



**Hewlett Packard**  
Enterprise

# Microsoft SQL Server on Linux: Getting Started

Hands on Lab

# Table of Contents

<b>Introduction.....</b>	<b>3</b>
References: .....	3
<b>Description of the environment.....</b>	<b>3</b>
Your environment .....	3
Access your server: .....	4
<b>Exercise 1- ODBC CLI installation .....</b>	<b>5</b>
<b>Ex 2 – SQL Server Docker image .....</b>	<b>6</b>
Installations.....	6
Database restoration .....	7
Basic operations: .....	8
Access MSSQL for Linux from MS SSMS .....	9
Next exercise preparation.....	9
<b>Exercise 3 – MSSQL Server and YUM.....</b>	<b>11</b>
Installation .....	11
Database restoration preparation .....	12
Database stress with log and data files on disk .....	13
<b>Exercise 4 - MSSQL and Persistent memory .....</b>	<b>17</b>
Persistent Memory preparation.....	17
Database restoration with log and data files on PMEMs .....	19
Database stress with log and data files on PMEMs .....	19
<b>Exercise 5 - NVDIMM persistency .....</b>	<b>21</b>
<b>Ex 6: Use of HammerDB.....</b>	<b>23</b>
<b>Ex7 - In Memory OLTP with HammerDB .....</b>	<b>27</b>
<b>EX 8: Create a Python application using SQL Server on Linux.....</b>	<b>30</b>
Columnstore indexes .....	31
<b>End of the lab.....</b>	<b>32</b>

# Introduction

Microsoft is introducing its SQL Server database product on various Linux distributions. This performant database runs already perfectly on HPE servers including our Mission Critical servers, with the Windows Operating Environment. The goal of this lab is to get a first experience with it on a Red Hat Enterprise system.

This lab intends to get familiar with a basic setup and management operations necessary to leverage the most of MS SQL Server for Linux on HPE servers. Although it contains some redundancy content with Microsoft's documentation, several aspects are unique like use cases involving HPE 8GB NVDIMMs.

You will use a Preview version of MS SQL Server for Linux with a free license fee. When

## References:

[SQL Platform Abstraction Layer \(PAL\)](#)

[Build an app using SQL Server](#) (C#, Java, Node.js, PHP, Python, R)

[Download](#) the Public Preview of SQL Server for Linux

[Join](#) the SQL Server vNext Early Adoption Program (SQL EAP)

# Description of the environment

In this lab, and once you have connected to your Teams Jump-Server, you will have to access a **ProLiant Gen9** server with RHEL 7 installed on an internal disk. You will have privileged access to the operating system.

## Your environment

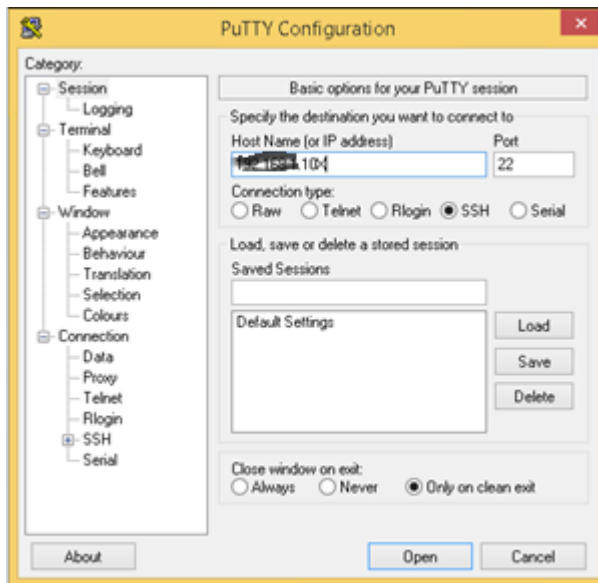
Ask the instructor for your team name/number.

**Team names/numbers go from lab01 to lab12.**

If **xx** is your lab/Team number, then, your corresponding server address will be: **192.168.1.2xx**

## Access your server:

From your Jump station **Open** a **PuTTY** or **Cygwin/ssh** session toward your server at **192.168.1.2XX** - where **XX** is your lab/Team number.



Answer **yes** to the Security Alert (if needed) and login as **root** / **password**:

# Exercise 1- ODBC CLI installation

In this first exercise, we will install and configure the yum repo file for this infra and then install the `sqlcmd` and `bcp` command line interfaces:

```
Host# cd /etc/yum.repos.d
Host# curl https://packages.microsoft.com/config/rhel/7/prod.repo > \
      sql-tools.repo
Host# echo -e "proxy=_none_" >> sql-tools.repo
Host# tail -2 sql-tools.repo
gpgkey=https://packages.microsoft.com/keys/microsoft.asc
proxy=_none_
Host# yum update
Host# yum -y install mssql-tools
...
Do you accept the license terms? (Enter YES or NO)
YES
...
Host# ln -sf /opt/mssql-tools/bin/sqlcmd /usr/bin/sqlcmd
Host# ln -sf /opt/mssql-tools/bin/bcp /usr/bin/bcp
```

# Ex 2 – SQL Server Docker image

SQL Server for Linux is proposed with two packaging: a [Docker image](#) and in a YUM repository. This exercise focuses on the Docker image; you will install first the Docker engine and then the MSSQL Docker image and restore a database on a persistent volume. More information can be found on the documentation Microsoft site.

## Installations

Install the Docker engine and start the service. Although we are running a RHEL distro, we have to install the CentOS Docker engine because we have no valid Red Hat subscription for this lab infra. Don't do that in a production environment.

```
Host# yum install docker-engine -y
...
Host# systemctl start docker.service
Host# docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
...
Hello from Docker!
This message shows that your installation appears to be working correctly.
...
```

Pull the MSSQL Docker image from the Internet:

```
Host# docker pull microsoft/mssql-server-linux
Using default tag: latest
latest: Pulling from microsoft/mssql-server-linux
...
Status: Downloaded newer image for microsoft/mssql-server-linux:latest
Host#
```

The following command runs the SQL Server container on TCP port 1433, and sets two environment variables: the End User License Agreement (**EULA**) and the System Administrator (**SA**) password.

Moreover, the **-v** option, maps the server's directory `/var/opt/mssql` with the container's `/var/opt/mssql` directory. Hence anything present in this directory will be seen by both the server and the container.

The **-d** option (detach) runs the container in background.

The following commands respectively verifies that the container is up and running and saves its UUID in a variable for future use. If the password is not strong enough, the container will not run and the `docker ps` command will not return the expected output:

```
Host# docker run -e "ACCEPT_EULA=Y" -e "SA_PASSWORD=P@ssw0rd" -p 1433:1433 \
-v /var/opt/mssql:/var/opt/mssql \
-d microsoft/mssql-server-linux:latest

Host# docker ps
...
Host# C_ID=$(docker ps --format "{{.ID}}")
Host# echo $C_ID
```

## Database restoration

The SQL Server is now up and running in its Docker container. Before restoring the [AdventureWorks database](#) we need to `unzip` the backup file in the volume's container. The `unzip` command lasts approximately 2 minutes because the database contains an extra big table required later:

```
Host# cd /usr/kits/SQL-Linux
Host# unzip AdventureWorks.zip -d /var/opt/mssql
Archive:  AdventureWorks2014.zip
  inflating: /var/opt/mssql/AdventureWorks2014.bak
```

To save time and avoid typos, create a bash alias for **sqlcmd** and its arguments to access the local database as the System Administrator (SA) user:

```
Host# alias sql='sqlcmd -S. -U SA -P "P@ssw0rd"'
```

For your convenience and to save time and typos again, the `RestoreAdventureDB.sql` transactional script performs the restore operation. Review the content of this script:

```
Host# cat RestoreAdventureDB.sql
-- RestoreAdventureDB.sql
-- Version 0.7

-- Restores the AdventureWorks DB to default/usual locations

USE [master]
RESTORE DATABASE AdventureWorks
FROM DISK = '/var/opt/mssql/AdventureWorks.bak'
WITH MOVE 'AdventureWorks2014_Data' TO '/var/opt/mssql/data/AdventureWorks2014_Data.mdf',
MOVE 'AdventureWorks2014_Log' TO '/var/opt/mssql/log/AdventureWorks2014_Log.ldf'
GO
```

The restore operation is a standard CLI command with `sqlcmd`. If you omit the password argument, `sqlcmd` will ask you to enter it in a silent mode. The following command lasts about 3 minutes:

```
Host# sql -i RestoreAdventureDB.sql
Changed database context to 'master'.
Processed 1062560 pages for database 'AdventureWorks', file 'AdventureWorks2014_Data'
on file 1.
...
RESTORE DATABASE successfully processed 1062567 pages in 107.304 seconds (87.233
MB/sec).
```

## Basic operations:

To test whether the restore is successful, you can perform various operations provided in the `SimpleQuery.sql` and `UpdateRecord.sql` files. Review their content and run them:

```
Host# cat SimpleQuery.sql
-- SimpleQuery.sql
-- Version 0.1

USE AdventureWorks;
GO
SELECT Name, GroupName
FROM HumanResources.Department
GO
```

```
Host# sql -i SimpleQuery.sql
Changed database context to 'AdventureWorks'.
Name                                     GroupName
-----
Engineering                             Research and Development
...
(16 rows affected)
```

```
Host# more UpdateRecord.sql
-- UpdateRecord.sql
-- Version 0.1

USE AdventureWorks;
GO

Update Production.WorkOrder
SET ScrappedQty = 1
WHERE WorkOrderID = 70370
```

```
Host# sql -i UpdateRecord.sql
Changed database context to 'AdventureWorks'.

(1 rows affected)
```

Several other T-SQL scripts are present in this directory. Feel free to test them and create others if you want.

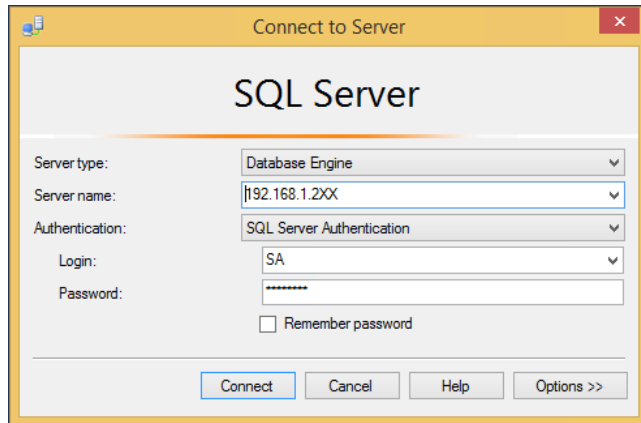


## Access MSSQL for Linux from MS SSMS

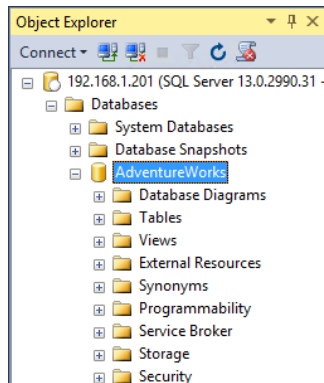


Start the Microsoft SQL Server Management Studio by clicking on its icon in the **taskbar**:

Connect to the Server by entering your server IP address, the **SA** username and **P@ssw0rd** as the password:



Once connected in the left pane of the Studio, expand your SQL Server and then the Database folder. You should see the AdventureWorks database in the list:

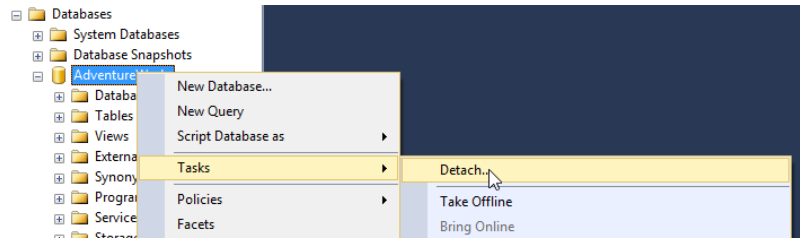


Feel free to browse the studio and perform actions in the database.

## Next exercise preparation

In the next exercise, we will install MSSQL Server differently. Hence we need to clean-up the Docker installation to avoid any network port contention.

Using the Studio, **Detach** the database: **Right click** on the AdventureWorks DB then expand **Tasks** and click on **Detach...**:

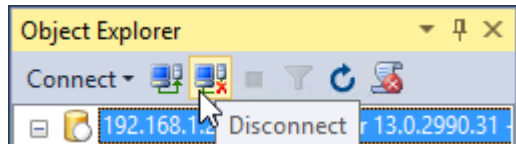


Highlight the AdventureWorks line, check it is **ready** to be detached:

Databases to detach:				
Database Name	Drop ...	Updat...	Status	Message
AdventureWorks	<input type="checkbox"/>	<input type="checkbox"/>	Ready	

If there are active connections, check the **Drop Connections** box and click **OK**.

**Disconnect** the Studio from the server: Highlight your server and then click on the Disconnect icon:



Minimize the Studio and, from the PuTTY/SSH session, **stop** and **remove** the Docker container, including the database files:

```
Host# docker stop $C_ID
Host# docker rm $C_ID
Host# systemctl stop docker.service

Host# rm -r /var/opt/mssql/{data,log,etc,secrets}
```

## Exercise 3 – MSSQL Server and YUM

The second packaging offered is **YUM** based. In this exercise we download the product from the Internet, install it, configure it and start it.


Then, we will stress the database with the logs on an internal disk first and secondly on a persistent memory location (NVDIMMs).

The goal of this exercise is to show the impact on the performance when the logs are located on a slow or fast device.

### Installation

We need first to download a YUM repo and configure it for this infra and then install the **mssql-server** package:

```
Host# cd /etc/yum.repos.d
Host# curl https://packages.microsoft.com/config/rhel/7/mssql-server.repo \
> mssql-server.repo
Host# echo -e "\nproxy=_none_" >> mssql-server.repo
Host# tail -2 mssql-server.repo
gpgkey=https://packages.microsoft.com/keys/microsoft.asc
proxy=_none_
Host# yum install mssql-server -y
```



```
Installing : mssql-server-14.0.1.246-6.x86_64

+-----+
| Please run /opt/mssql/bin/sqlservr-setup to complete the setup of |
|                               Microsoft(R) SQL Server(R) .         |
+-----+

Verifying : mssql-server-14.0.1.246-6.x86_64

Installed:
  mssql-server.x86_64 0:14.0.1.246-6

Complete!
```

Configure the MSSQL Server by accepting the license and supplying a password for the System Administrator (SA) account:

```
Host# /opt/mssql/bin/sqlservr-setup
...
Do you accept the license terms? If so, please type YES: YES
...
Please enter a password for the system administrator (SA) account: P@ssw0rd
Please confirm the password for the administrator (SA) account: P@ssw0rd
...
Do you wish to start the SQL Server service now? [y/n]: y
Do you wish to enable SQL Server to start on boot? [y/n]: y

Setup completed successfully
```

Verify it is active (running):

```
Host# systemctl status mssql-server
```

```
Host# systemctl status mssql-server
* mssql-server.service - Microsoft(R) SQL Server(R) Database Engine
   Loaded: loaded (/usr/lib/systemd/system/mssql-server.service; disabled;
   Active: active (running) since Fri 2016-10-28 07:54:16 CDT; 19s ago
   Main PID: 1675 (sqlservr)
   CGroup: /system.slice/mssql-server.service
           |-1675 /opt/mssql/bin/sqlservr
           `--1677 /opt/mssql/bin/sqlservr
```

Configure the firewall to open the tcp port 1433:

```
Host# firewall-cmd --zone=public --add-port=1433/tcp --permanent
Host# firewall-cmd --reload
```

## Database restoration preparation

In this paragraph, we create several directories that will be used for database restoration locations: `disk_log` and `disk_data` will hold the corresponding files on the internal disk; `pmem_log` and `pmem_data` will be used as NVDIMM-N mount points for the log and data files:

```
Host# cd /var/opt/mssql
Host# mkdir disk_{log,data} pmem_{log,data}
Host# chown mssql:mssql disk_{log,data} pmem_{log,data}
```

Restore the AdventureWorks database with the log file on the internal disk. You can review the content of the **RestoreAdventureWithLogOnDisk.sql**

```
Host# cd /usr/kits/SQL-Linux
Host# cat RestoreDBWithLogAndDataOnDisk.sql
...
```

```
Host# cat RestoreDBWithLogAndDataOnDisk.sql
-- RestoreDBWithLogAndDataOnDisk.sql
-- Version 0.6

-- Restore the AdventureWorks DB with Log and Data files on internal disks
USE [master]
RESTORE DATABASE AdventureWorks
FROM DISK = '/var/opt/mssql/AdventureWorks.bak'
WITH MOVE 'AdventureWorks2014_Data' TO '/var/opt/mssql/disk_data/AdventureWorks2014_Data.mdf',
MOVE 'AdventureWorks2014_Log' TO '/var/opt/mssql/disk_log/AdventureWorks2014_Log.ldf'
GO

-- Set Recovery mode to full
ALTER DATABASE AdventureWorks SET RECOVERY FULL ;
GO
```

Restore the Database (between 2 and 3 minutes) and verify that you can perform a simple query:

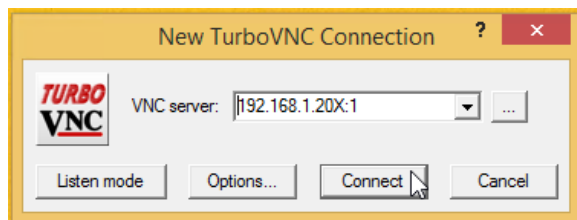
```
Host# sql -i RestoreDBWithLogAndDataOnDisk.sql
Host# sql -i SimpleQuery.sql
```

## Database stress with log and data files on disk

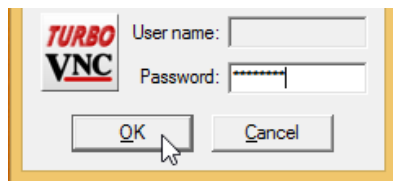
Before stressing the DB, we want to have a graphical view of the CPU load. Click on the **VNC** icon in the taskbar to start a VNC session toward your Linux server to prepare the next exercise:



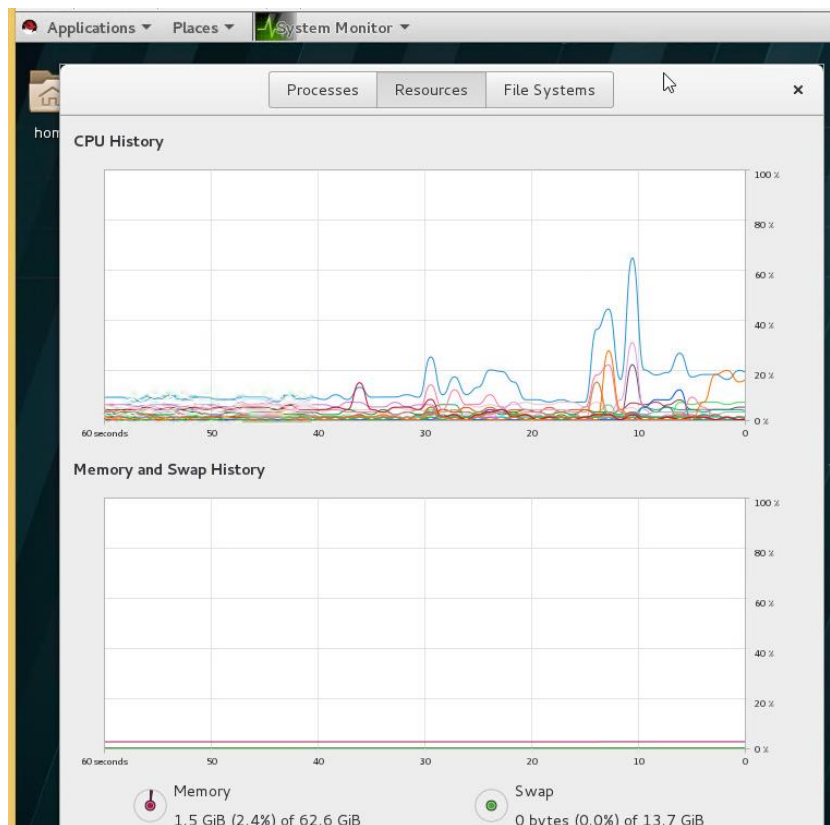
Enter your server's IP address and followed by **:1** and click on **Connect**:



Enter **password** as password and click **Ok**:



You should see a desktop with the **gnome-system-monitor** started. If not, notify the instructor. **Resize** the **gnome-system-monitor** to enlarge the CPU graph:



To stress the DB, we will use a complex query contained in the `cpu_stress.sh` script. The runtime will be measured by the `time` built-in command:

```
Host# cat cpu_stress.sh
#!/usr/bin/bash

# Version: 0.2

# This script performs a complex query on a the lineitem table generating
# a lot of I/Os and CPU usage.
#
# The number of iteration of the query is passed as an argument
#
# The lineitem table has been added (and populated) in the AdventureWorks
# Database. This table is not part of the originale AdventureWorks2014 DB.

if [[ -z $1 ]]; then
    printf " Error : Usage : ${0##*/} iterations \n\007"
    printf " Exiting\n\007"
    exit 1
fi

NUM_ITER=$1

LOGIN=sa
PASS=P@ssw0rd
BASE=AdventureWorks

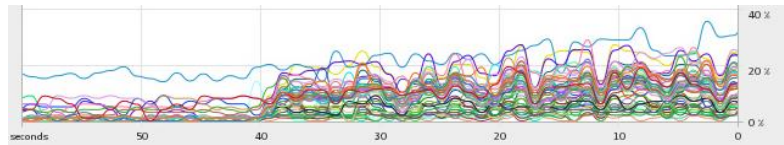
cmdsql=$( sqlcmd -U $LOGIN -P $PASS -d $BASE -p -Q "
SET FMONLY OFF;
SET NOCOUNT ON;
DECLARE @RowCount int, @Iteration int = 1;
WHILE @Iteration <= $NUM_ITER
BEGIN
    CHECKPOINT;
    DBCC DROPCLEANBUFFERS WITH NO_INFOMSGS;
select l_returnflag, l_linestatus, sum(l_quantity) as sum_qty,
sum(l_extendedprice) as sum_base_price, sum(l_extendedprice*(1-l_discount)) as sum_disc_price,
sum(l_extendedprice*(1-l_discount)*(1+l_tax)) as sum_charge, avg(l_quantity) as avg_qty,
avg(l_extendedprice) as avg_price, avg(l_discount) as avg_disc, count(*) as count_order
from lineitem
group by l_returnflag, l_linestatus
order by l_returnflag,l_linestatus;
SET @Iteration += 1;
END")
```

With the log and data files on disk, one iteration of the query should last about 1 minute and 20 seconds. Launch the query with 1 iteration:

```
Host# time cpu_stress.sh 1

real    1m20.474s
user    0m0.004s
sys     0m0.008s
```

During the run, take a look at the CPU graph; the load of each core should barely exceed 20%:



To prepare the next exercise, we will drop the DB and restore it later. We could have used an Offline/Online or Detach/Attach method to move the log file on a persistent memory location. However, to avoid any problem during those phases it is safer to drop/restore. Moreover, doing so makes sure that we are starting the second test in the exact same conditions as the first one:

```
Host# sql -i DropAdventureWorks.sql
Host#
```



## Exercise 4 - MSSQL and Persistent memory

In this exercise, we will place the log and data files of our database on [HPE Non Volatiles DIMMs \(NVDIMM-N\)](#) and then stress it the same way we did in the previous exercise.

### Persistent Memory preparation

Verify that your system has four NVDIMMs of type-N (former type-1). Each NVDIMM holds 8GB of DDR4, a memory controller and a NAND flash. In case of power failure, the Smart-Array battery will allow the transfer of the DDR4 into the NAND flash.

Upon reboot the content of the NAND flash will be transferred automatically back to the DDR4.

Review the content of the provided `GetNVDIMMInventory.sh` script, and then retrieve the memory composition of your system:

```
Host# cat GetNVDIMMInventory.sh
#!/usr/bin/sh

# This script uses the hprest interface tool to retrieve all the
# DIMMs present in the system. It prints their location (procNdimmx)
# and their technology (RDIMM or RNVDIMM).

# Version: 0.5

HPREST="hprest --nologo --redfish"

# Inband log in iLO
$HPREST login >/dev/null 2>&1

# Find DIMM locations
DIMM_LOCATION=$( $HPREST rawget /redfish/v1/systems/1/memory/ \
  --silent | \
  jq '[.Members[]]' | awk -F\" '/redfish/ {print $2}' )

# For each DIMM extract and print its technology
for dimm in $DIMM_LOCATION ; do
  DIMM_TECHNO=$( $HPREST rawget $dimm --silent | grep DIMMTechnology )
  d=${dimm%%/*}
  echo "${d##*/}: $DIMM_TECHNO"
done

# Logout from iLO
$HPREST logout >/dev/null 2>&1
exit
```

```
Host# ./GetNVDIMMInventory.sh
```

```
Host# ./GetNVDIMMInventory.sh
proc1dimm1: "DIMMTechnology": "RNVDIMM"
proc1dimm4: "DIMMTechnology": "RNVDIMM"
proc1dimm9: "DIMMTechnology": "RDIMM"
proc1dimm12: "DIMMTechnology": "RDIMM"
proc2dimm1: "DIMMTechnology": "RNVDIMM"
proc2dimm4: "DIMMTechnology": "RNVDIMM"
proc2dimm9: "DIMMTechnology": "RDIMM"
proc2dimm12: "DIMMTechnology": "RDIMM"
Host#
```

The above screenshot shows four RNDIMMs (two per socket) and four regular RDIMMs.

The `NvDimmNMemInterleaving` UEFI parameter controls how the NVDIMMs are presented to the OS; if disabled the OS sees one `/dev/pmem` device per NVDIMM. If enabled the OS sees one `pmem` device per socket.

Get the value of this parameter and then, verify you have effectively two `pmem` devices, each of them having a size of 16GB:

```
Host# hprest login --selector HpBios.
Host# hprest get | grep -i meminterl
NvDimmNMemInterleaving=Enabled

Host# fdisk -l /dev/pmem*
```

```
Host# ls /dev/pmem*
/dev/pmem0 /dev/pmem1
Host# fdisk -l /dev/pmem*

Disk /dev/pmem0: 17.2 GB, 17179869184 bytes, 33554432 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 4096 bytes
I/O size (minimum/optimal): 4096 bytes / 4096 bytes

Disk /dev/pmem1: 17.2 GB, 17179869184 bytes, 33554432 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 4096 bytes
I/O size (minimum/optimal): 4096 bytes / 4096 bytes

Host#
```

In the next paragraph, we will use `/dev/pmem0` for the log file and `/dev/pmem1` for the data file. Create a filesystem on them, mount them and make sure it belongs to the `mssql` user and group.

In order to get the `pmem` devices mounted automatically upon reboot and before the start of the `mssql` service (next exercise), we need to modify the `/etc/fstab` file:

```
Host# mkfs -t xfs -f /dev/pmem0
Host# mkfs -t xfs -f /dev/pmem1

Host# sed -i 's/^##//' /etc/fstab

Host# mount -a
Host# mount | grep pmem
/dev/pmem0 on /var/opt/mssql/pmem_log type xfs ...
/dev/pmem1 on /var/opt/mssql/pmem_data type xfs ...

Host# chown mssql:mssql /var/opt/mssql/pmem_{log,data}
Host# ls -ld /var/opt/mssql/pmem*
drwxr-xr-x 2 mssql mssql 6 Dec 16 08:51 /var/opt/mssql/pmem_data
drwxr-xr-x 2 mssql mssql 6 Dec 16 08:51 /var/opt/mssql/pmem_log
```

## Database restoration with log and data files on PMEMs

Restore the AdventureWorks database with the log and data files on PMEMs. You can review the content of the `RestoreDBWithLogAndDataOnPmem.sql`:

```
Host# cat RestoreDBWithLogAndDataOnPmem.sql
...
USE [master]
RESTORE DATABASE AdventureWorks
FROM DISK = '/var/opt/mssql/AdventureWorks.bak'
WITH MOVE 'AdventureWorks2014_Data' TO
'/var/opt/mssql/pmem_data/AdventureWorks2014_Data.mdf',
MOVE 'AdventureWorks2014_Log'
TO '/var/opt/mssql/pmem_log/AdventureWorks2014_Log.ldf'
GO
...

Host# sql -i RestoreDBWithLogAndDataOnPmem.sql
```

Verify that you can perform a simple query:

```
Host# sql -i SimpleQuery.sql
Changed database context to 'AdventureWorks'.
...
(16 rows affected)
```

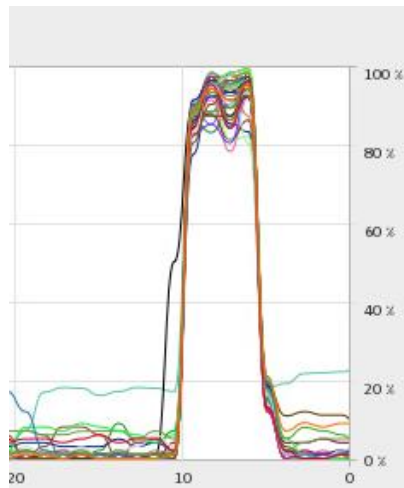
## Database stress with log and data files on PMEMs

Submit one iteration of the complex query:

```
Host# time cpu_stress.sh 1

real    0m5.668s
user    0m0.004s
sys     0m0.007s
```

You should notice a runtime of less than **10sec** and a CPU load pattern much higher compared to the previous run:



## Exercise 5 - NVDIMM persistency

In this exercise, we simulate a power failure to test the NVDIMMs persistency. Review and then launch the `test-persistency.sh` script. This script increments the `ScrappedQty` value each second:

```
hstlinvm# cat test-persistency.sh
#!/bin/sh

# This script increments each second a record in the database.
# Run it, open a session to the iLO of the SQL Server and
# power off hard to simulate a power cut.
# Note the last record number.

# Wait 6 seconds and power on

# After recovery of the DB, run the VerifyUpdateRecord.sql to
# verify that the record has the correct value.

# Version: 0.5

let i=1
SQL='sqlcmd -S. -U SA -P P@ssw0rd '

while [ $i ] ; do
    $SQL -d AdventureWorks -Q "Update Production.WorkOrder \
    SET ScrappedQty = $i \
    WHERE WorkOrderID = 70370"
    echo "Qty updated with $i"
    let i++
    sleep 1
done
```

```
Host# ./test-persistency.sh
```

```
(1 rows affected)
Qty updated with 1

...

(1 rows affected)
Qty updated with 18
```

Open a PuTTY/Cygwin session toward the iLO of your server at `192.168.1.10X` (`X` is your group number) with username **demopaq** and **password** as password.

Simulate the power outage:

```
</>hpiLO-> power off hard
```

When the scripts stops modifying the database, note the record number. We will need it when the server is back online.

Wait 10/15 seconds and **power on** the server. Start the Virtual Serial Port (**vsp**). Watch the server during the POST and the boot process. You should see several messages concerning the recovery of the NVDIMMs:

```
</>hpiLO-> power on
```

```
...
```

```
</>hpiLO-> vsp
```

When the server is back online (login prompt), open a PuTTY/Cygwin session as **root** / **password** and verify the value of the **ScrappedQty** counter:

```
Host# cd /usr/kits/SQL-Linux
```

```
Host# sql -i VerifyUpdatedRecord.sql
```

```
Host# sql -i VerifyUpdatedRecord.sql
Changed database context to 'AdventureWorks'.
WorkOrderID ScrappedQty
-----
          70370          18

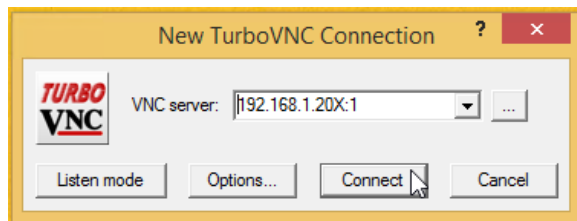
(1 rows affected)
Host#
```

## Ex 6: Use of HammerDB

**HammerDB** is a recognized database tester being able to load several databases including MS SQL Server. Starting at version 2.22 HammerDB supports the Microsoft ODBC driver for Linux and thus can be launched on Linux for loading a MS SQL Server database located either on Linux or Windows. Moreover, this version of HammerDB allows In Memory OLTP for improving the performance of various data processing.

In this exercise, we will perform a simple OLTP test with HammerDB launched on our Linux server.

Restart the VNC session toward your server (`password` for `password`) and install HammerDB from the X terminal of the VNC session:

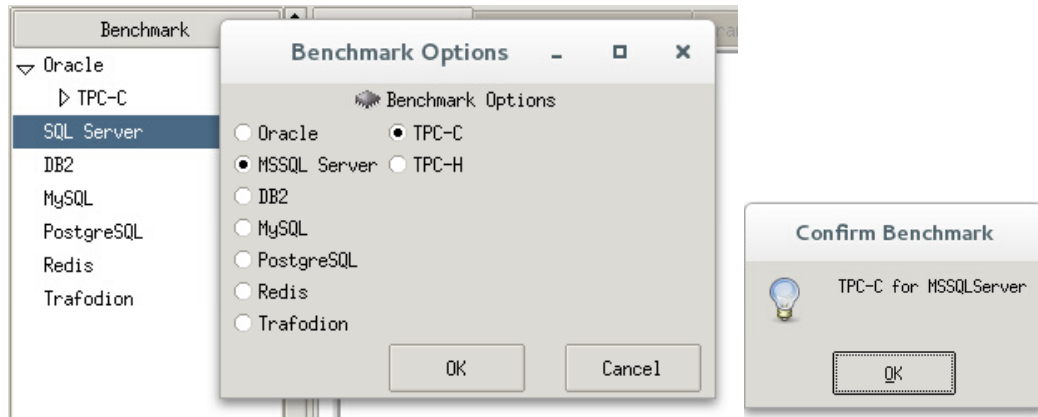


```
Host# yum install libXScrnSaver -y
Host# cd /usr/kits/SQL-Linux
Host# ./HammerDB --mode silent --prefix /usr/kits/SQL-Linux/HDB
```

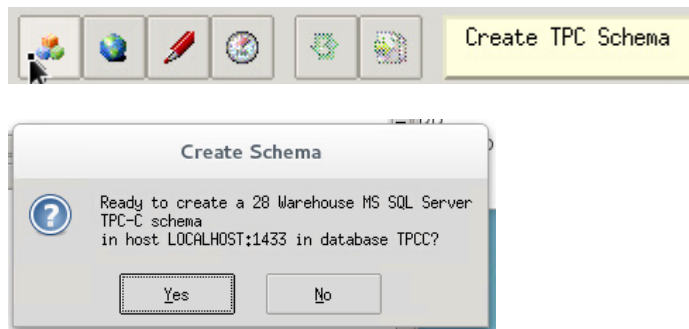
Still from the X Terminal, modify the HammerDB configuration file (`config.xml`) with the SA password, 28 virtual users (one per core), 28 Data WareHouses and launch it in background:

```
Host# cd HDB
Host# sed -i 's/admin/P@ssw0rd/' config.xml
Host# sed -i 's/\(_users>\)1/\128/' config.xml
Host# sed -i 's/\(_threads>\)1/\128/' config.xml
Host# sed -i 's/\(_ware>\)1/\128/' config.xml
Host# ./hammerdb.tcl &
```

In the left pane, Double click on **SQL Server** and click **Ok** twice:



In the top row of icons, click on the **Create TPC Schema** icon and then **Yes**:



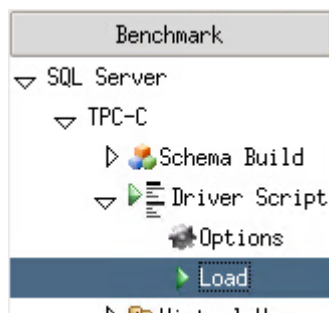
Once the Schema is complete (~5 minutes) , click on the **Destroy Virtual**



**User icon:**

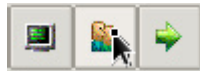
```
Virtual User 1
Workers: 0 Active 28 Done
CREATING TPCC INDEXES
CREATING TPCC STORED
PROCEDURES
UPDATING SCHEMA STATISTICS
TPCC SCHEMA COMPLETE
```

In the left pane, expand the **TPC-C -> Driver Script** menu and double click on the **Load** icon:





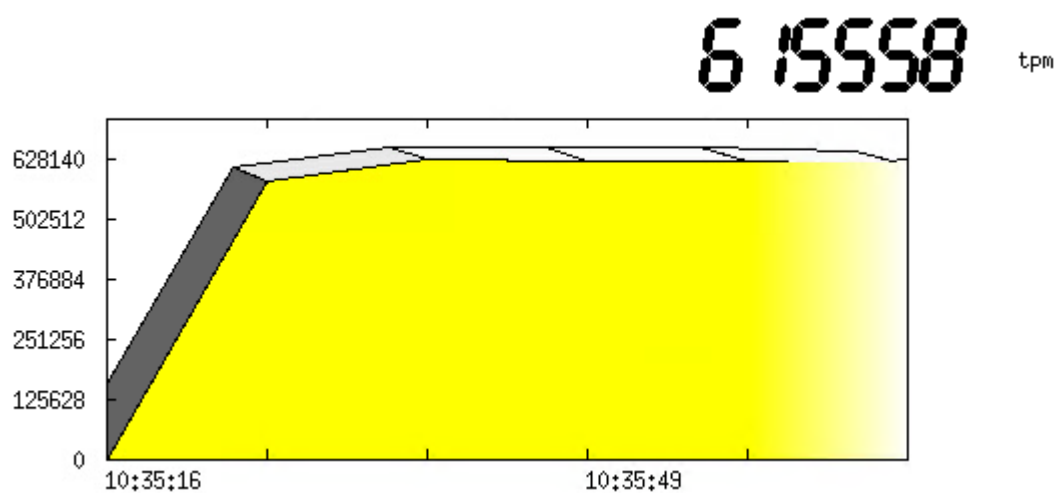
Create the Virtual Users:



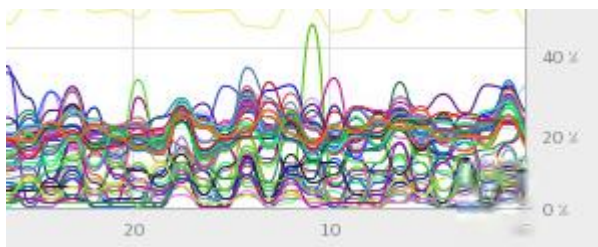
Run the Virtual Users:



Open the Transaction Counter and wait for the graph to appear:



Remember that that we are evaluating a Preview version of MS SQL Server on Linux and thus, the number of TPMs you obtain may be not representative of what we will get in the final version of the product.



Once the load is stabilized you can stop the **Transaction Counter** and kill the virtual users:



## Ex7 - In Memory OLTP with HammerDB

In this exercise, we will enable the In Memory OLTP feature of HammerDB 2.22. As explained in the [Microsoft documentation](#) this feature requires a several preliminary steps before populating the database. Hence we need first to drop the tpcc database used in the previous exercise. If the following command fails because the database is still in use, kill HammerDB and restart it in background:

```
Host# sql -Q "drop database tpcc ; "
```

Review the supplied **CreateTpccWithInMemoryOnPmem.sql** script. This script creates an empty database with the data and log files on the PMEM devices:

```
Host# cd /usr/kits/SQL-Linux/
Host# cat CreateTpccWithInMemoryOnPmem.sql
-- Create a database suitable for In Memory OLTP with data and log file on pmem.
--
-- Version: 0.1

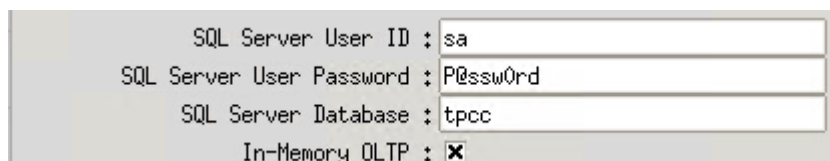
CREATE DATABASE tpcc
on
( NAME = N'tpcc', FILENAME = N'/var/opt/mssql/pmem_data/tpcc.mdf' )
LOG ON
( NAME = N'tpcc_log', FILENAME = N'/var/opt/mssql/pmem_log/tpcc_log.ldf' )
GO

ALTER DATABASE tpcc ADD FILEGROUP tpcc_mod CONTAINS MEMORY_OPTIMIZED_DATA
ALTER DATABASE tpcc ADD FILE (name='tpcc_mod1',
filename='c:\var\opt\mssql\pmem_data\tpcc_mod1') TO FILEGROUP tpcc_mod
ALTER DATABASE tpcc SET MEMORY_OPTIMIZED_ELEVATE_TO_SNAPSHOT=ON
GO
Host#
```

Run the script:

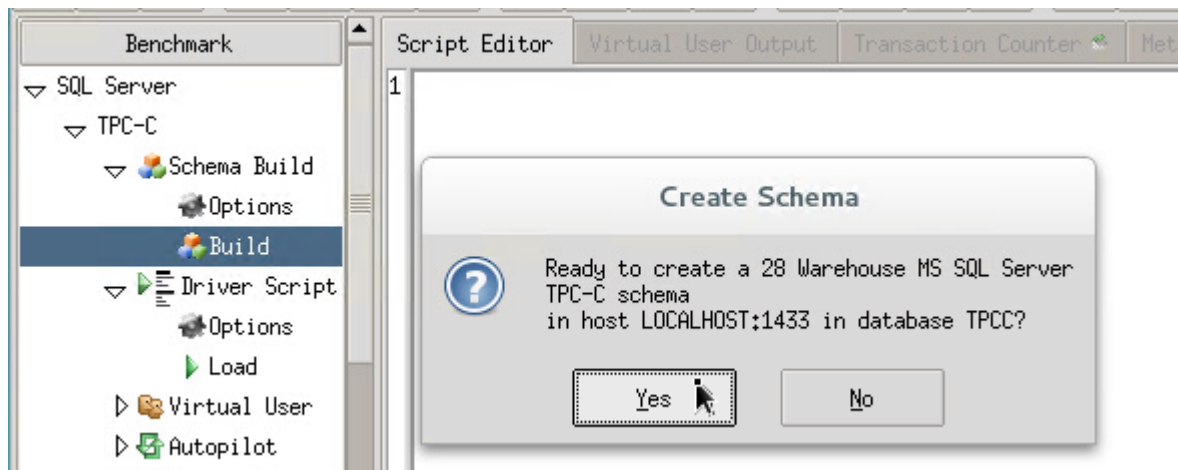
```
Host# cd /usr/kits/SQL-Linux
Host# sql -i CreateTpccWithInMemoryOnPmem.sql
Host#
```

In the left pane of HammerDB, open the **Schema Build -> Options** dialog box and **tick** the **In-Memory OLTP** box. Then, click **OK**:

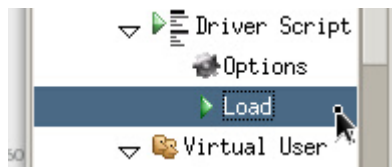


SQL Server User ID :	sa
SQL Server User Password :	P@ssw0rd
SQL Server Database :	tpcc
In-Memory OLTP :	<input checked="" type="checkbox"/>

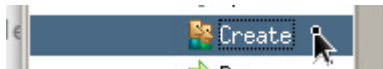
Double click on **Schema Build -> Build** and then click on **Yes** to build the schema:



When the Schema is **Complete**, kill the Virtual user  and then double click on the **Driver Script -> Load** icon:



**Create** the 28 Virtual Users by Double clicking on the **Virtual User -> Create** icon:



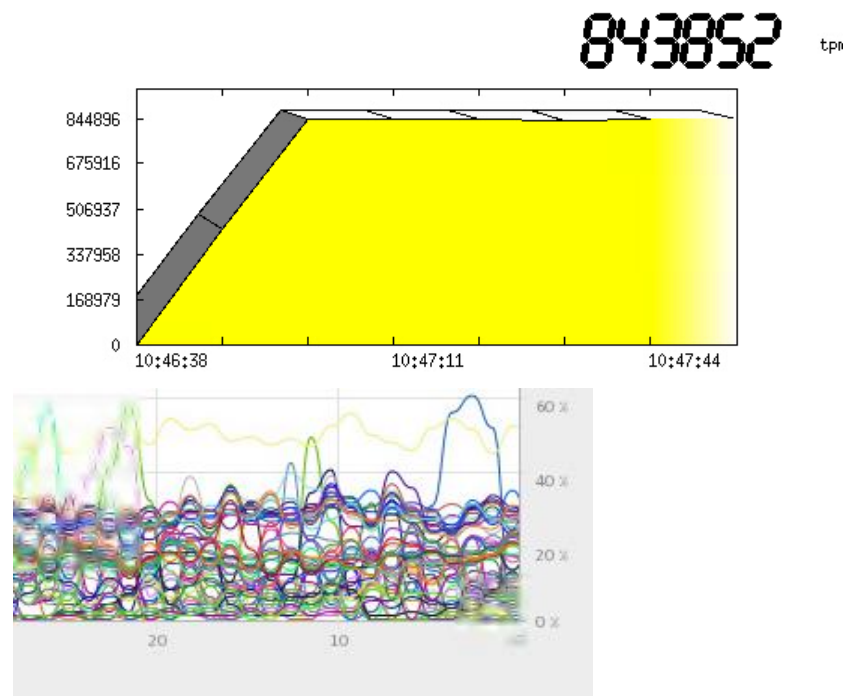
Start the **Run**:



Open the **Transaction Counter** pane and wait for the graphic to appear:



NOTE: With this preview version of MS SQL Server for Linux, you may not get much higher performance with the In-Memory feature compared to the previous exercise.



Let the load go for a while and then clean up the environment before jumping to the next exercise:

Stop the **Transaction counter**, **Destroy** the virtual users and kill the HammerDB application.

## EX 8: Create a Python application using SQL Server on Linux

Along with SQL Server on Linux, [Microsoft helps developers](#) to develop applications in their favorite language.

In this example, we deploy a suitable environment for developing a Python application to be used against our MS SQL Server for Linux. Then, we will create a simple database and run a CRUD (Create, Read, Update, Delete) Python script against it.

In the las exercise, we compare the performance of arithmetical operations without Columnstore indexes first and then with Columnstore indexes.

For achieving this goal, several required packages are located in the Extra Packages for RHEL (EPEL) repository. The EPEL repository file is already on your server but is disabled. We need first to enable it:

```
Host# yum-config-manager --enable epel
...
```

Install Python 2.7 with ad hoc development modules:

```
Host# yum install -y python python-pip python-wheel python-devel
Host# yum group install -y "Development tools"
```

Download and the Microsoft tools for MS SQL Server on Linux repo file and configure it:

Install the `unixODBC-utf16-devel` package:

```
Host# yum install unixODBC-devel -y
```

Run a basic query retrieving the version of the SQL Server:

```
Host# sql -Q "SELECT @@VERSION"
```

Install the MS ODBC Python module (ignore the yellow warning message if any):

```
Host# pip install pyodbc==3.1.1
```

Create an empty database

```
Host# sql -Q "CREATE DATABASE SampleDB;"
```

Review the `crud.py` file which creates, read, update and delete a few rows to verify everything works fine:

```
Host# cd /usr/kits/SQL-Linux/
Host# cat crud.py
import pyodbc
server = 'localhost'
database = 'SampleDB'
username = 'sa'
password = 'P@ssw0rd'
cnxn = pyodbc.connect('DRIVER={ODBC Driver 13 for SQL Server};SERVER='+server+';PORT=1443;DATABASE='+database+';UID='+username+';PWD='+ password)
cursor = cnxn.cursor()

print ('Creating Table')
tsql = "CREATE TABLE Employees (Name nvarchar(255), Location nvarchar(255));"
if cursor.execute(tsql):
    print ('Successfully Created Table!')
cnxn.commit()

print ('Inserting a new row into table')
#Insert Query
tsql = "INSERT INTO Employees (Name, Location) VALUES (?,?);"
if cursor.execute(tsql, 'Jake', 'United States'):
    print ('Successfully Inserted!')
cnxn.commit()

#Update Query
print ('Updating Location for Nikita')

tsql="UPDATE Employees SET Location = ? WHERE Name = ?"
if cursor.execute(tsql, 'Sweden', 'Nikita'):
    print ('Successfully Updated!')
cnxn.commit()

#Delete Query
print ('Deleting user Jared')
tsql="DELETE FROM Employees WHERE Name = ?"
if cursor.execute(tsql, 'Jared'):
    print ('Successfully Deleted!')
cnxn.commit()

#Select Query
print ('Reading data from table')
tsql="SELECT Name, Location FROM Employees;"
if cursor.execute(tsql):
    row = cursor.fetchone()
    while row:
        print (str(row[0]) + " " + str(row[1]))
        row = cursor.fetchone()
Host#
```

Run the script:

```
Host# cd /usr/kits/SQL-Linux
Host# python crud.py
...
```

## Columnstore indexes

MS SQL Server for Linux supports **Columnstore** indexes. In this exercise, we will create and populate a table with 5,000,000 rows of random data. Then, we will perform operations on this table; first with no columnstore indexes and second with columnstore indexes.

Review the `CreateSampleTable.sql` script and run it to create the table and populate it:

```
Host# cat CreateSampleTable.sql
...
Host# sql -d SampleDB -i CreateSampleTable.sql
```

Review the `columnstore.py` program which performs various arithmetical operations on the table:

```
Host# cat columnstore.py
import pyodbc
from datetime import datetime
server = 'localhost'
database = 'SampleDB'
username = 'sa'
password = 'P@ssw0rd'
cnxn = pyodbc.connect('DRIVER={ODBC Driver 13 for SQL Server};SERVER='+server+';PORT=1433;DATABASE='+database+';UID='+username+';PWD='+ password)
cursor = cnxn.cursor()
tsql = "SELECT SUM(Price) as sum FROM Table_with_5M_rows"
a = datetime.now()
if cursor.execute(tsql):
    b = datetime.now()
    c = b - a
    row = cursor.fetchone()
    while row:
        print ('Sum:', str(row[0]))
        row = cursor.fetchone()
    print ('QueryTime:', c.microseconds, 'ms')
cnxn.commit()
Host#
```

Launch it and note the `QueryTime`:

```
Host# python columnstore.py
...
('QueryTime:', 37667, 'ms')
```

Create columnstore indexes in this table and run the python program again. You should notice a faster `QueryTime`.

```
Host# sql -d SampleDB -Q "CREATE CLUSTERED COLUMNSTORE \
    INDEX Columnstoreindex ON Table_with_5M_rows;"
Host# python columnstore.py
```

## End of the lab