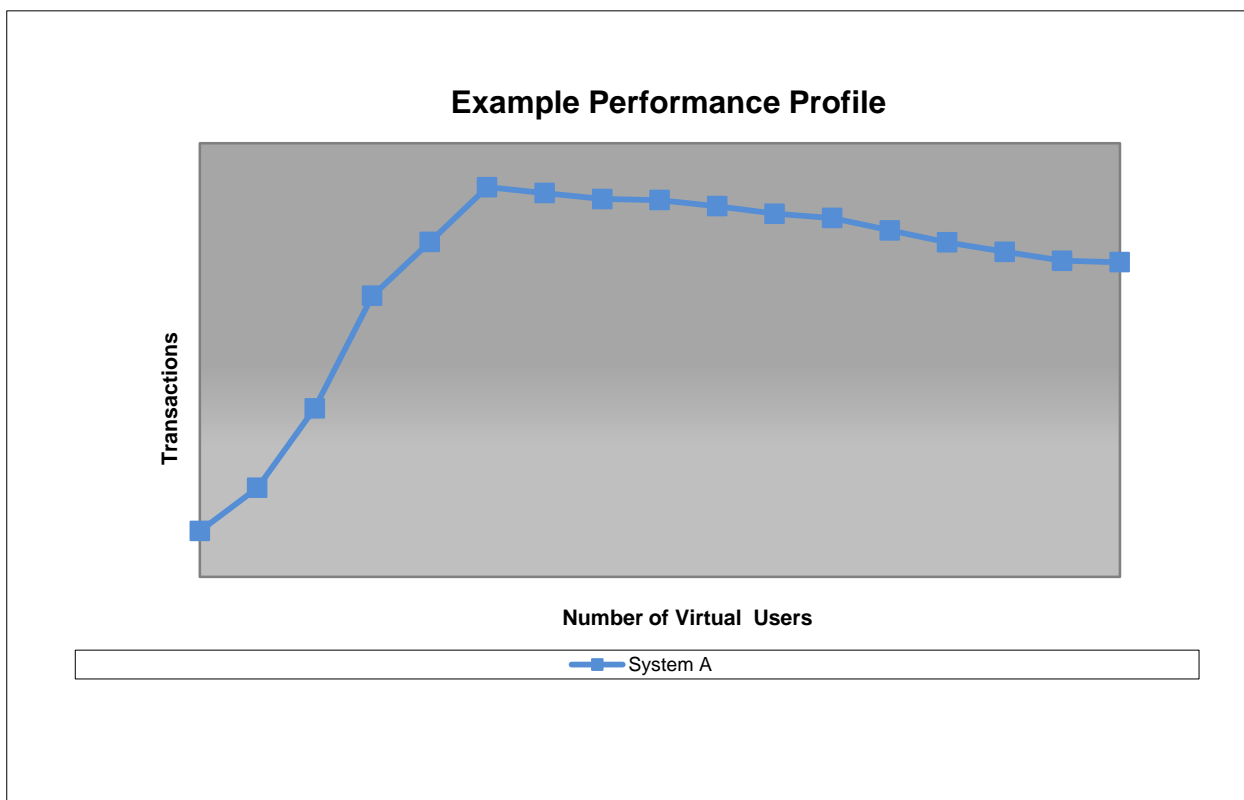


Figure 50 System A performance profile

TIP: The data produced for an official TPC-C benchmark is the logical equivalent of discarding all of the data points and keeping the top value only. Here you have produced significantly more data regarding the performance of the system across all levels of utilisation.

When you have data for multiple systems you can now add the performance data to the same spreadsheet and use the resultant graph to show a comparison.

Conclusion

You should now be equipped to perform fully scalable transactional workloads against a PostgreSQL Database environment. An experienced PostgreSQL Database tester will be able to go from bare metal to test in one working day to install a system, create the schema and start the automated test collecting the results the following working day making a 2 day project time possible to determine a full system performance profile. Do not forget that Hammerora is Open Source and you have the ability to modify the test scripts to fit whatever purpose meets your needs for testing PostgreSQL transactional performance.

Support and Discussion

Need help? Try the Hammerora Sourceforge forum here :

<http://sourceforge.net/projects/hammerora/forums/forum/292313>

Want to discuss your results or have tips on tuning and configuration? open a Discussion topic here:

<http://sourceforge.net/projects/hammerora/forums/forum/292312/topic/3848751>