To go along with the Understanding how SQL Server executes a query.pdf document here is a good talk by kevin kline:<https://www.youtube.com/watch?v=34VqSliEfsc> Also, a simplified picture is shown below in this word doc.

Course hands on lab content <https://www.dropbox.com/s/x74iqdqevr2eknw/SQL_Admin_JumpStart.iso?dl=0>

Hands on content at <https://www.microsoft.com/en-au/learning/companion-moc.aspx>

Mva hands on lab content <https://www.dropbox.com/s/x74iqdqevr2eknw/SQL_Admin_JumpStart.iso?dl=0>

* 1GiB = 1024MiB = 1024x1024KiB = 1024x1024x1024B
* 1GB = 1000MB = 1000x1000KB = 1000x1000x1000B

CRYSTALDISKMARK AND DISKSPD

--find the open connections to the database

USE master

SELECT \* FROM sys.sysprocesses WHERE dbid = DB\_ID('TinyDB')

--this should also give some idea

exec sp\_who2

exec sp\_whoisactive

KILL <SPID>

--you could also try to set it to read-only in which case it will ask u if u want to close all connections

--u can also open Activity monitor in SSMS context menu for the server to find the processes

--get the current instance configs

USE master;

GO

SELECT \* FROM sys.configurations;

GO

-- Step 1: Open and execute the query 91 - Workload.sql from

-- Solution Explorer. WARNING: Make sure you have followed

-- the instructions at the top of that window before clicking

-- Execute.

-- Step 2: Query the currently executing requests.

-- Note that a large number of requests is returned but that

-- most are system requests.

SELECT \* FROM sys.dm\_exec\_requests;

GO

-- Step 3: The is\_user\_process column of the sys.dm\_exec\_sessions view can

-- be used to filter out system activity

SELECT \* FROM sys.dm\_exec\_sessions WHERE is\_user\_process = 1;

GO

-- Step 4: Use that column to filter the currently executing requests

-- by joining the two tables on session\_id.

SELECT s.original\_login\_name, s.program\_name, r.command,

r.wait\_type, r.wait\_time, r.blocking\_session\_id, r.sql\_handle

FROM sys.dm\_exec\_requests AS r

INNER JOIN sys.dm\_exec\_sessions AS s

ON r.session\_id = s.session\_id

WHERE s.is\_user\_process = 1;

GO

-- Step 5: Note that we can also retrieve details of the SQL Batch that

-- is being executed, instead of just the handle. We do that by

-- using the sys.dm\_exec\_sql\_text function.

SELECT s.original\_login\_name, s.program\_name, r.command, t.text,

r.wait\_type, r.wait\_time, r.blocking\_session\_id

FROM sys.dm\_exec\_requests AS r

INNER JOIN sys.dm\_exec\_sessions AS s

ON r.session\_id = s.session\_id

OUTER APPLY sys.dm\_exec\_sql\_text(r.sql\_handle) AS t

WHERE s.is\_user\_process = 1;

GO

-- Step 6: Do not be too concerned about the complexity of the subquery

-- below but note that it is possible to find the actual statement

-- that is being executed rather than the batch.

SELECT s.original\_login\_name, s.program\_name, r.command,

(SELECT TOP (1) SUBSTRING(t.text, r.statement\_start\_offset / 2 + 1,

((CASE WHEN r.statement\_end\_offset = -1

THEN (LEN(CONVERT(nvarchar(max), t.text)) \* 2)

ELSE r.statement\_end\_offset

END) - r.statement\_start\_offset) / 2 + 1)) AS SqlStatement,

r.wait\_type, r.wait\_time, r.blocking\_session\_id

FROM sys.dm\_exec\_requests AS r

INNER JOIN sys.dm\_exec\_sessions AS s

ON r.session\_id = s.session\_id

OUTER APPLY sys.dm\_exec\_sql\_text (r.sql\_handle) AS t

WHERE s.is\_user\_process = 1;

GO

-- Step 7: Stop the script execution in the 91 - Workload.sql window

-- (Change to that window and click the Cancel Executing

-- Query toolbar button then return to this window)

-- Step 8: Investigate how the procedure cache is distributed at this point

SELECT cacheobjtype,

objtype ,

COUNT(\*) as CountofPlans,

SUM(usecounts) as UsageCount,

SUM(usecounts)/CAST(count(\*)as float) as AvgUsed ,

SUM(size\_in\_bytes)/1024./1024. as SizeinMB

FROM sys.dm\_exec\_cached\_plans

GROUP BY cacheobjtype, objtype

ORDER BY CountOfPlans DESC;

GO

-- Step 9: Locate the top 10 queries based on Average Reads

SELECT TOP (10) total\_logical\_reads/execution\_count AS AvgReads,

SUBSTRING(st.text, (qs.statement\_start\_offset/2) + 1,

((CASE statement\_end\_offset

WHEN -1 THEN DATALENGTH(st.text)

ELSE qs.statement\_end\_offset END

- qs.statement\_start\_offset)/2) + 1) as StatementText

FROM sys.dm\_exec\_query\_stats AS qs

CROSS APPLY sys.dm\_exec\_sql\_text(qs.sql\_handle) AS st

ORDER BY total\_logical\_reads/execution\_count DESC;

GO

-- Step 10: View I/O statistics for the database files

SELECT DB\_NAME(fs.database\_id) AS DatabaseName,

mf.name AS FileName,

mf.type\_desc,

fs.\*

FROM sys.dm\_io\_virtual\_file\_stats(NULL,NULL) AS fs

INNER JOIN sys.master\_files AS mf

ON fs.database\_id = mf.database\_id

AND fs.file\_id = mf.file\_id

ORDER BY fs.database\_id, fs.file\_id DESC;

GO

-- Step 11: View general wait statistics

SELECT \* FROM sys.dm\_os\_wait\_stats;

GO

select \*

from msdb.dbo.suspect\_pages

--size of db

exec sp\_spaceused

--size of db

SELECT DB\_NAME(database\_id) AS DatabaseName,

Name AS Logical\_Name,

Physical\_Name,

(size \* 8) / 1024 SizeMB

FROM sys.master\_files

WHERE DB\_NAME(database\_id) = 'ManufacturingDB'

select \*

from sys.dm\_db\_file\_space\_usage

select \*

from sys.dm\_db\_log\_space\_usage

Use DB2BMoved;

GO

Select \* from sys.database\_files;

|  |  |
| --- | --- |
| **Drive Letter and Directory** | **Purpose** |
| C: | System drive, Page file |
| L:\SQLLogs  K:\SQLLogs | SQL Server Log drive(s) |
| M:\SQLBackups  N:\SQLBackups | SQL Server Backup drive(s) |
| P:\SQLData  Q:\SQLData  R:\SQLData  S:SQLData | SQL Server Data drive(s) |
| T:\TempDB | SQL Server TempDB |
| Z: | Optical drive |

Table 1: Naming Standards and Structure for Logical Drives

<https://www.red-gate.com/simple-talk/sql/database-administration/provisioning-a-new-sql-server-instance-part-two/>

**Physical RAM**                    **MaxServerMem Setting**   
8GB                                        6000   
12GB                                     9000   
16GB                                     12000   
24GB                                     20000  
32GB                                     27000   
48GB                                     43000   
64GB                                     58000  
72GB                                     66000  
96GB                                     90000  
128GB                                   120000  
192GB                                   184000  
256GB                                   248000

--Database Backups

SELECT

CONVERT(CHAR(100), SERVERPROPERTY('Servername')) AS Server,

msdb.dbo.backupset.database\_name,

msdb.dbo.backupset.backup\_start\_date,

msdb.dbo.backupset.backup\_finish\_date,

msdb.dbo.backupset.expiration\_date,

CASE msdb..backupset.type

WHEN 'D' THEN 'Full Database backup'

WHEN 'I' THEN 'Differential'

WHEN 'L' THEN 'Transaction Log'

END AS backup\_type,

msdb.dbo.backupset.backup\_size,

msdb.dbo.backupmediafamily.logical\_device\_name,

msdb.dbo.backupmediafamily.physical\_device\_name,

msdb.dbo.backupset.name AS backupset\_name,

msdb.dbo.backupset.description

FROM msdb.dbo.backupmediafamily

INNER JOIN msdb.dbo.backupset ON msdb.dbo.backupmediafamily.media\_set\_id = msdb.dbo.backupset.media\_set\_id

WHERE (CONVERT(datetime, msdb.dbo.backupset.backup\_start\_date, 102) >= GETDATE() - 7)

and msdb.dbo.backupset.database\_name like '%natsoil%'

ORDER BY

msdb.dbo.backupset.database\_name,

msdb.dbo.backupset.backup\_finish\_date

--Database Backups

SELECT

BS.server\_name AS [Server Name]

,BS.database\_name AS [Database Name]

,BS.recovery\_model AS [Recovery Model]

,BMF.physical\_device\_name [Location]

,(CAST(BS.backup\_size / 1000000 AS INT)) AS [Size of Backup (MB)]

,CASE BS.[type] WHEN 'D' THEN 'Full'

WHEN 'I' THEN 'Differential'

WHEN 'L' THEN 'Transaction Log'

END AS [Type of Backup]

,BS.backup\_start\_date AS [Backup Date]

,BS.first\_lsn AS [First LSN]

,BS.last\_lsn AS [Last LSN]

FROM msdb.dbo.backupset BS

INNER JOIN msdb.dbo.backupmediafamily BMF ON BS.media\_set\_id = BMF.media\_set\_id

WHERE BS.database\_name like '%natsoil%'

ORDER BY backup\_start\_date DESC

,backup\_finish\_date

If you don’t have clear RPO/RTO requirements from your business, you might start out with a job schedule like this for ola hallengren scripts:

| **Job Name** | **Run Schedule** |
| --- | --- |
| CommandLog Cleanup | Every Sunday at 12:00AM |
| DatabaseBackup – SYSTEM\_DATABASES – FULL | Every day at 11:55PM |
| DatabaseBackup – USER\_DATABASES – DIFF | Every day at 12:00PM |
| DatabaseBackup – USER\_DATABASES – FULL | Every day at 12:00AM |
| DatabaseBackup – USER\_DATABASES – LOG | Every 30 minutes |
| DatabaseIntegrityCheck – SYSTEM\_DATABASES | Every Saturday at 8:00AM |
| DatabaseIntegrityCheck – USER\_DATABASES | Every Saturday at 9:00AM |
| IndexOptimize – USER\_DATABASES | Every Monday at 3:00AM |
| Output File Cleanup | Every Sunday at 12:01AM |
| sp\_delete\_backuphistory | Every Sunday at 12:02AM |
| sp\_purge\_jobhistory | Every Sunday at 12:03AM |

If we have a Service Level Agreement (SLA) of 10 minutes, take log backup every 10 minutes.

Raid 01 vs raid 10:

**Example Scenario**

For example, with a 30-minute data-loss requirement and a 2 hour downtime requirement, you might design a restore strategy that is:

* Perform a tail-of-the-log backup
* Restore from the most recent full backup
* Restore all log backups since the most recent full backup

And then set the full backup frequency at every day at midnight and the log backup frequency at every half hour. The full backup takes 20 minutes to complete and each log backup takes 5 minutes to complete.

The worst case scenario is that a crash occurs at 23:59:59. This means the restore sequence is (with some estimated completion times):

* Perform the tail-of-the-log backup to backup the log generated since the log backup at 23:30. This takes 5 minutes.
* Restore the full backup from midnight. This takes 20 minutes.
* Restore the 48 log backups from today, starting with the one at 00:00 and ending with the one at 23:30. Each one takes 5 minutes, for total time of 240 minutes.
* Restore the tail-of-the-log backup. This takes 5 minutes.

In total, the restore sequence takes 270 minutes, or 4 hours 30 minutes. This is clearly longer than the downtime requirement.

The solution is to add differential backups during each day, say every 4 hours starting at midnight, with each one taking 10 minutes to complete.

The restore sequence then becomes:

* Perform the tail-of-the-log backup to backup the log generated since the log backup at 23:30. This takes 5 minutes.
* Restore the full backup from midnight. This takes 20 minutes.
* Restore the most recent differential backup from 20:00. This takes 10 minutes.
* Restore the 8 log backups from today, starting with the one at 20:00 and ending with the one at 23:30. Each one takes 5 minutes, for total time of 40 minutes.
* Restore the tail-of-the-log backup. This takes 5 minutes.

Now the total restore sequence takes 80 minutes, or 1 hour 20 minutes. This is within the downtime requirement.

USE Branch;

GO

SELECT file\_id,

name,

size as SizeInPages,

FILEPROPERTY(name, 'SpaceUsed') as SpaceUsedInPages,

size\*8/1024. as SizeInMB,

FILEPROPERTY(name, 'SpaceUsed')\*8/1024. as SpaceUsedInMB,

physical\_name

FROM sys.database\_files;

GO

EXEC sp\_spaceused 'dbo.Prospect', 'true'

use tools

exec SDU\_Tools.ListUserTableAndIndexSizes

N'branch'

--ESTIMATE SIZE OF A ROW IN A TABLE. VAR COLS NEED EXTRA HANNDLING NOT TAKEN CARE OF IN THIS QUERY

--https://docs.microsoft.com/en-us/sql/relational-databases/databases/estimate-the-size-of-a-heap

-- Row\_Size = Fixed\_Data\_Size + Variable\_Data\_Size + Null\_Bitmap + 4

;WITH CTE AS

(select OBJECT\_NAME(object\_id) as tablename, COUNT(1) as nr\_columns, sum(max\_length) as maxrowlength

from sys.columns as S

where object\_id = object\_id('FactInventory')

GROUP BY OBJECT\_NAME (object\_id)

)

SELECT \*, maxrowlength + 2 + ((nr\_columns + 7) / 8) + 4

FROM CTE

dbcc ind('natsoil','codes',1)—65 or 64 pages?

EXEC sp\_spaceused 'dbo.codes', 'true'--114 pages tot, 64 pages of data

set statistics io on

select \*

from codes--gives u 64 pages(logical reads)

USE Branch;

GO

SELECT i.data\_space\_id,

s.name as Filegroup

FROM sys.indexes AS i

INNER JOIN sys.data\_spaces AS s

ON i.data\_space\_id = s.data\_space\_id

WHERE i.object\_id = OBJECT\_ID(N'dbo.Contact');

GO

RESTORE HEADERONLY

FROM DISK = 'C:\mssqlDatabases\backups\AdventureWorks2012.bak';

GO

-- Step 6: Use RESTORE FILELISTONLY to get a list of files that are contained in the backup.

-- Mention that S means filestream backup and that filestream is an advanced

-- topic that is out of scope for this course.

RESTORE FILELISTONLY

FROM DISK = 'C:\mssqlDatabases\backups\AdventureWorks2012.bak';

GO

-- Step 7: Use RESTORE VERIFYONLY to check backup.

RESTORE VERIFYONLY

FROM DISK = 'C:\mssqlDatabases\backups\AdventureWorks2012.bak';

GO

EXEC sp\_configure 'show advanced options',1;

GO

RECONFIGURE;

GO

EXEC sp\_configure 'xp\_cmdshell',1;

GO

RECONFIGURE;

GO

select OBJECT\_NAME(t.object\_id), OBJECT\_NAME(t.stats\_id), \*

from sys.stats as t

where OBJECT\_NAME(t.object\_id) = N'Person'

dbcc show\_statistics([person.person], [IX\_Person\_LastName\_FirstName\_MiddleName])

select \*

from sys.dm\_exec\_connections

where session\_id = @@SPID

select SUSER\_SNAME(owner\_sid) as owner, \*

from sys.databases

exec sp\_helpdb

select \*

from sys.database\_files

select \*

from sys.objects

exec sp\_help

Primary file group is the default but does not have to be. Primary data file goes in PFG. For other files(secondary data files), if the file group is not specified, they go to default file group whichever that is.

Dbcc stands for database console command(during sysbase days it stood for database consistency checker)

dbcc checkdb

dbcc checkdb with estimateonly

dbcc checkdb with data\_purity

logs: windows event log, sql server error log. Recovery vs restore

select \*

from sys.server\_principals

select \*

from sys.database\_principals

select s.name as [Login Name], p.name as [User Name], default\_schema\_name

from sys.server\_principals as s

inner join sys.database\_principals as p

on s.sid = p.sid

EXEC sp\_dbcmptlevel

EXEC sp\_dbcmptlevel 'NatSoil'

select compatibility\_level

from sys.databases as s

where s.[name] = 'NatSoil'

--EXEC sp\_dbcmptlevel 'NatSoil', 120

--EXEC sp\_dbcmptlevel 'NatSoil', 110

--will work for stored procs, views, etc. but not tables. 'sys.databases' is a view

exec sp\_helptext 'sys.databases'

select OBJECT\_DEFINITION(OBJECT\_ID('sys.databases'))

exec sp\_help 'dbo.CODES'

--will not work..if you do not want to use sh\_help stored proc and want to use queries to

--get equivalent info, then write your own using sys.tables, sys.columns, etc.

select OBJECT\_DEFINITION(OBJECT\_ID('dbo.CODES'))

--better option than this is to query the sys.dm\_exec\_sessions dmv as shown below

dbcc useroptions

select \*

from sys.dm\_exec\_sessions

where session\_id = @@SPID

sysindexes

sysobjects

syscolumns

sysdatabases

sys.indexes

sys.objects

sys.columns

sys.databases

use tempdb

--create table t

--(

--id char(30),

--id1 char(30),

--id2 char(30),

--id3 char(30),

--)

select \*

from sys.indexes as i

where i.object\_id = object\_id('t')

select \*

from sys.partitions as p

where p.object\_id = object\_id('t')

--only contains 'in\_row\_data'

select a.\*

from sys.allocation\_units as a

inner join sys.partitions as p on a.container\_id = p.partition\_id

where p.object\_id = object\_id('t')

--alter table t

-- add id4 varchar(8000);

--rerun the 3 queries and see the sys.allocation\_units table contains 'in\_row\_data' and 'row\_overflow\_data'

--alter table t

-- add id5 varbinary(max);

--rerun the 3 queries and see the sys.allocation\_units table contains 'in\_row\_data', 'row\_overflow\_data' and 'lob\_data'

--create clustered index idx1 on t(id);

--rerun the 3 queries and see the type entry in indexes table changes to clustered

--create nonclustered index idx2 on t(id1);

--rerun the 3 queries and see that a new entry has been made in all three tables

dbcc traceon(3604)

dbcc ind(natsoil, samples, -1) –get the page number to feed into the command below

dbcc page(natsoil,1,22733,1)

--get index depth and index level. The index\_id param would be 0 for heap, 1 for CI and 2 or more for NCI. If null is passed, that means all indexes

SELECT

[index\_depth],

[index\_level],

[page\_count],

[record\_count]

FROM sys.dm\_db\_index\_physical\_stats (

DB\_ID (N'AdventureWorks2012'),

OBJECT\_ID (N'Test1'),

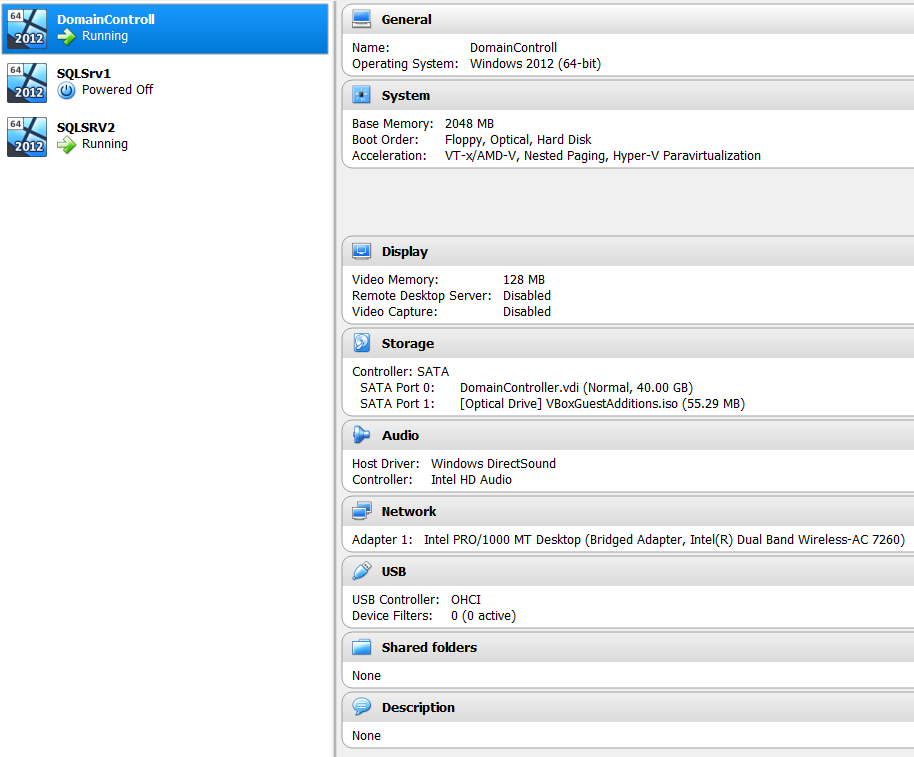
1,

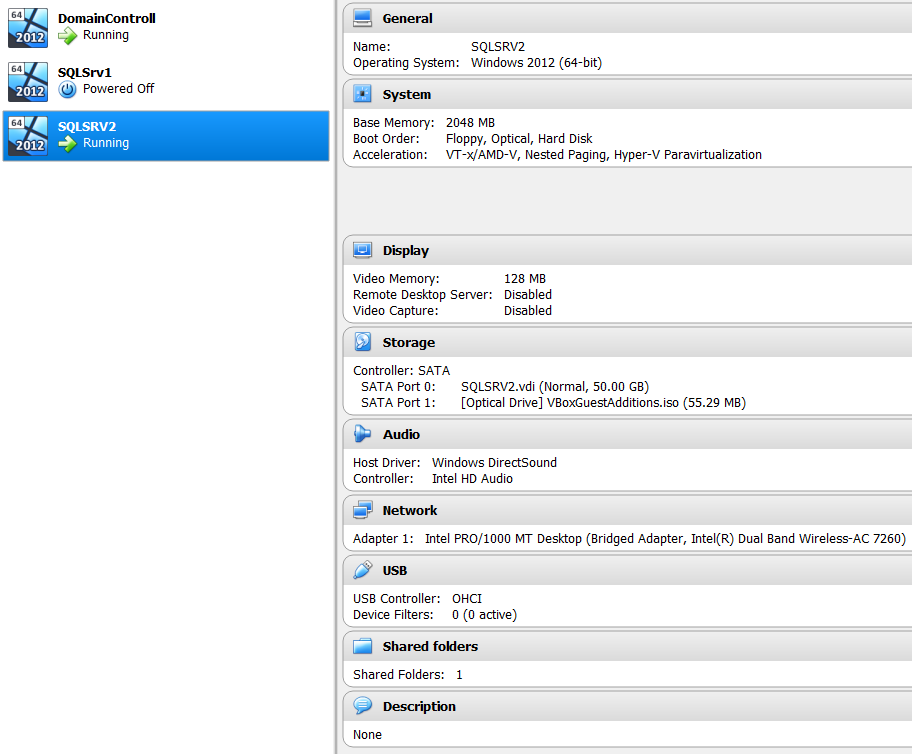
0,

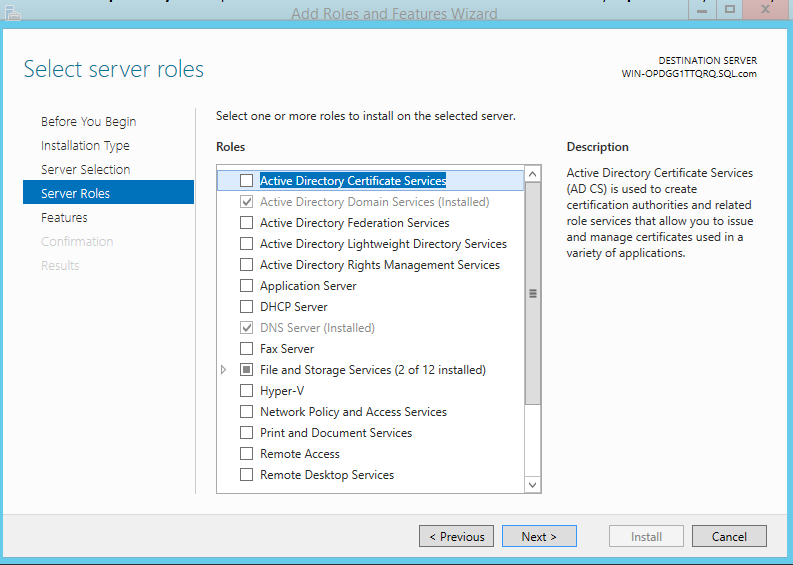
'DETAILED');

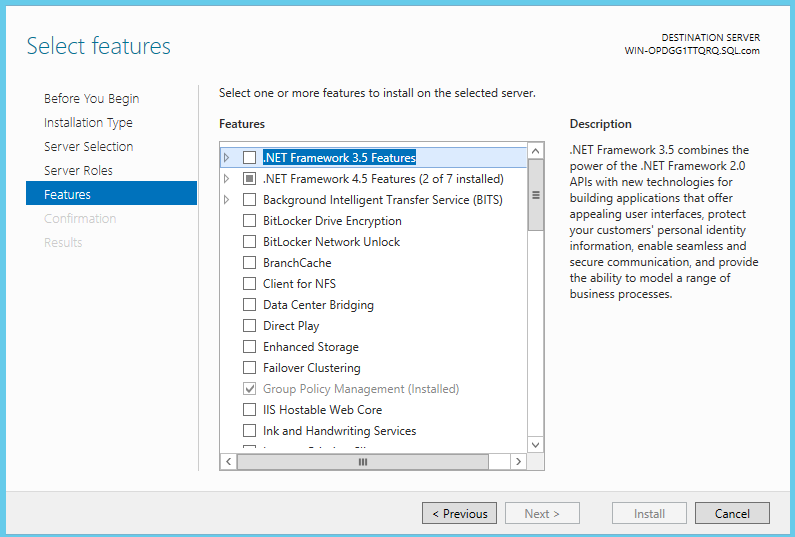
GO

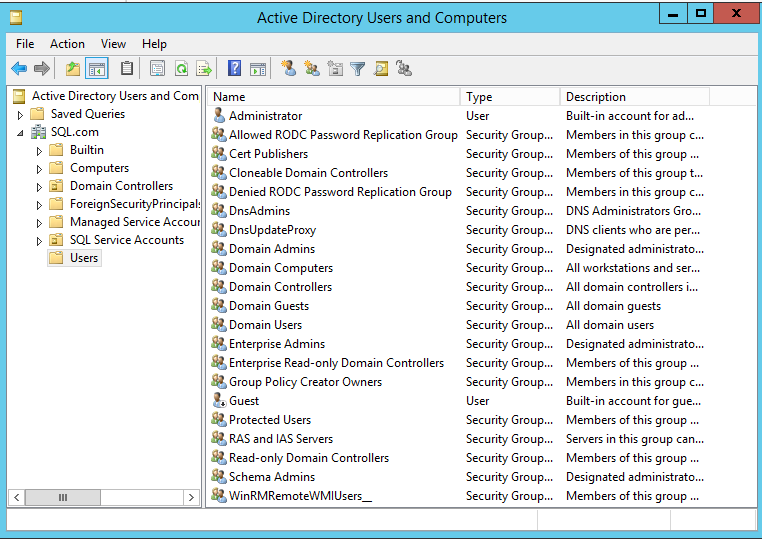
C:\Users\sin17h>wmic partition get BlockSize, StartingOffset, Name, Index

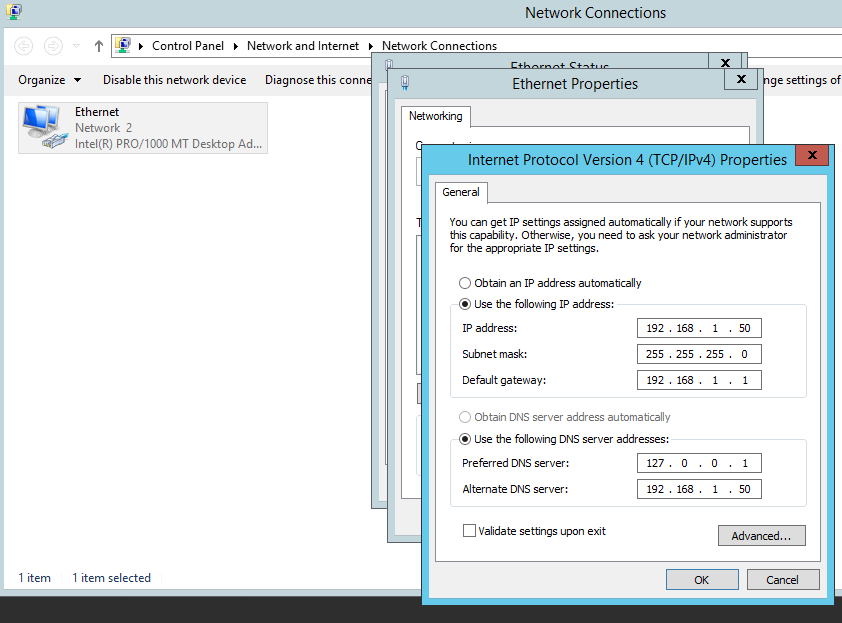


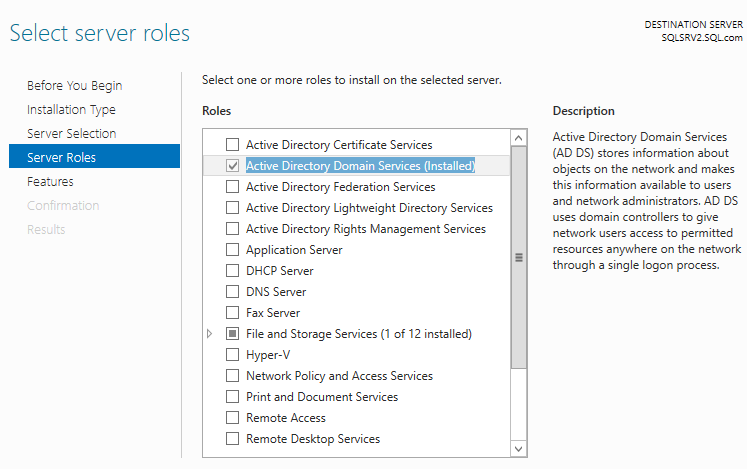


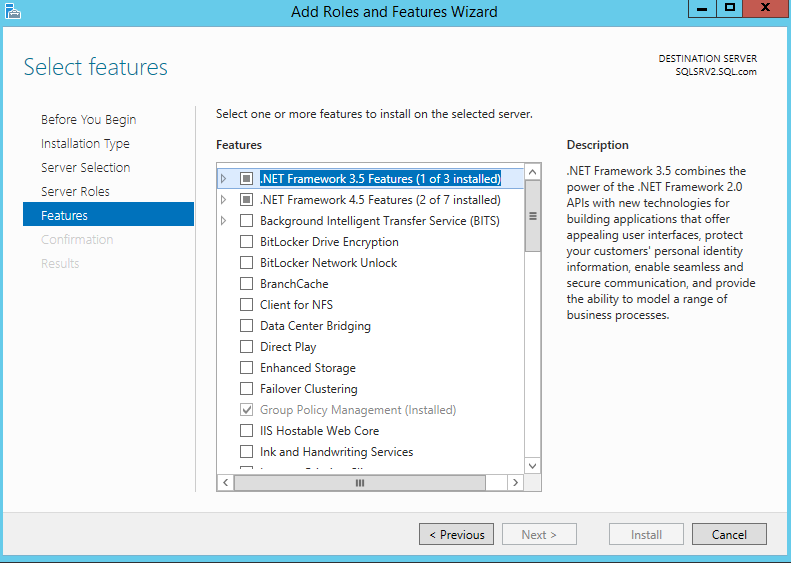


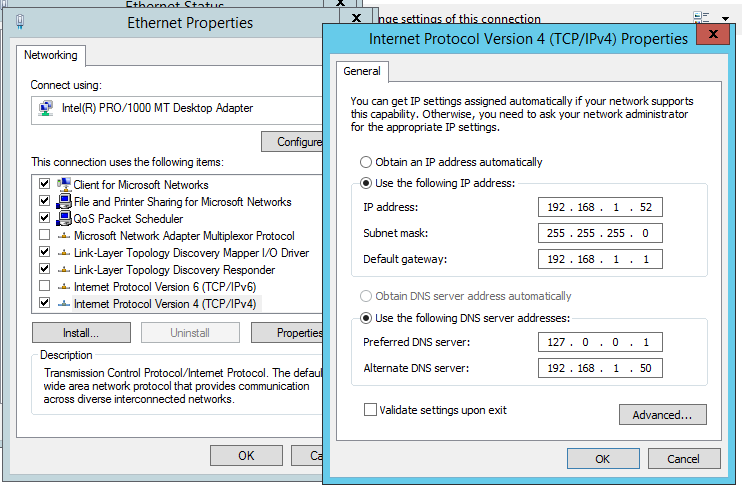




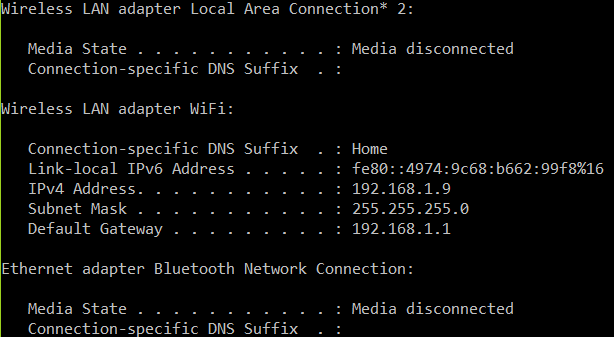


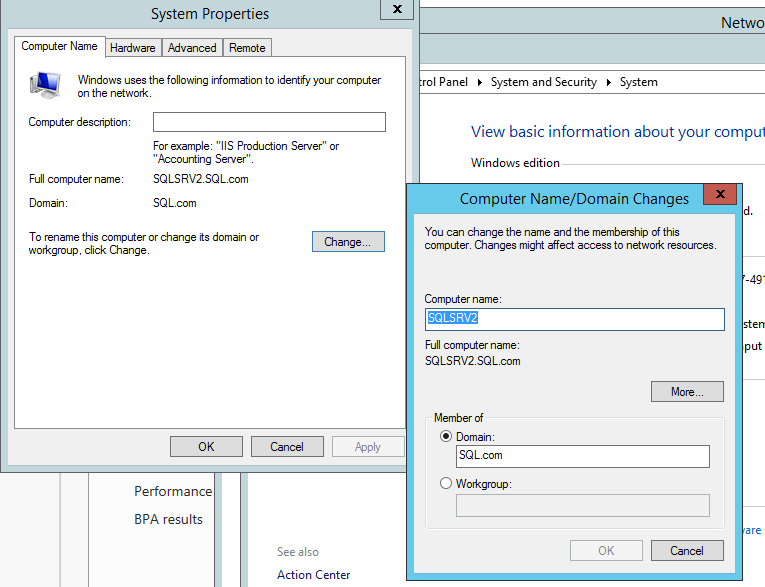


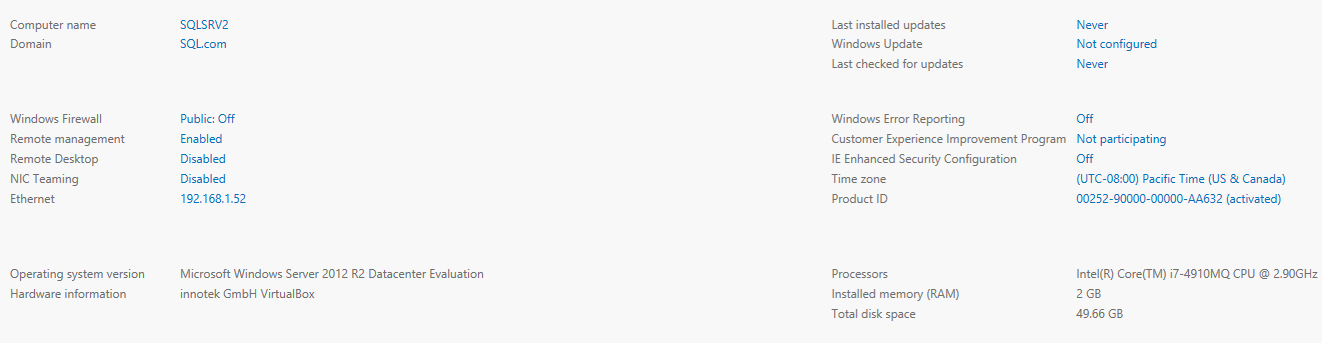




Default gateway from host machine







SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED ;

WITH XMLNAMESPACES

(DEFAULT 'http://schemas.microsoft.com/sqlserver/2004/07/showplan')

SELECT query\_plan AS CompleteQueryPlan ,

n.value('(@StatementText)[1]', 'VARCHAR(4000)') AS StatementText ,

n.value('(@StatementOptmLevel)[1]', 'VARCHAR(25)')

AS StatementOptimizationLevel ,

n.value('(@StatementSubTreeCost)[1]', 'VARCHAR(128)')

AS StatementSubTreeCost ,

n.query('.') AS ParallelSubTreeXML ,

ecp.usecounts ,

ecp.size\_in\_bytes

FROM sys.dm\_exec\_cached\_plans AS ecp

CROSS APPLY sys.dm\_exec\_query\_plan(plan\_handle) AS eqp

CROSS APPLY query\_plan.nodes

('/ShowPlanXML/BatchSequence/Batch/Statements/StmtSimple')

AS qn ( n )

WHERE n.query('.').exist('//RelOp[@PhysicalOp="Parallelism"]') = 1

Which solution do you use for SQL Server monitoring?

:one: Nothing (Ad-hoc queries) `4`

:two: No, really, nothing. `2`

:three: Redgate `6`

:four: SentryOne `6`

:five: Solarwinds `5`

:six: In-house solution `4`

And/Or Grafana/Telegraf, Grafana/InfluxDB used by MS Tiger Team, SO's Opserver. Configured Zabbix or Nagios aren't anywhere near "in-house" or "nothing" either :wink: (edited)

Which solution do you use for SQL Server Monitoring? (continued)

:one: dbareports

:two: Database Health Monitor `1`

:three: SQL Power Tools

:four: Idera `3`

:five: Heroix

:six: Zabbix `1`

--it invalidates the plan in the cache so that when the proc is called the next time, it will be recompiled. There could be multiple plans in cache for same proc for different session options

Exec Sp\_recompile GetCustInfo

--this also gives u the session options

select \*

from sys.dm\_exec\_sessions

where is\_user\_process = 1

The estimated number of rows can be different from actual number of rows in a plan due to 2 reasons:

1. The statistics are not up-to-date.
2. The plan that is being used was not compiled for that parameter value(due to parameter sniffing)

Run glenn berry’s [dmv queries](https://www.sqlskills.com/blogs/glenn/category/dmv-queries/) to collect stats:

PS C:\Windows\system32> $output = Invoke-DbaDiagnosticQuery -SqlInstance mssql-asris1.it.csiro.au –InstanceOnly

PS C:\Windows\system32> Export-DbaDiagnosticQuery -InputObject $output -ConvertTo Excel

PS C:\Windows\system32> $output = Invoke-DbaDiagnosticQuery -SqlInstance mssql-asris1.it.csiro.au

PS C:\Windows\system32> Export-DbaDiagnosticQuery -InputObject $output -ConvertTo Excel

<https://www.youtube.com/watch?v=BY2ZQTMGRDA>

--view cached plans

select distinct cp.bucketid, cp.cacheobjtype, cp.usecounts, co.sqlbytes, cp.size\_in\_bytes, co.[sql]

from sys.syscacheobjects as co

inner join sys.dm\_exec\_cached\_plans as cp on co.bucketid = cp.bucketid

select t.text, p.size\_in\_bytes, p.usecounts

from sys.dm\_exec\_cached\_plans as p

cross apply sys.dm\_exec\_sql\_text(p.plan\_handle) as t

fSELECT st.text, qs.execution\_count, qs.total\_elapsed\_time,

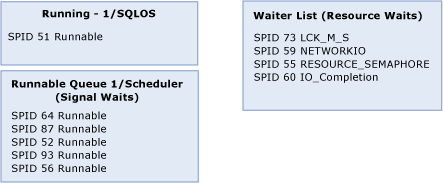
convert(decimal(11,2), qs.total\_elapsed\_time \* 1.0/qs.execution\_count) as avg\_time

FROM sys.dm\_exec\_query\_stats AS qs

CROSS APPLY sys.dm\_exec\_sql\_text(qs.sql\_handle) AS st

ORDER BY total\_logical\_reads/execution\_count DESC;

GO



The current wait list can be seen in sys.dm\_os\_waiting\_tasks. The current runnable queue is found in sys.dm\_exec\_requests where the status is “runnable”. The total time that is spent waiting in sys.dm\_os\_waiting\_tasks is found in the column wait\_time\_ms and the time that is spent waiting for CPU in the runnable queue is called signal\_wait \_time\_ms. Not only is the info about runnable queue exposed by sys.dm\_exec\_requests using wait\_type ‘runnable’ but info about threads waiting for resources is also exposed through sys.dm\_exec\_requests with wait\_type set to the particular resource wait. The one kind of resource waits not exposed through sys.dm\_exec\_requests is the ‘threadpool’ wait\_type as that request has not been assigned a thread and has not started executing. For that, we need to used dmv sys.dm\_os\_waiting\_tasks as it will expose both started and not-started tasks.

Useful DMVs for performance tuning purposes include sys.dm\_exec\_requests, sys.dm\_os\_waiting\_tasks, and sys.dm\_os\_wait\_stats. [sys.dm\_os\_wait\_stats (for the cumulative ones), sys.dm\_os\_waiting\_tasks for the current waits]

<https://www.youtube.com/watch?v=34VqSliEfsc>

