# SQL Server Performance Tuning checklist

By: Abdul Hameed Majeed

Table of Contents

[SQL Server Performance Tuning checklist 1](#_Toc13126244)

[1. Introduction 3](#_Toc13126245)

[2. Hardware Configuration 4](#_Toc13126246)

[3. Installation 5](#_Toc13126247)

[4. SQL Server Software Configuration 6](#_Toc13126248)

[5. TempDB 9](#_Toc13126249)

[6. Listing Most Expensive Queries and Store Procedures 10](#_Toc13126250)

[7. Indexes 11](#_Toc13126251)

[a. Clustered Indexes 11](#_Toc13126252)

[b. Non-Clustered Indexes 11](#_Toc13126253)

[c. Missing indexes 12](#_Toc13126254)

[d. Un-used indexes 13](#_Toc13126255)

[e. Covering Indexes 16](#_Toc13126256)

[f. Filtered Indexes 17](#_Toc13126257)

[g. Index on Computed Columns/ Views 17](#_Toc13126258)

[h. OLAP – On-Line Analytical Processing 18](#_Toc13126259)

[a. ColumnStore Indexes 18](#_Toc13126260)

[b. Spatial 19](#_Toc13126261)

[c. Full-text 19](#_Toc13126262)

[d. XML 19](#_Toc13126263)

[e. Memory-optimized non-clustered indexes 20](#_Toc13126264)

[f. Fragmentation 21](#_Toc13126265)

[10. Update Stats 22](#_Toc13126266)

[11. Page Splits 22](#_Toc13126267)

[12. Query Optimizer 23](#_Toc13126268)

[a. Optimize for Ad-Hoc WorkLoad 24](#_Toc13126269)

[13. Data Compression 25](#_Toc13126270)

[14. Table Variable Vs Temporary Variable Vs CTE 27](#_Toc13126271)

[15. In-Memory Tables 27](#_Toc13126272)

[q. Limitations 30](#_Toc13126273)

[16. Identifying poor performance Issues 32](#_Toc13126274)

[a. Locking/ Blocking & Deadlocking 32](#_Toc13126275)

[b. Waite Stats 35](#_Toc13126276)

[d. Statistics 36](#_Toc13126277)

[e. Backup & Maintenance plan to avoid performance issues 37](#_Toc13126278)

[17. Useful Tools 38](#_Toc13126279)

[a. Query Store 38](#_Toc13126280)

[b. Extended Events 39](#_Toc13126281)

[c. PerfMon 39](#_Toc13126282)

[d. SQL Profiler 41](#_Toc13126283)

[e. Database Engine Tuning Advisor 41](#_Toc13126284)

[f. SQL Server Default Trace 41](#_Toc13126285)

[g. Activity Monitor 43](#_Toc13126286)

[h. DMVs 44](#_Toc13126287)

[18. ISOLATION 45](#_Toc13126288)

[19. Stored Procedures: Interpreted Vs CLR Vs Natively Compiled 46](#_Toc13126289)

[a. Interpreted/ Standard Stored Procedure 46](#_Toc13126290)

[b. CLR/ Extended Stored Procedure 47](#_Toc13126291)

[c. Natively compiled stored procedures 49](#_Toc13126292)

[20. SQL Standard Reports 51](#_Toc13126293)

[21. Execution plans 52](#_Toc13126294)

[22. Data Collection 53](#_Toc13126295)

[23. Log Files 54](#_Toc13126296)

[24. Good Table Design 55](#_Toc13126297)

[25. Achieving high availability of SQL Server in case of disaster 55](#_Toc13126298)

[26. Summary 55](#_Toc13126299)

[27. SSIS 57](#_Toc13126300)

[a. Balance Data Distributor 57](#_Toc13126301)

[b. Parallel Processing 57](#_Toc13126302)

[c. Conditional Split 57](#_Toc13126303)

[d. Cache Transform 57](#_Toc13126304)

[e. Merge 57](#_Toc13126305)

[f. Merge Join 57](#_Toc13126306)

[g. MultiCast 58](#_Toc13126307)

[h. OLEDB command 58](#_Toc13126308)

[i. Archiving Old data 59](#_Toc13126309)

[j. Avoid Implicit Data Conversion 59](#_Toc13126310)

[k. SCD 59](#_Toc13126311)

[l. Avoid using sort component 59](#_Toc13126312)

[m. Create cubes for reporting to improve performance 59](#_Toc13126313)

[n. Choosing transaction 60](#_Toc13126314)

[o. CDC is much faster than SCD 60](#_Toc13126315)

[p. Buffer size tuning 60](#_Toc13126316)

[q. Three types of components to aid performance? 60](#_Toc13126317)

[r. Synchronous, Asynchronous, Full Blocking, Semi Blocking, non Blocking 60](#_Toc13126318)

[s. Implementing multithreading 60](#_Toc13126319)

[t. Partitioning 61](#_Toc13126320)

[u. Page split performance issues with SSIS 61](#_Toc13126321)

[v. Cube Sub Aggregation 61](#_Toc13126322)

[w. What are the different caching modes in Lookup 61](#_Toc13126323)

[x. Create Statistics Stat1 62](#_Toc13126324)

[y. Improving Bulk Insert performance 62](#_Toc13126325)

[z. Cube Architecture 62](#_Toc13126326)

[28. Index A: Useful Link 62](#_Toc13126327)

[29. DBCCs (Database Console Commands) 62](#_Toc13126328)

[30. Index B: Useful System Stored Procedures 63](#_Toc13126329)

[31. Index C: Useful SQLs 63](#_Toc13126330)

## Introduction

* 1. Purpose of this document is to list the most common performance tuning techniques for SQL Server database systems, with various diagnosis and implementation techniques to bring about performance improvements.
  2. It is said that 80% of problems can be solved by doing 20% of activities and the most common issues can be resolved by looking into following four topics:
     1. Listing Most Expensive Queries and Store Procedures
     2. Indexes
     3. Update Stats
     4. Page splits

## Hardware Configuration

* 1. Always ensure there is sufficient resources available for the application. Always consider scalability. Being able to expand for future business growth is important.  Specification of a new system depends on the following:
     1. size and number of the databases in use,
     2. the number of users,
     3. the existing infrastructure
     4. Rate of Database Growth
     5. Dedicated or Shared
     6. OS Requirements: Always reserve 1 GB of RAM for the OS, plus an additional 1 GB for each 4 GB
  2. Disk Drives
     1. Preferably high speed SAN disk, separate storage for SQL log, data, and temp files.
     2. If you're mostly reads (90+ %) Raid5 is also fine. If your database is write heavy (50+ %) Raid10
     3. Use Perfmon to measure the disk access speed counters
        1. Physical Disk\Disk Sec/Reads
        2. Physical Disk\Disk Sec/Writes
        3. <10 ms = good performance
        4. 10 to 20 ms = slow performance
        5. 20 to 50 ms = Poor performance
     4. Error 1101 or 1105 is returned when db runs out of disk space.
     5. Error 9002 is returned when transaction log runs out of disk space.
  3. CPU
     1. 8 CPU cores with a high-performing processors should be sufficient for most OLTP system.
     2. See Perfmon section for CPU measureable counters.
  4. Memory
     1. 16-64 GB RAM should be sufficient for most OLTP systems
     2. The following are the SQL Server components within the SQL instance that use memory from the buffer pool
        1. Database Page Cache
        2. Internal log caches
        3. Procedure cache or query plan cache
        4. Query Workload space
        5. Locks (Memory grants)
        6. Connection context
        7. Optimizing queries
        8. System-level data structures
     3. The default setting for SQL Server’s Max Memory is less harmful in SQL Server 2016 and higher. Always leave some memory for the OS and other apps. Total memory minus 10% or 20% for OS & other Apps.

--- Current Max Memory settings

SELECT \* FROM sys.configurations

WHERE name = 'Max Server Memory (MB)'

---- 2147483647 this is the default value which means no upper limit

SP\_CONFIGURE 'Max Server Memory’, 2147483647

GO

RECONFIGURE

GO

* + 1. Network (NIC)
       1. By pinging the SQL Server from client PC -> anything more than 10ms is slow
       2. Network packets is 8000 bytes by default

select wait\_time\_ms / (case waiting\_tasks\_count when 0 then NULL else waiting\_tasks\_count end) as AvgWaitMS, \*

from sys.dm\_os\_wait\_stats

where wait\_type = 'ASYNC\_NETWORK\_IO'

go

## Installation

* 1. Always install data, log, tempdb, backup files and index files on separate drives for optimum performance.
  2. Use SAN disk storage where possible -> Storage Area Network -> External SAN.
  3. Auto-growth -> SQL server holds up database processing while auto-growth event occurs. This can cause poor performance. Leads to physical fragmentation and autogrowth increment. Therefore this should be set to a sensible value based on their growth profile.
  4. Monitor auto-growth events.

-- Drop temporary table if it exists

IF OBJECT\_ID('tempdb..#info') IS NOT NULL

       DROP TABLE #info;

-- Create table to house database file information

CREATE TABLE #info (

     databasename VARCHAR(128)

     ,name VARCHAR(128)

    ,fileid INT

    ,filename VARCHAR(1000)

    ,filegroup VARCHAR(128)

    ,size VARCHAR(25)

    ,maxsize VARCHAR(25)

    ,growth VARCHAR(25)

    ,usage VARCHAR(25));

-- Get database file information for each database

SET NOCOUNT ON;

INSERT INTO #info

EXEC sp\_MSforeachdb 'use ?

select ''?'',name,  fileid, filename,

filegroup = filegroup\_name(groupid),

''size'' = convert(nvarchar(15), convert (bigint, size) \* 8) + N'' KB'',

''maxsize'' = (case maxsize when -1 then N''Unlimited''

else

convert(nvarchar(15), convert (bigint, maxsize) \* 8) + N'' KB'' end),

''growth'' = (case status & 0x100000 when 0x100000 then

convert(nvarchar(15), growth) + N''%''

else

convert(nvarchar(15), convert (bigint, growth) \* 8) + N'' KB'' end),

''usage'' = (case status & 0x40 when 0x40 then ''log only'' else ''data only'' end)

from sysfiles

';

-- Identify database files that use default auto-grow properties

SELECT databasename AS [Database Name]

      ,name AS [Logical Name]

      ,filename AS [Physical File Name]

      ,growth AS [Auto-grow Setting] FROM #info

WHERE (usage = 'data only' AND growth = '1024 KB')

   OR (usage = 'log only' AND growth = '10%')

ORDER BY databasename

-- get rid of temp table

DROP TABLE #info;

--- Change the Auto-growth setting using

ALTER DATABASE MyDB

  MODIFY FILE

  (NAME=MyDB\_Log,FILEGROWTH=20MB);

## SQL Server Software Configuration

1. You should set the **Cost Threshold for parallelism** = 50 where the default is set to 5. Specify the threshold at which Microsoft SQL server creates and runs parallel plans for queries. Set this property to 50, so that more of simpler queries run on a single thread. There’s no right or wrong number, the default setting 5 is low. It’s appropriate for purely OLTP applications.
2. Set the Optimize for **Ad hoc Workloads option to 1(true)**. This option is used to improve the efficiency of the plan cache for workloads that contain many single use ad hoc batches. When this option is set to 1, the database engine stores a small compiled plan stub in the plan cache when a batch is compiled for the first time, instead of full compiled plan.
3. **Backup Compression ->** From SQL Server 2008R2, you can tick a check box for backup compression. This means backups reduced, they will take less time, and the restores will be quicker.
4. Setting the **Max Degree of Parallelism** option to 0 will allow the SQL Server Engine to use all available CPUs up to 64 processors in parallel plan execution. You can specify any value between 2 and 32,767. If the value is greater than the actual number of processors, then the actual number of processors will be used in the parallel plan execution that is available. If the SQL Server instance is hosted on a single processor server, the Max Degree of Parallelism value will be ignored.
5. Choosing relevant data collation ->
   * 1. Collation means assigning some order to the characters in an Alphabet, ASCII or Unicode etc. It is performed at Server, Database, Column or expression level collation
     2. Choosing the wrong collation may affect your database performance.
     3. SQL collations should provide better performance than Windows collations for non-Unicode types due to simpler comparison rules but the difference is significant only in the most severe circumstances, such as a table scan with LIKE '% %' in WHERE clause.
     4. Binary collations are said to provide the best performance but the cost of unnatural (non-dictionary) comparisons and sort order is high; most users would expect 'a' to sort before 'B' but that is not the case with binary collations.
     5. Binary collation -> sort & compare data based on binary values

--check collation type used

SELECT name, description

  FROM sys.fn\_helpcollations();

SELECT SERVERPROPERTY('collation');

SELECT DATABASEPROPERTYEX('TestCollations', 'Collation');

---Sets collation

ALTER DATABASE TestCollations

  COLLATE Latin1\_General\_100\_CI\_AS;

  GO

1. Buffer pool Extension (BPE) Introduce in SQL Server 2014
   * 1. The idea behind BPE is like virtual RAM (better known as swap space): fast, low latency persistent storage is used to replace a portion of memory.
     2. Buffer pool is extended with Solid State Drive (SSD) -> Solid-state drives store data in memory (RAM) in a persistent manner.
     3. Only clean pages are written to BPE, hence no data loss
     4. Increase random I/O performance with reduce latency
     5. Waite type used while waiting for BPE I/Os to complete is ‘EC’
     6. Can significantly slow down performance of large serial range scans
     7. sequential scans over contiguous data (not overly fragmented indexes) can be slow
     8. Worker threads read one page (8kb) at a time from the BPE.
     9. Read a page, process page, read next page
     10. In summary, we can say that buffer pool extension introduces another level of cache. Whenever the page is evicted from buffer pool, it goes and sits in buffer pool extension file. The next request of the same page will not be a physical IO from the MDF/NDF file, but would be read from the BPE file, which should reside in SSD, which is faster than normal spinning based media. The latency and read/write speed of SSD is faster than traditional disks and hence would give performance improvement.
     11. https://docs.microsoft.com/en-us/sql/database-engine/configure-windows/buffer-pool-extension?view=sql-server-2017 -> more read

--Review current BPE configuration

SELECT [path], state\_description, current\_size\_in\_kb,

CAST(current\_size\_in\_kb/1048576.0 AS DECIMAL(10,2)) AS [Size (GB)]

FROM sys.dm\_os\_buffer\_pool\_extension\_configuration;

--Reduce SQL Server Max memory to restrict the BP and force the use of BPE

EXEC sys.sp\_configure 'show advanced options', '1' RECONFIGURE WITH OVERRIDE;

GO

EXEC sys.sp\_configure 'max server memory (MB)', '2000';

GO

RECONFIGURE WITH OVERRIDE;

GO

EXEC sys.sp\_configure 'show advanced options', '0' RECONFIGURE WITH OVERRIDE;

GO

/\*\*\*\*\*\*\*\*\*\*\* WE HAVE TO RESTART SQL SERVER FOR BPE TO SEE THE RAM CHANGE \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

--Enable BPE

--Go look at the file size on disk right after you run this

ALTER SERVER CONFIGURATION

SET BUFFER POOL EXTENSION ON (FILENAME = 'C:\Temp\BP\_Extension.BPE', SIZE = 2 GB);

--Now that it is enabled we'll inspect the configuration again

SELECT [path], state\_description, current\_size\_in\_kb,

CAST(current\_size\_in\_kb/1048576.0 AS DECIMAL(10,2)) AS [Size (GB)]

FROM sys.dm\_os\_buffer\_pool\_extension\_configuration;

--Try to read enough data to fill BP and start using BPE

USE AdventureworksDW2016CTP3;

GO

SELECT \* FROM dbo.FactResellerSalesXL\_PageCompressed;

--If the above didn't do the trick then query this table as well

--SELECT \* FROM dbo.FactResellerSalesXL\_CCI;

--Let's see what went to BPE. If there are no results then go query more data.

SELECT DB\_NAME(database\_id) AS [Database Name], COUNT(page\_id) AS [Page Count],

CAST(COUNT(\*)/128.0 AS DECIMAL(10, 2)) AS [Buffer size(MB)],

AVG(read\_microsec) AS [Avg Read Time (microseconds)]

FROM sys.dm\_os\_buffer\_descriptors

WHERE database\_id <> 32767

AND is\_in\_bpool\_extension = 1

GROUP BY DB\_NAME(database\_id)

ORDER BY [Buffer size(MB)] DESC;

--Turn BPE off. Go look in c:\temp to see what happens to the physical data file

ALTER SERVER CONFIGURATION

SET BUFFER POOL EXTENSION OFF;

--Put Max Server Memory back where it was

EXEC sys.sp\_configure 'show advanced options', '1' RECONFIGURE WITH OVERRIDE;

GO

EXEC sys.sp\_configure 'max server memory (MB)', '3500';

GO

RECONFIGURE WITH OVERRIDE;

GO

EXEC sys.sp\_configure 'show advanced options', '0' RECONFIGURE WITH OVERRIDE;

GO

----DBCC command which can drop the clean buffers called as DBCC DROPCLEANBUFFERS

## TempDB

1. TempDB configuration is a repeat offender when it comes to performance bottlenecks. Look out for multiple data files and check that trace flags 1117 and 1118 are enabled. For SQL Server 2016, trace flags 1117 and 1118 are automatically enabled.
2. PAGELATCH\_XX wait types are an **indicator** of tempdb contention
3. Place your tempdb files on dedicated high-speed disks
4. Grow your tempdb data files to a desired size
5. Set your files to grow equally
6. Simple recovery model for Tempdb
7. It is recreated every time SQL Server is restarted
8. Row versions -> Isolation Snapshot
9. Backup & Restore is not allowed on TEMPDB
10. TempDB busy database -> Should be configured correctly which holds
    1. Global or local temp tables
    2. Temp SP
    3. Table variables
    4. Cursors
    5. Results for spools or Sorting
    6. On-line index operations
    7. Multiple active result sets
    8. After triggers

--Lists all tempdb files

use master

select \* from sys.master\_files where database\_id = db\_id('tempdb')

go

--Lists Trace flag T1118 status

dbcc tracestatus()

go

1. Allocation Contention
   * 1. **Trace Flag 1118 – sets Full Extents Only:**
        1. That trace flag tells SQL Server that it should avoid “mixed extents” and use “full extents”. This means that each newly allocated object in every database on the instance gets its own private 64KB of data.
        2. This trace flag is listed in KB 2154845, it’s documented as safe to use. But it hasn’t made its way into the list of Trace Flags in SQL Server Books Online.

--File no’s should be same as CPU core from SQL Server 2016+

Select \* from tempdb.sys.tables/ db\_files

ALTER Database [tempdb] modify file (NAME=tempdev, Filename=’T:\temp.mdf’, size=512000KB, FileGrowth=1024000KB0

ALTER Database [tempdb] modify file (NAME=tempdev, Filename=’T:\templog.ldf’, size=512000KB, FileGrowth=1024000KB0

--Autogrowth should be number instead of %

Exec Sp\_helpfile -> current size & location

SELECT cpu\_count FROM sys.dm\_os\_sys\_info -> CPU count

--To add more temp files:

ALTER Database [tempdb] modify file (NAME=tempdev2, Filename=’T:\temp2.ndf’, size=512000KB, FileGrowth=1024000KB0

1. Enable **IFI (Instant File Initialization)** -> Larger the growth operation, the more noticeable the performance improvement is with IFI enabled. For instance, a data file growing by 20 GB can take minutes to initialize without IFI. This can make a huge difference whenever you are proactively growing out data files.
2. Data and log file initialisation happens when:
   1. Create a database.
   2. Add data or log files, to an existing database.
   3. Increase the size of an existing file (including autogrow operations).
   4. Restore a database or filegroup.
   5. IFI has to be enabled from Windows Security Policy. Steps are as follows:
      1. Start Menu -> run secpol.msc -> Local Security Policy -> Expand the Local Policies Folder -> Click on User Rights Assignment -> Go to the Perform Volume Maintenance Tasks option -> Double click it -> Add your SQL Server Service account -> click OK
      2. Restart your SQL Server services
      3. <https://docs.microsoft.com/en-us/sql/relational-databases/databases/database-instant-file-initialization?view=sql-server-2017> -> More reads
3. P336

## Listing Most Expensive Queries and Store Procedures

* 1. Identify which queries and stored procedures that have the most IO, CPU usage and are taking the longest to run. Working on these will give you the best performance improvement.

---- Top 10 Expensive queires

SELECT

TOP 10 SUBSTRING(qt.TEXT, (qs.statement\_start\_offset/2)+1,

((CASE qs.statement\_end\_offset

WHEN -1 THEN DATALENGTH(qt.TEXT)

ELSE qs.statement\_end\_offset

END - qs.statement\_start\_offset)/2)+1),

qs.execution\_count,

qs.total\_logical\_reads, qs.last\_logical\_reads,

qs.total\_logical\_writes, qs.last\_logical\_writes,

qs.total\_worker\_time,

qs.last\_worker\_time,

qs.total\_elapsed\_time/1000000 total\_elapsed\_time\_in\_S,

qs.last\_elapsed\_time/1000000 last\_elapsed\_time\_in\_S,

qs.last\_execution\_time,

qp.query\_plan

FROM

sys.dm\_exec\_query\_stats qs

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

CROSS APPLY sys.dm\_exec\_query\_plan(qs.plan\_handle) qp

ORDER BY

qs.total\_logical\_reads DESC -- logical reads

-- ORDER BY qs.total\_logical\_writes DESC -- logical writes

-- ORDER BY qs.total\_worker\_time DESC -- CPU time

## Indexes

* + Data Access - Determine how the data is being accessed (SELECT) and maintained (INSERT, UPDATE, DELETE) then index accordingly.  No index on table is known as a Heap table with a series of 8k pages.
  + Plan - Have a process to analyze the indexes versus haphazardly reviewing tables or T-SQL code to build your indexing strategy.
  + Index selection - Identify a clustered index and one or more non clustered indexes for each table.  During the testing and validation phases, fine tune or eliminate the indexes based on the data access.
  + Covering indexes - Have index with numerous columns to improve the data access and to avoid bookmark lookup operations
  + ASC vs. DESC order - The order of the indexes can make a significant difference in the data access for covering indexes.
  + Fill Factor - Depending on how the data is maintained (INSERT, UPDATE, DELETE), the fill factor could significantly impact performance, page splits and storage requirements.
  + Balance - Be sure to have relevant indexes, not excessive or duplicating.
  + Best column candidate are: sequential, distinct values, quarried based on range values (<=,>=, between), Where, Join, Group By, Order By, etc.
  + Primary key is the Default clustered index unless specified otherwise
  + Index based on a single column known as simple, index based on several columns known as Composite key.
  + Primary key constraint does not allow NULLS but UNIQUE constraint & indexes do

1. Clustered Indexes
   * 1. Clustered indexes are the cornerstone of good database design. A poorly-chosen clustered index leads to high execution times, causing wasted disk space, poor IO, heavy fragmentation etc.
     2. Leaf nodes in tree contains actual data in table
     3. efficient clustered index key, which are:
        1. Narrow – as narrow as possible, in terms of the number of bytes it stores
        2. Unique
        3. Static – ideally, never updated
        4. Ever-increasing – to avoid fragmentation and improve write performance
2. Non-Clustered Indexes
   * 1. copy of data in leaf node is stored in separate structure along with pointer to heap or clustered index
     2. Pointers are used to locate the actual row in source table
     3. Excellent for queries that don’t return large result set
     4. For Columns that are frequently used in WHERE clause
     5. For columns that have many distinct values
     6. Too many will slow down data modifications
     7. Covered indexes specially fast and efficient → Multiple columns within critical query → All required columns are stored in the index table → saves the step of going to underlying data source table
     8. The nonclustered index is faster for data insertion and update operation. It does not affect the physical order of records because it create logical order.
     9. You can have 999 indexes or 1700 byte

---- create nonclustered filtered index. Only nonclustered index ---- support WHERE clause. Use INCLUDE keyword to include columns ---- used in select statement, if reduce index key size.

CREATE NONCLUSTERED INDEX Ix\_Consumer\_CreateDate ON Consumer(CreateDate) INCLUDE(CustomerId, CountryCode, City) WHERE Status = 1;

---- Index information

SELECT type\_desc, is\_primary\_key, is\_unique, is\_unique\_constraint FROM sys.Indexes

### Missing indexes

* 1. Having a large number of indexes can be detrimental to DML transactions. Appropriate indexing means only creating indexes which have a positive impact to your read performance that outweigh the negative impact on write performance.
  2. To find missing indexes use the following DMVs:
     1. **sys.dm\_db\_missing\_index\_group\_stats**: use to retrieve metrics on group of missing indexes.
     2. **sys.dm\_db\_missing\_index\_groups**: use this DMV as an intermediary between **sys.dm\_db\_missing\_index\_details** and **sys.dm\_db\_missing\_index\_group\_stats**.
     3. **sys.dm\_db\_missing\_index\_details**: use this DMV to identify the columns used for equality and inequality predicates.

---- This script will help you identify missing indexes:

SELECT

migs.avg\_total\_user\_cost \* (migs.avg\_user\_impact / 100.0) \* (migs.user\_seeks + migs.user\_scans) AS improvement\_measure,

'CREATE INDEX [missing\_index\_' + CONVERT (varchar, mig.index\_group\_handle) + '\_' + CONVERT (varchar, mid.index\_handle)

+ '\_' + LEFT (PARSENAME(mid.statement, 1), 32) + ']'

+ ' ON ' + mid.statement

+ ' (' + ISNULL (mid.equality\_columns,'')

+ CASE WHEN mid.equality\_columns IS NOT NULL AND mid.inequality\_columns IS NOT NULL THEN ',' ELSE '' END

+ ISNULL (mid.inequality\_columns, '')

+ ')'

+ ISNULL (' INCLUDE (' + mid.included\_columns + ')', '') AS create\_index\_statement,

migs.\*, mid.database\_id, mid.[object\_id]

FROM

sys.dm\_db\_missing\_index\_groups mig

INNER JOIN sys.dm\_db\_missing\_index\_group\_stats migs ON migs.group\_handle = mig.index\_group\_handle

INNER JOIN sys.dm\_db\_missing\_index\_details mid ON mig.index\_handle = mid.index\_handle

WHERE

migs.avg\_total\_user\_cost \* (migs.avg\_user\_impact / 100.0) \* (migs.user\_seeks + migs.user\_scans) > 10

ORDER BY

migs.avg\_total\_user\_cost \* migs.avg\_user\_impact \* (migs.user\_seeks + migs.user\_scans) DESC

### Un-used indexes

----The following script list all unused indexes:

IF OBJECT\_ID('tempdb..#Results') IS NOT NULL

DROP TABLE #Results;

CREATE TABLE [dbo].#Results(

[Server Name] [nvarchar](128) NULL,

[DB Name] [nvarchar](128) NULL,

[source] [varchar](10) NOT NULL,

[objectname] [nvarchar](128) NULL,

[object\_id] [int] NOT NULL,

[indexname] [sysname] NULL,

[data\_compression] [varchar](24) NOT NULL,

[index\_id] [int] NOT NULL,

[rowcnt] [bigint] NULL,

[datapages] [bigint] NULL,

[is\_unique] [bit] NULL,

[count] [int] NULL,

[user\_seeks] [bigint] NOT NULL,

[user\_scans] [bigint] NOT NULL,

[user\_lookups] [bigint] NOT NULL,

[user\_updates] [bigint] NOT NULL,

[total\_usage] [bigint] NOT NULL,

[%Reads] [bigint] NULL,

[%Writes] [bigint] NULL,

[%Seeks] [bigint] NULL,

[%Scans] [bigint] NULL,

[%Lookups] [bigint] NULL,

[%Updates] [bigint] NULL,

[last\_user\_scan] [datetime] NULL,

[last\_user\_seek] [datetime] NULL,

[run\_time] [datetime] NOT NULL

) ON [PRIMARY]

EXECUTE sys.sp\_MSforeachdb

'USE [?];

declare @dbid int

select @dbid = db\_id()

INSERT INTO #Results

SELECT @@SERVERNAME AS [Server Name]

, db\_name() AS [DB Name]

, ''Usage Data'' ''source''

, objectname=object\_name(s.object\_id)

, s.object\_id, indexname=i.name

, data\_compression\_desc, i.index\_id

, s2.rowcnt, sa.total\_pages, is\_unique

, (select count(\*)

from sys.indexes r

where r.object\_id = s.object\_id) ''count''

, user\_seeks, user\_scans, user\_lookups, user\_updates, user\_seeks + user\_scans + user\_lookups + user\_updates AS [total\_usage]

, CAST(CAST(user\_seeks + user\_scans + user\_lookups AS DEC(12,2))/CAST(REPLACE((user\_seeks + user\_scans + user\_lookups + user\_updates), 0, 1) AS DEC(12,2)) \* 100 AS DEC(5,2)) [%Reads]

, CAST(CAST(user\_updates AS DEC(12,2))/CAST(REPLACE((user\_seeks + user\_scans + user\_lookups + user\_updates), 0, 1) AS DEC(12,2)) \* 100 AS DEC(5,2)) [%Writes]

, CAST(CAST(user\_seeks AS DEC(12,2))/CAST(REPLACE((user\_seeks + user\_scans + user\_lookups + user\_updates), 0, 1) AS DEC(12,2)) \* 100 AS DEC(5,2)) [%Seeks]

, CAST(CAST(user\_scans AS DEC(12,2))/CAST(REPLACE((user\_seeks + user\_scans + user\_lookups + user\_updates), 0, 1) AS DEC(12,2)) \* 100 AS DEC(5,2)) [%Scans]

, CAST(CAST(user\_lookups AS DEC(12,2))/CAST(REPLACE((user\_seeks + user\_scans + user\_lookups + user\_updates), 0, 1) AS DEC(12,2)) \* 100 AS DEC(5,2)) [%Lookups]

, CAST(CAST(user\_updates AS DEC(12,2))/CAST(REPLACE((user\_seeks + user\_scans + user\_lookups + user\_updates), 0, 1) AS DEC(12,2)) \* 100 AS DEC(5,2)) [%Updates]

, last\_user\_scan

, last\_user\_seek

, getdate() run\_time

from sys.dm\_db\_index\_usage\_stats s

join sys.indexes i on i.object\_id = s.object\_id

and i.index\_id = s.index\_id

join sysindexes s2 on i.object\_id = s2.id

and i.index\_id = s2.indid

join sys.partitions sp on i.object\_id = sp.object\_id

and i.index\_id = sp.index\_id

join sys.allocation\_units sa on sa.container\_id = sp.hobt\_id

where objectproperty(s.object\_id, ''IsUserTable'') = 1

and database\_id = @dbid'

EXECUTE sys.sp\_MSforeachdb

'USE [?];

declare @dbid int

select @dbid = db\_id()

INSERT INTO #Results

SELECT @@SERVERNAME

, db\_name()

, ''NA''

, object\_name(i.object\_id)

, o.object\_id

, i.name

, data\_compression\_desc

, i.index\_id

, s2.rowcnt

, sa.total\_pages

, is\_unique

, (select count(\*)

from sys.indexes r

where r.object\_id = i.object\_id) ''count''

, 0

, 0

, 0

, 0

, 0

, 0

, 0

, 0

, 0

, 0

, 0

, 0

, 0

, getdate()

FROM sys.indexes i

JOIN sys.objects o

ON i.object\_id = o.object\_id

join sysindexes s2 on i.object\_id = s2.id

and i.index\_id = s2.indid

join sys.partitions sp on i.object\_id = sp.object\_id

and i.index\_id = sp.index\_id

join sys.allocation\_units sa on sa.container\_id = sp.hobt\_id

WHERE OBJECTPROPERTY(o.object\_id,''IsUserTable'') = 1

AND i.index\_id NOT IN (

SELECT s.index\_id

FROM sys.dm\_db\_index\_usage\_stats s

WHERE s.object\_id = i.object\_id

AND i.index\_id = s.index\_id

AND database\_id = @dbid)

--AND i.index\_id NOT IN (0,1)'

SELECT \*

FROM #Results

WHERE [DB Name] NOT IN ('MASTER', 'msdb', 'MODEL', 'TEMPDB')

DROP TABLE #Results;

/\*

declare @dbid int

select @dbid = db\_id()

SELECT @@SERVERNAME AS [Server Name]

, db\_name() AS [DB Name]

, 'Usage Data' 'source'

, objectname=object\_name(s.object\_id)

, s.object\_id

, indexname=i.name

, data\_compression\_desc

, i.index\_id

, s2.rowcnt

, sa.total\_pages

, is\_unique

, (select count(\*)

from sys.indexes r

where r.object\_id = s.object\_id) 'count'

, user\_seeks

, user\_scans

, user\_lookups

, user\_updates

, user\_seeks + user\_scans + user\_lookups + user\_updates AS [total\_usage]

, CAST(CAST(user\_seeks AS DEC(12,2))/CAST(REPLACE((user\_seeks + user\_scans + user\_lookups + user\_updates), 0, 1) AS DEC(12,2)) \* 100 AS DEC(5,2)) [% Seeks]

, CAST(CAST(user\_scans AS DEC(12,2))/CAST(REPLACE((user\_seeks + user\_scans + user\_lookups + user\_updates), 0, 1) AS DEC(12,2)) \* 100 AS DEC(5,2)) [% Scans]

, CAST(CAST(user\_lookups AS DEC(12,2))/CAST(REPLACE((user\_seeks + user\_scans + user\_lookups + user\_updates), 0, 1) AS DEC(12,2)) \* 100 AS DEC(5,2)) [% Lookups]

, CAST(CAST(user\_updates AS DEC(12,2))/CAST(REPLACE((user\_seeks + user\_scans + user\_lookups + user\_updates), 0, 1) AS DEC(12,2)) \* 100 AS DEC(5,2)) [% Updates]

, last\_user\_scan

, last\_user\_seek

, getdate() run\_time

from sys.dm\_db\_index\_usage\_stats s

join sys.indexes i on i.object\_id = s.object\_id

and i.index\_id = s.index\_id

join sysindexes s2 on i.object\_id = s2.id

and i.index\_id = s2.indid

join sys.partitions sp on i.object\_id = sp.object\_id

and i.index\_id = sp.index\_id

join sys.allocation\_units sa on sa.container\_id = sp.hobt\_id

where objectproperty(s.object\_id, 'IsUserTable') = 1

and database\_id = @dbid

--and 'etblHistory' = object\_name(s.object\_id)

UNION ALL

SELECT @@SERVERNAME AS [Server Name]

, db\_name() AS [DB Name]

, 'NA'

, objectname = object\_name(o.object\_id)

, o.object\_id

, indexname = i.name

, i.index\_id

, s2.rowcnt

, sa.total\_pages

, is\_unique

, data\_compression\_desc

, (select count(\*)

from sys.indexes r

where r.object\_id = i.object\_id) 'count'

, 0

, 0

, 0

, 0

, 0

, 0

, 0

, 0

, 0

, 0

, 0

, getdate() run\_time

FROM sys.indexes i

JOIN sys.objects o

ON i.object\_id = o.object\_id

join sysindexes s2 on i.object\_id = s2.id

and i.index\_id = s2.indid

join sys.partitions sp on i.object\_id = sp.object\_id

and i.index\_id = sp.index\_id

join sys.allocation\_units sa on sa.container\_id = sp.hobt\_id

WHERE OBJECTPROPERTY(o.object\_id,'IsUserTable') = 1

AND i.index\_id NOT IN (

SELECT s.index\_id

FROM sys.dm\_db\_index\_usage\_stats s

WHERE s.object\_id = i.object\_id

AND i.index\_id = s.index\_id

AND database\_id = @dbid)

--AND i.index\_id NOT IN (0,1)

order by last\_user\_scan, last\_user\_seek

1. Covering Indexes
   1. Index with INCLUDE clause is the covering index. Covering index is basically used to cover the query(include columns from Select list which are not part of index) and to avoid bookmark lookup. 1023 columns can added in the INCLUDE clause.
   2. A column used in an aggregate in a SELECT clause. Columns that are only returned by the query, and that are not part of the filtering or sorting of data.
   3. By including frequently queried columns in nonclustered indexes, we can dramatically improve query performance by reducing I/O costs. Since the data pages for a nonclustered index are frequently readily available in memory, covering indexes are usually the ultimate in query resolution.

CREATE NONCLUSTERED INDEX IDX\_Employees\_Covering ON Employees (DepartmentId,PositionId)

INCLUDE (FirstName,LastName,Birthdate,ManagerId,Salary,Address,City,State,HiredDate)

1. Filtered Indexes
   1. Select data from a subset of data using the where clause, therefore smaller indexes
   2. Save space, uses less stats, reduces maintenance cost and statistics are more accurate.
   3. Filtered Statistics can improve cardinality estimation, i.e. when joining lookup table, or while joining fact table and dimension table. For this reason, SQL Server supports the creation of up to 30,000 statistics on non-indexed columns. Better estimation with filtered statistics can lead to faster query execution against star schema based on data warehouses.
   4. By estimating the optimal join strategy, with the amount of memory and executing the query in parallel with huge gain.

CREATE INDEX dType ON E.Iitem (Dtype) WHERE Dtype IS NOT NULL

### Index on Computed Columns/ Views

* 1. You should create an indexed view because it persists data for computed column and automatically calculated each time data is inserted, updated or deleted in the underlying table.
  2. Index view utilizes physical disk storage
  3. Frequently used view data can be viewed without executing it’s query
  4. Query optimizer even uses view indexes in queries that do not directly name the view in the from clause
  5. They work best when the underlying data is frequently updated
  6. Excellent for queries with join & aggregated data involving many rows
  7. best used for OLAP systems
  8. best to drop and recreate when data load takes place
  9. cost of updating index view is more than the normal table indexes

----Creating a view

Create View vEmployee WIth Schemabinding As

Select BusinessEntityID, FirstName, LastName

From Person.Person

Select \* from vEmployee

Exec sp\_spaceused N'dbo.vEmployee'

Create Unique Clustered Index [Ix\_vEmployee] ON

dbo.vEmployee(BusinessEntityID)

### OLAP – On-Line Analytical Processing

### ColumnStore Indexes

* 1. With OLTP system care is required, SSIS , SSRS and Datawarehousing is ideal
  2. Column stored in one page → with compression → storage increases -> IO reduces
  3. Uses compression to optimize memory usage & performance
  4. Maintenance of these indexes are more costly than row store indexes
  5. in SQL Server 2016 → Both types of columnstore are read/write
  6. Clustered columnstore:
     1. Allows one Clustered columnstore indexes which include all table columns
     2. The clustered columnstore index will improve query performance on large data warehouse dimension tables and fact tables by achieving a high level of compression of column data.
     3. clustered column indexes removes row based storage
     4. **If you have a clustered columnstore index, then you can have only a non-clustered primary key row based index.**
     5. You should use columnstore compression to meet the business requirement for faster query performance. It will achieve 10 times better compression than rowstore indexes.
     6. Archive compression: Is designed for maximum data compression for data warehouses.

---- Creating columnstore index

CREATE CLUSTERED COLUMNSTORE INDEX [NCCS-Kids1] ON [dbo].[Kids1] WITH (DROP\_EXISTING = OFF, COMPRESSION\_DELAY = 0)

* 1. Nonclustered Columnstore;
     1. For real time analytics queries that scan large amount of data, especially on large table. Nonclustered Columnstore index are suitable for data warehouse that refresh daily.
     2. Non-clustered column indexes → Ideal when column cover query
     3. Used for very large data set
     4. I m+ rows per table
     5. You filter on non-clustered column indexes
     6. Best practice to have non-clustered indexes to your clustered column store indexes
     7. Non-Clustered columnstore index is generally more appropriate for OLTP table → when no change to base table structures.

---To get stats

SELECT \* FROM sys.dm\_db\_column\_store\_row\_group\_physical\_stats

---Alter to compress and re-organize

ALTER index ccolumnstore ON f.sales REORGANIZE WITH C compress\_all\_row\_group=ON);

---Rebuild

ALTER INDEX [name] ON s.table REBUILD;

---Creating

CREATE NONCLUSTERED COLUMNSTORE INDEX MyColumnStore ON dbo.tblCustomer (Ccode, CustName)

CREATE NONClustered COLUMNSTORE INDEX Ix\_Cons\_Date ON Consumer(CreateDate) INCLUDE(CustomerId, CountryCode, City)

WHERE Status=1;

### Spatial

1. A spatial index is a type of extended index that allows you to index a spatial column. A spatial column is a table column that contains data of a spatial data type, such as geometry or geography.

### Full-text

1. Only one full-text index is allowed per table or indexed view, and each full-text index applies to a single table or indexed view. A full-text index can contain up to 1024 columns.

### XML

1. An XML index is a special type of index that is created on XML binary large objects (BLOBs) in the XML data type columns, to enhance the performance of the queries that are retrieving data from that table. Indexing all tags, values, and paths over the XML instances in that column

### Memory-optimized non-clustered indexes

1. SQL Server 2014 introduced the In-memory OLTP engine which is fully integrated in SQL Server and allows you to create memory optimized tables. These memory optimized tables have a completely different data and index structure
   1. No locking, blocking or latching when you access data.
   2. There are no data pages or index pages or buffer pool for memory optimized tables.
2. There are two types of indexes which can be created on memory optimized tables
   1. HASH index or RANGE index.
   2. A memory-optimized table **must always have at least one index**, although if you create a primary key on the table, this requirement will be satisfied.
   3. You can create up to 8 indexes on a memory optimized table including the one supporting the primary key although no unique index is supported other than the primary key.
3. All 8 indexes can only be non-clustered indexes. These indexes don't duplicate data, but rather they just point to the rows in the chain.
   1. These indexes are not logged and don't get stored on disk.   The indexes are maintained online and created every time during recovery. Garbage collection performs clean up when required.
   2. The limitation of **eight indexes** on memory-optimized tables has been eliminated with SQL Server 2017.
   3. <https://sqlwithmanoj.com/tag/in-memory-tables/> -> Removed limitations in SQL Server 2017.
4. One very important point to note about In-Memory OLTP indexes are that they get created when you create the table and there is no way to drop, add or modify indexes on the memory optimized table.  This is because the index pointers are stored as part of the row structure.
5. You need to do thorough analysis and proper index planning, based on the workload or queries, before you create the memory optimized table.

---Created a primary key hash index on the CustomerID column and ---another hash index on the Age column:

CREATE TABLE [CustomerWithHashIndex](

[CustomerID] INT NOT NULL PRIMARY KEY NONCLUSTERED HASH WITH (BUCKET\_COUNT = 1000000),

[Name] NVARCHAR(250) NOT NULL,

[CustomerSince] DATETIME NULL,

Age SMALLINT NOT NULL INDEX [IAge] NONCLUSTERED HASH WITH (BUCKET\_COUNT = 200)

)

WITH (MEMORY\_OPTIMIZED = ON, DURABILITY = SCHEMA\_AND\_DATA);

--Create a memory optimized table with a hash index and a range index.

CREATE TABLE [CustomerWithRangeIndex](

[CustomerID] INT NOT NULL PRIMARY KEY NONCLUSTERED HASH WITH (BUCKET\_COUNT = 1000000),

[Name] NVARCHAR(250) NOT NULL,

[CustomerSince] DATETIME NULL,

Age SMALLINT NOT NULL INDEX [IAge] NONCLUSTERED)

WITH (MEMORY\_OPTIMIZED = ON, DURABILITY = SCHEMA\_AND\_DATA);

### Fragmentation

* 1. Happens with constant Updates, Inserts and Deletes
  2. 5 to 30% -> re-organise -> re-shuffles pages in order
  3. >30% -> Re-build -> very taxing -> drop & rebuilt fresh
  4. Create maintenance plan to rebuild indexes after a backup usually, 3 ways to do that:
     1. SQL Agent job -> Create schedule -> Copy & paste index re-building script
     2. Management -> Maintenance plan Wizard -> Index -> Create schedule -> Select relevant tasks -> Apply to multiple database -> If Index frag >30 & Count >1000 Then execute
     3. Using OLA’s script -> ola.hallemgrem -> several jobs to backup
  5. Rebuilding indexes
     1. Disk space is reclaimed because pages are compact
     2. The index rows are re-ordered
     3. Fragmentation is removed
  6. Index fragmentation → logical order differ from physical →
     1. sysy.dm\_db\_index\_physical\_stats
     2. avg\_fragmentation\_in\_percent
     3. fragmentation\_count
     4. avg\_fragmentation\_size\_in\_pages

--Findinding average fragmentation level

SELECT OBJECT\_NAME(OBJECT\_ID), index\_id,index\_type\_desc,index\_level,  
avg\_fragmentation\_in\_percent,avg\_page\_space\_used\_in\_percent,page\_count  
FROM sys.dm\_db\_index\_physical\_stats  
(DB\_ID(N'AdventureWorksLT'), NULL, NULL, NULL , 'SAMPLED')  
ORDER BY avg\_fragmentation\_in\_percent DESC

-- Find the average fragmentation percentage of all indexes

-- in the HumanResources.Employee table.

SELECT a.index\_id, name, avg\_fragmentation\_in\_percent

FROM sys.dm\_db\_index\_physical\_stats (DB\_ID(N'AdventureWorks2012'),

OBJECT\_ID(N'HumanResources.Employee'), NULL, NULL, NULL) AS a

JOIN sys.indexes AS b

ON a.object\_id = b.object\_id AND a.index\_id = b.index\_id;

GO

--Rebuild Index for All tables in database:

 Exec sp\_msforeachtable 'ALTER INDEX ALL ON ? REBUILD'

GO

--Reorganize Index for All tablex in database:

Exec sp\_msforeachtable 'ALTER INDEX ALL ON ? Reorganize'

GO

## Update Stats

If your database statistics are not up to date, it will cause the query optimizer to have inaccurate information to build the best query plan.

* 1. The optimizer could opt for a parallel plan when a nonparallel plan was the best option making you think that you have something wrong with the max degree of parallelism because you see lots of SOS\_SCHEDULER\_YIELD or CXPACKET waiting tasks.
  2. If this is the case you should update the statistics right away and then execute the DBCC FREEPROCCACHE command to clear the procedure cache so new plans can be built using the updated statistics.

UPDATE STATISTICS table\_name WITH Scan\_type; Two type of Scan: FULLSCAN, SAMPLE BY.

* 1. FULL SCAN update statistics by scanning all rows in the table or indexed view. FULLSCAN or SAMPLE 100 PERCENT gives the same result.
  2. SAMPLE BY update statistics based on a specified percentage or specified number of rows. For PERCENT, number can be from 0 through 100 and for ROWS, number can be from 0 to the total number of rows.

UPDATE STATISTICS Transaction WITH FULLSCAN;

UPDATE STATISTICS Transaction WITH SAMPLE 100 PERCENT;

EXEC sp\_updatestats

GO

DBCC FREEPROCCACHE() ?

GO

## Page Splits

1. Occurs due to insert in the middle of pages
2. Page consists of 8KB (8192B) -> Page header (96B) + Data Row 1..m (8060B) + Row Offset (36B)
3. Need to manage pages to avoid page splits -> which causes performance issues
4. If row size of 4200 bytes then a single row can be stored on a page -> 3860b is wasted space. If 4000b then two rows can be stored on the same page.
5. This could results in excessive I/O and updating indexes
6. Choose index keys that are either increasing or decreasing in value. This will cause insert either beginning or end of pages to avoid splitting.
7. Use default Fill Factor setting (0 or 100) -> unless testing has been done for a specific value with positive results.
8. Do have fill factor 100/0 on read-only workloads. On systems that are not applying insert or updates, a fill factor of 100 makes perfect sense.
9. SQLAccess: Page Splits/sec ->Perfmon counter
10. PAGEIO Latch indicate → data cannot be obtained from memory cache. SQL server has to go to the disk to get the relevant data.
11. PAGEIOLATCH\_SH → is the actual memory reservation to be filled with requested data from disk. SH= Shared wait.
12. Shows mid split markers -> AllocUnit, AllocUnitName, PageId, SlotId
13. SQLServer.Page\_Split, 1=Data Page, 2=index page, 10=IAM page (memory data)

-> via Extended Events

DBCC traceon(3604) -> to view used pages

DBCC IND ('DB\_Name',table\_name,-1)

DBCC page('DB\_Name',1,page\_id,1)

SELECT \* FROM fn\_dblog (NULL, NULL) WHERE operation=N’Lop\_delete\_split’

## Query Optimizer

1. The SQL Server Query Optimizer is a cost-based. Each possible execution plan has an associated cost in terms of the amount of computing resources used. The Query Optimizer must analyze the possible plans and choose the one with the lowest estimated cost.
2. Each SQL requires the use of resources, disk reads, CPU & Memory usage
3. Each SQL is Compiled -> Parsed -> Validated -> Optimized-> access plan created & cached -> Executed
4. **Re-compile causes performance issues** -> different execution plan for same SQL query. The following will cause a change in execution plan:
   1. Schema, Statistics, set option changed, Temp table, DML, DDL, table & view column or index changes
   2. Too many recompilation degrades performance
   3. Perfmon counters: SQLStatistics -> Batch Requests/ Sec, SQL Compilation/Sec, SQL ReCompilation/Sec
   4. Profiler -> Blank -> Event -> Stored Proc, sp:Recompile, SP:stmtCompleted, TSQL:SQL:STMTRecompile?
   5. Any SELECT \* will cause a recompile
   6. Use extended events instead of profiler on production
   7. Normally, if you see batch request 10% or higher then rewrite SP. For systems that have very long, complex stored procedures, *Batch Requests/sec* may not be a good metric to determine how busy the server is.
5. Specifying columns in select statements will optimize the query
6. The more filters in the Where clause the better. Less data is returned.
7. Select only columns that you need avoid network, bandwidth.
8. General rule, always join columns that have indexes, keys on them and avoid joining columns like character data
9. Revisit indexing often. Use Index Tuning Wizard for a guide.
10. Create indexes on Boolean and numeric data types. As they provide a high value of uniqueness which are great candidates for indexes
11. Move queries to stored procedures when possible because you can get a reliable performance gain from doing so
12. When the SELECT statement in *MyProc2* is optimized in SQL Server, the value of @d2 is not known. Therefore, the Query Optimizer uses a default estimate for the selectivity of OrderDate > @d2, (in this case 30 percent).
13. SQL Server detects the changes that invalidate an execution plan and marks the plan as not valid. The conditions that invalidate a plan include the following:
    1. Changes made to a table or view using (ALTER TABLE and ALTER VIEW).
    2. Changes made to a single procedure using (ALTER PROCEDURE).
    3. Changes or dropping any indexes used by the execution plan.
    4. Updates on statistics such as UPDATE STATISTICS, or generated automatically.
    5. An explicit call to sp\_recompile.
    6. Large numbers of changes to keys (generated by INSERT or DELETE statements from other users that modify a table referenced by the query).
    7. For tables with triggers, if the number of rows in the inserted or deleted tables grows significantly.
    8. Executing a stored procedure using the WITH RECOMPILE option.
14. When the PARAMETERIZATION option is set to FORCED, any literal value that appears in a SELECT, INSERT, UPDATE, or DELETE statement, submitted in any form, is converted to a parameter during query compilation.
    1. From SSMS -> Right click on the database then go to Properties, Options, Parameterization -> set option.
    2. The following query clauses are not parameterized. Note that in these cases, only the clauses are not parameterized. Other clauses within the same query may be eligible for forced parameterization.
       1. The <select\_list> of any SELECT statement. This includes SELECT lists of subqueries and SELECT lists inside INSERT statements.
       2. Subquery SELECT statements that appear inside an IF statement.
       3. The TOP, TABLESAMPLE, HAVING, GROUP BY, ORDER BY, OUTPUT...INTO, or FOR XML clauses of a query.
       4. The pattern and escape\_character arguments of a LIKE clause.

### Optimize for Ad-Hoc WorkLoad

1. Execution Plan → Optimize for ad hoc workloads
2. Reduce memory used to store compiled plan of single use ad hoc queries and dynamic sql statements by just storing the compiled plan stub instead of the full compiled plan
3. Increase CPU usage when ad hoc query is executed the second time by compiling the plan again for sorting/ caching
4. Useful when there are lots of single use ad hoc queries to reduce memory overhead. Can cause excessive CPU usage when significant number of ad hoc queries are executed more than once.
5. When the plan is stable it is recommended to parametrize to avoid such plan cache pollution
6. Query\_hash & query\_plan\_hash are same for adhoc queries which have different parameter values. This will cache a single plan and reuse it.
7. You need to enable optimize adhoc queries so that queries plans are cached, check CPU compile time. Useful for single dynamic sql statetments. Parameterise sql statements in order to save cache pollution.
8. Setting the **optimize for ad hoc workloads** to 1 affects only new plans; plans that are already in the plan cache are unaffected. To affect already cached query plans immediately, the plan cache needs to be cleared using
9. <https://docs.microsoft.com/en-us/sql/database-engine/configure-windows/optimize-for-ad-hoc-workloads-server-configuration-option?view=sql-server-2017> 🡪 More reads
10. Forced parameterization is when the database engine parameterizes any literal value that appears in a SELECT, UPDATE, INSERT, or DELETE statement submitted in any form, but there are a few exceptions.
11. Simple parameterization SQL Server internally will add parameters where needed so that it can try to reuse a cached execution plan.
12. <https://www.mssqltips.com/sqlservertip/2935/sql-server-simple-and-forced-parameterization/>
13. Automatic tuning in SQL Server 2017 (14.x) notifies you whenever a potential performance issue is detected, and lets you apply corrective actions, or lets the Database Engine automatically fix performance problems.
14. <https://docs.microsoft.com/en-us/sql/relational-databases/automatic-tuning/automatic-tuning?view=sql-server-2017>

exec sp\_configure 'optimize for ad hoc workloads'

go

exec sp\_configure 'optimize for ad hoc workloads', 1

reconfigure

go

exec sp\_configure 'optimize for ad hoc workloads'

go

exec sp\_configure 'optimize for ad hoc workloads'

go

exec sp\_configure 'optimize for ad hoc workloads', 0

reconfigure

go

exec sp\_configure 'optimize for ad hoc workloads'

go

--Forced

ALTER DATABASE AdventureWorks2012 SET PARAMETERIZATION FORCED

--Simple

ALTER DATABASE AdventureWorks2012 SET PARAMETERIZATION SIMPLE

---- let SQL Server 2017 to automatically correct any plan that regressed.

ALTER DATABASE current SET AUTOMATIC\_TUNING ( FORCE\_LAST\_GOOD\_PLAN = ON )

---- obtain a script that fixes the issue and additional information about the ---- estimated gain

SELECT reason, score,

script = JSON\_VALUE(details, '$.implementationDetails.script'),

planForceDetails.\*,

estimated\_gain = (regressedPlanExecutionCount + recommendedPlanExecutionCount)

\* (regressedPlanCpuTimeAverage - recommendedPlanCpuTimeAverage)/1000000,

error\_prone = IIF(regressedPlanErrorCount > recommendedPlanErrorCount, 'YES','NO')

FROM sys.dm\_db\_tuning\_recommendations

CROSS APPLY OPENJSON (Details, '$.planForceDetails')

WITH ( [query\_id] int '$.queryId',

regressedPlanId int '$.regressedPlanId',

recommendedPlanId int '$.recommendedPlanId',

regressedPlanErrorCount int,

recommendedPlanErrorCount int,

regressedPlanExecutionCount int,

regressedPlanCpuTimeAverage float,

recommendedPlanExecutionCount int,

recommendedPlanCpuTimeAverage float

) AS planForceDetails;

## Data Compression

1. There is Row or Page compression types
2. Compression Ratio depends on data and indexing strategy
3. Reduces I/Os, so improving performance
4. Have CPU & DML overhead to handle the compression/decompression but is compensated by reduced I/O.
5. The optimizer will provide inaccurate costing due to reduced size of tables due to row or page compression.
6. The release of SQL Server 2016 SP1 contains page compression and this is now available on all editions of SQL Server.
7. It can reduce the size of data and indexes by over 60%.
8. Improves cache performance because more data stored in memory
9. For example implementing Page Compression on a 100m row a table can reduce it down to 35% of its original size. From say 58Gb to 20Gb.
10. For large amount of data in your database some of these index rebuild operations may take a long time.
11. Sparse columns are not supported and will need to be converted.
12. Page level compression does Row level first. Row level turns fixed length data into variable length types, freeing empty spaces. It also ignores zero & null values saving additional spaces.

----List tables and their row sizes

SELECT DISTINCT  s.name,t.name, i.name, i.type, i.index\_id,

    p.partition\_number, p.rows

FROM sys.tables t

LEFT JOIN sys.indexes i

ON t.object\_id = i.object\_id

JOIN sys.schemas s

ON t.schema\_id = s.schema\_id

LEFT JOIN sys.partitions p

ON i.index\_id = p.index\_id

    AND t.object\_id = p.object\_id

WHERE t.type = 'U'

  AND p.data\_compression\_desc = 'NONE'

ORDER BY p.rows desc

----Estimates savings from Compression

EXEC sp\_estimate\_data\_compression\_savings

    @schema\_name = 'Sales',

    @object\_name = 'SalesOrderDetail',

    @index\_id = NULL,

    @partition\_number = NULL,

    @data\_compression = 'ROW'

EXEC sp\_estimate\_data\_compression\_savings

    @schema\_name = 'Sales',

    @object\_name = 'SalesOrderDetail',

    @index\_id = NULL,

    @partition\_number = NULL,

    @data\_compression = 'PAGE'

---- How to compress

ALTER INDEX PK\_SalesOrderDetail\_SalesOrderID\_SalesOrderDetailID

        ON Sales.SalesOrderDetail

        REBUILD PARTITION = ALL

        WITH (DATA\_COMPRESSION = PAGE);

ALTER INDEX IX\_SalesOrderDetail\_ProductID

        ON Sales.SalesOrderDetail

        REBUILD PARTITION = ALL

        WITH (DATA\_COMPRESSION = PAGE);

ALTER TABLE Production.TransactionHistory REBUILD PARTITION = ALL

WITH (DATA\_COMPRESSION = ROW);

GO

ALTER INDEX IX\_TransactionHistory\_ProductID ON Production.TransactionHistory REBUILD PARTITION = ALL WITH (DATA\_COMPRESSION = PAGE);

GO

## Table Variable Vs Temporary Variable Vs CTE

* + 1. @Table Variable
       1. No statistics available & no index available
       2. incorrect estimation by query optimizer
       3. @Table variable does not support parallelism
       4. 1mb of memory allocated as the minimum for a query
       5. The sort should get a warning sign of tempdb spill
       6. By using option (recompile) will help the optimizer to estimate better on memory allocation. It always uses 30% of the total rows in a table.
       7. Used for small data sets
       8. Table variables are created by using the DECLARE statement. They reside in the tempdb database. Table variables are accessible only within the session that created them.
    2. #Temp table
       1. Has statistics
       2. Appropriate memory is allocated for the query processes
       3. Supports parallelism
       4. Concurrent temp table could lead to meta data contention so you may need to use table variables
       5. Used for large data set
       6. Temporary table (#temp) is created in the tempdb database. So table variable is faster than temporary table. Temporary tables are allowed CREATE INDEXes using Primary Key or Unique Constraint.
    3. CTE
       1. CTE is a good tool but it does have limitations.  It can only be used in the current query scope whereas a temporary table or table variable can exist for the entire duration of the session allowing you to perform many different DML operations against them.
       2. T-SQL code more readable as well as makes writing recursive queries much less complex as the CTE can reference itself.
       3. If the data in the CTE is very small, then its ok to use but for large data volume performance degrades.

## In-Memory Tables

1. Although Microsoft has invested a lot of effort in In-Memory OLTP technology, currently the functionality is not satisfactory. We cannot migrate databases directly without losing referential integrity or performing manual work.
2. Creation involves 4 main steps:
   1. Create the one & only file group
   2. Create in-memory table WITH (MEMORY\_OPTIMIZED=ON)
   3. Create Natively compiled SP -> No select \* , must be schema binding
   4. Use Create Natively compiled SP to populate table & modify data
   5. Must contain exactly one Atomic block
   6. Do not implement full T-SQL
3. Improved performance for OLTP environment
4. Fast, inexpensive and large capacity memory now available
5. Ideal for heavy calculations in T-SQL
6. A memory –optimized table has a representation of itself in Active memory & secondary copy on disk, duality is hidden from user
7. must have at least 1 Primary index
8. changes not written to transaction logs, runs faster as no logging overhead
9. No use of TempDB & no IO overheads, No locking
10. Instead of locks, the memory-optimised table adds a new version of an updated row in the table itself. The original row is kept until after the transaction is committed
11. During transaction, other processes can read original version of row
12. SPs can be natively compiled, resulting in duration that are 99% faster than interpreted
13. Moving table from disk to memory, 5 to 20 times faster performance
14. A non-durable memory-optimized tables do not incur logging overhead, transactions writing to them run faster than write operations on durable tables. However, to optimize performance of durable memory-optimized tables, configure delayed durability at the database or transaction level. Just as with disk-based tables, delayed durability for a memory-optimized table reduces the frequency with which SQL Server flushes log records to disk and enables SQL Server to commit transactions before writing log records to disk.
15. Durable With this type, SQL Server guarantees full durability just as if the table were disk-based. If you do not specify the durability option explicitly when you create a memory-optimized table, it is durable by default. To explicitly define a durable table, use the SCHEMA\_AND\_DATA durability option.
16. https://www.red-gate.com/simple-talk/sql/t-sql-programming/converting-database-memory-oltp/

--- Durable table definition

CREATE TABLE Examples.Order\_IM\_Durable (

OrderID INT NOT NULL PRIMARY KEY NONCLUSTERED,

OrderDate DATETIME NOT NULL,

CustomerCode NVARCHAR(5) NOT NULL

)

WITH (MEMORY\_OPTIMIZED = ON, DURABILITY=SCHEMA\_AND\_DATA);

GO

---- non-durable memory-optimized tables

CREATE TABLE Examples.Order\_IM\_Nondurable (

OrderID INT NOT NULL PRIMARY KEY NONCLUSTERED,

OrderDate DATETIME NOT NULL,

CustomerCode NVARCHAR(5) NOT NULL

)

WITH (MEMORY\_OPTIMIZED = ON, DURABILITY=SCHEMA\_ONLY);

GO

--Set at database level only, all transactions commit as delayed durable

ALTER DATABASE ExamBook762Ch3\_IMOLTP

SET DELAYED\_DURABILITY = FORCED;

--Override database delayed durability at commit for durable transaction

BEGIN TRANSACTION;

INSERT INTO Examples.Order\_IM\_Hash

(OrderId, OrderDate, CustomerCode)

VALUES (1, getdate(), 'cust1');

COMMIT TRANSACTION WITH (DELAYED\_DURABILITY = OFF);

GO

--Set at transaction level only

ALTER DATABASE ExamBook762Ch3\_IMOLTP

SET DELAYED\_DURABILITY = ALLOWED;

BEGIN TRANSACTION;

INSERT INTO Examples.Order\_IM\_Hash

(OrderId, OrderDate, CustomerCode)

VALUES (2, getdate(), 'cust2');

COMMIT TRANSACTION WITH (DELAYED\_DURABILITY = ON);

--Set within a natively compiled stored procedure

CREATE PROCEDURE Examples.OrderInsert\_NC\_DD

@OrderID INT,

@CustomerCode NVARCHAR(10)

WITH NATIVE\_COMPILATION, SCHEMABINDING

AS

BEGIN ATOMIC

WITH (DELAYED\_DURABILITY = ON,

TRANSACTION ISOLATION LEVEL = SNAPSHOT, LANGUAGE = N'English')

DECLARE @OrderDate DATETIME = getdate();

INSERT INTO Examples.Order\_IM (OrderId, OrderDate, CustomerCode)

VALUES (@OrderID, @OrderDate, @CustomerCode);

END;

GO

--Disable delayed durability completely for all transactions

-- and natively compiled stored procedures

ALTER DATABASE ExamBook762Ch3\_IMOLTP

SET DELAYED\_DURABILITY = DISABLED;

--Enable statistics collection at the query level

EXEC sys.sp\_xtp\_control\_query\_exec\_stats @new\_collection\_value = 1;

--Check the current status of query-level statistics collection

DECLARE @c BIT;

EXEC sys.sp\_xtp\_control\_query\_exec\_stats @old\_collection\_value=@c output;

SELECT @c AS 'Current collection status';

--Disable statistics collection at the query level

EXEC sys.sp\_xtp\_control\_query\_exec\_stats @new\_collection\_value = 0;

--Enable statistics collection at the query level for a specific

--natively compiled stored procedure

DECLARE @ncspid int;

DECLARE @dbid int;

SET @ncspid = OBJECT\_ID(N'Examples.OrderInsert\_NC');

SET @dbid = DB\_ID(N'ExamBook762Ch3\_IMOLTP')

EXEC [sys].[sp\_xtp\_control\_query\_exec\_stats] @new\_collection\_value = 1,

@database\_id = @dbid, @xtp\_object\_id = @ncspid;

--Check the current status of query-level statistics collection for a specific

--natively compiled stored procedure

DECLARE @c bit;

DECLARE @ncspid int;

DECLARE @dbid int;

SET @ncspid = OBJECT\_ID(N'Examples.OrderInsert\_NC');

SET @dbid = DB\_ID(N'ExamBook762Ch3\_IMOLTP')

EXEC sp\_xtp\_control\_query\_exec\_stats @database\_id = @dbid,

@xtp\_object\_id = @ncspid, @old\_collection\_value=@c output;

SELECT @c AS 'Current collection status';

--Disable statistics collection at the query level for a specific

--natively compiled stored procedure

DECLARE @ncspid int;

DECLARE @dbid int;

EXEC sys.sp\_xtp\_control\_query\_exec\_stats @new\_collection\_value = 0,

@database\_id = @dbid, @xtp\_object\_id = @ncspid;

### Limitations

* 1. Limitations → no tempdb access therefore no cursors, case statements, INTO, Subqueris in update, CTE, VIEW etc
  2. You must estimate & provide adequate available memory capacity
  3. You may need to employ partitioning techniques to provide better memory management
  4. The database compatibility level must be 130 or above

SELECT d,compatibility\_level FROM sys.databases as d WHERE d.name=db\_Name()

ALTER DATABASE CURRENT SET compatibility\_level=130

* 1. Transactions involving both disk & memory based tables are called cross-container transactions
  2. Require transaction isolation level to be SNAPSHOT

ALTER DATABASE CURRENT SET MEMORY\_OPTIMIZED\_ELEVATE\_TO\_SNAPSHOT=ON

* 1. If you create one or more databases with memory-optimized tables, you should enable Instant **File Initialization (IFI)** by granting the SQL Server service startup account the SE\_MANAGE\_VOLUME\_NAME user right. Without IFI, memory-optimized storage files (data and delta files) will be initialized upon creation, which can have negative impact on the performance of your workload
  2. In-Memory OLTP does not support migrations in most cases of tables, referential integrity, check constraint and so on. It is good for new projects where you expect many transactions, but migrating an existing database is not properly supported.
  3. You cannot issue an ALTER TABLE statement to convert a disk-based table into a memory-optimized table. Instead, you must use a more manual set of steps
     1. Suspend application activity.
     2. Take a full backup.
     3. Rename your disk-based table.
     4. Issue a CREATE TABLE statement to create your new memory-optimized table.
     5. INSERT INTO your memory-optimized table with a sub-SELECT from the disk-based table.
     6. DROP your disk-based table.
     7. Take another full backup.
     8. Resume application activity
  4. CLR types, XML type, SQL Variant, user-defined types, and LOB legacy types (TEXT, NTEXT & IMAGE) are not supported in In-memory OLTP tables. The conversion process can convert most of the unsupported types to nvarchar(MAX).

1. A Computed column in an In-Memory optimized table is supported starting with SQL Server 2017. You cannot convert the ‘ComputedText’ property, i.e. formula.

--- Get all columns that have user-defined data types

   WITH X

  AS

  (SELECT \*

      FROM sys.types

      WHERE is\_user\_defined = 1)

  SELECT  s.name ,t.name   ,c.\*

  FROM SYS.columns c

  INNER JOIN sys.tables t

      ON c.object\_id = t.object\_id

  INNER JOIN sys.schemas s

      ON t.schema\_id = s.schema\_id

  WHERE EXISTS (SELECT \*

      FROM x

      WHERE x.user\_Type\_id = c.user\_type\_id);

1. In SQL Server 2016, only eight indexes including the primary key index are allowed for each memory-optimized table or table type. Starting with SQL Server 2017, there is no longer a limit.
2. I**DENTITY** columns seed and increment must be set to 1.

---To get all columns that have an identity with a seed or increment ---- different from 1

SELECT s.name ,t.name ,c.seed\_value ,c.increment\_value

  FROM sys.identity\_columns c

  INNER JOIN sys.tables t

      ON c.object\_id = t.object\_id

  INNER JOIN sys.schemas s

      ON t.schema\_id = s.schema\_id

  WHERE seed\_value <> 1

  OR increment\_value <> 1;

1. Every foreign key will result in warnings and errors. The Microsoft Advisor reports that is not possible to convert such a table by using the wizard
2. CASCADE is not supported

--- To get all relationships that have some actions on DELETE or UPDATE

SELECT

      f.name AS ForeignKey

     ,SCHEMA\_NAME(f.SCHEMA\_ID) SchemaName

     ,OBJECT\_NAME(f.parent\_object\_id) AS TableName

     ,COL\_NAME(fc.parent\_object\_id, fc.parent\_column\_id) AS ColumnName

     ,SCHEMA\_NAME(o.SCHEMA\_ID) ReferenceSchemaName

     ,OBJECT\_NAME(f.referenced\_object\_id) AS ReferenceTableName

     ,COL\_NAME(fc.referenced\_object\_id, fc.referenced\_column\_id)

         AS ReferenceColumnName

     ,f.delete\_referential\_action\_desc

     ,f.update\_referential\_action\_desc

  FROM sys.foreign\_keys AS f

  INNER JOIN sys.foreign\_key\_columns AS fc

      ON f.OBJECT\_ID = fc.constraint\_object\_id

  INNER JOIN sys.objects AS o

      ON o.OBJECT\_ID = fc.referenced\_object\_id

  WHERE f.delete\_referential\_action != 0

  OR f.update\_referential\_action != 0

  ORDER BY SCHEMA\_NAME(f.SCHEMA\_ID);

## Identifying poor performance Issues

### Locking/ Blocking & Deadlocking

* + 1. Multiple users access the same data at the same time creating a circular chain
    2. Avoid locking contention to improve overall performance, you should keep each transaction short and concise, so it can execute quickly → while holding fewest and smallest possible locks.
    3. Usually, due to poor indexing, improper written queries, insufficient resources etc.
    4. SP\_WHO2 -> sproc -> BlkBy -> 58 -> 52 is blocked
    5. Sys.dm\_exec\_request, Sys.dm\_os\_waiting\_tasks
    6. Activity Monitor -> Right Click Server -> Spid -> BlkBy
    7. SSMS Reports -> Right Click db -> Reports -> Standard ->
    8. Perfmon Counter -> SQLServer:Locks: Lock Waits/sec:\_Total
       1. No of time per sec sql server is not able to retain a lock for a resource
       2. Ideally, you don’t want any request to wait for a lock
    9. SQLServer:General Statistics: Process Block
       1. Identifies the number of blocked processes
       2. Ideally, you don’t want to see any blocked processes
    10. SQL Server recognizes deadlock condition → it terminates one of the transaction and rolls it back
    11. Deadlock graph → Profiler → Blank template → Events → Deadlock Graph → Run
    12. Trace Flag 1204 & 1222 → DBCC TRACEON(1204, 1222,-1) → deadlog info sent to log -> DBCC TRACEOFF (1204, 1222, -1)
    13. Deadlock error 1205 -> Victim error message
    14. Extended Events deadlock graph → discover past deadlocks → Analyse deadlocks after it has happened
    15. Use TRY-CATCH in SP scripts to capture deadlocks
    16. Other methods for resolving deadlocks → Use Snapshot or Read\_Committed\_Snapshot → requires good space for tempdb
        1. Use NOLOCK query hint → dirty reads possible
        2. Add covering non-clustered index → beware of index maintenance overhead
        3. Use HOLDLOCK (HOLDLOCK specifies a certain isolation level, to write SERIALIZABLE) or UPDLOCK (UPDLOCK as such does not specify an isolation level, but in practice you will get REPEATABLE READ) query hints
    17. Three lock escalation mode → **Auto** (use this for partition tables), **Table** (default), **Disable** (consumes vast memory).

---To view blocked process execute the following query

USE [master]

GO

SELECT session\_id

,blocking\_session\_id

,wait\_time

,wait\_type

,last\_wait\_type

,wait\_resource

,transaction\_isolation\_level

,lock\_timeout

FROM sys.dm\_exec\_requests

WHERE blocking\_session\_id <> 0

---You can view information about current locks and the processes

--blocking them using the sys.dm\_tran\_locks dynamic management view.

--This column has one of three values: GRANT, WAIT or CONVERT -> upgrade ---lock status

USE [master]

GO

SELECT \* from sys.dm\_tran\_locks

WHERE request\_status = 'CONVERT'

GO

--To view locking in the particular database, execute the following query --joins sys.dm\_tran\_locks with sys.partitions:

USE [master]

GO

SELECT tl.resource\_type

,tl.resource\_associated\_entity\_id

,OBJECT\_NAME(p.object\_id) AS object\_name

,tl.request\_status

,tl.request\_mode

,tl.request\_session\_id

,tl.resource\_description

FROM sys.dm\_tran\_locks tl

LEFT JOIN sys.partitions p

ON p.hobt\_id = tl.resource\_associated\_entity\_id

WHERE tl.resource\_database\_id = DB\_ID()

GO

--Execute the following to view wait stats for all block processes on SQL Server:

USE [master]

GO

SELECT w.session\_id

,w.wait\_duration\_ms

,w.wait\_type

,w.blocking\_session\_id

,w.resource\_description

,s.program\_name

,t.text

,t.dbid

,s.cpu\_time

,s.memory\_usage

FROM sys.dm\_os\_waiting\_tasks w

INNER JOIN sys.dm\_exec\_sessions s

ON w.session\_id = s.session\_id

INNER JOIN sys.dm\_exec\_requests r

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

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

WHERE s.is\_user\_process = 1

GO

EXEC sp\_who 'active';

GO

EXEC sp\_who2 'active';

GO USE master;

GO

KILL spid

----important details about blocking session id,time(from when the blocking session is running),executing query,user account who is executing this blocking session

SELECT

[s\_tst].[session\_id],

[s\_es].[login\_name] AS [Login Name],

DB\_NAME (s\_tdt.database\_id) AS [Database],

[s\_tdt].[database\_transaction\_begin\_time] AS [Begin Time],

[s\_tdt].[database\_transaction\_log\_bytes\_used] AS [Log Bytes],

[s\_tdt].[database\_transaction\_log\_bytes\_reserved] AS [Log Rsvd],

[s\_est].text AS [Last T-SQL Text],

[s\_eqp].[query\_plan] AS [Last Plan]

FROM

sys.dm\_tran\_database\_transactions [s\_tdt]

JOIN

sys.dm\_tran\_session\_transactions [s\_tst]

ON

[s\_tst].[transaction\_id] = [s\_tdt].[transaction\_id]

JOIN

sys.[dm\_exec\_sessions] [s\_es]

ON

[s\_es].[session\_id] = [s\_tst].[session\_id]

JOIN

sys.dm\_exec\_connections [s\_ec]

ON

[s\_ec].[session\_id] = [s\_tst].[session\_id]

LEFT OUTER JOIN

sys.dm\_exec\_requests [s\_er]

ON

[s\_er].[session\_id] = [s\_tst].[session\_id]

CROSS APPLY

sys.dm\_exec\_sql\_text ([s\_ec].[most\_recent\_sql\_handle]) AS [s\_est]

OUTER APPLY

sys.dm\_exec\_query\_plan ([s\_er].[plan\_handle]) AS [s\_eqp]

ORDER BY

[Begin Time] ASC;

GO

### Waite Stats

* + 1. There can be many factors
    2. Locking, poor index, badly designed queries, latches, Network, disk IO waits etc
    3. Queries have 3 states ->
    4. Running
    5. Runnable (locks removed)
    6. Suspended (Locked)
    7. Wait Stats → if row is narrow then many rows can fit into the same page →when doing update → this will lead PAGELATCH\_EX/ PAGELATCH\_SH Waite issues. → Concurrent access to same page issues. Increase row size by adding a dummy column so that 1 row per page is stored.
    8. SQL Server command timeout/ Application timeout → default period is 30 sec
    9. Better to tune queries rather than increase the timeout value
    10. CXPacket, PageIOLatch\_xx, Lck\_M\_X, Async\_Network\_IO
    11. Asyn\_Network\_IO (communication to client app), CXPacket (parallel exec), OLEDB (calling remote DB), PageIOLatch (exclusive access to page request from disk)

---- Get wait info for s specific application

use master

select r.session\_id, r.status, r.command, r.wait\_type, r.wait\_time, r.wait\_resource

from sys.dm\_exec\_requests r

inner join sys.dm\_exec\_sessions s on (r.session\_id = s.session\_id)

where program\_name like 'SQLTest%'

-----

Select \* from sys.dm\_os\_wait\_stats / waiting\_tasks/ dm\_exec\_requests?

Select \* from sys.dm\_os\_wait\_stats order by 3 desc

Select \* from sys.dm\_os\_wait\_stats WHERE wait\_type=’Lck\_M\_S’ order by 3 desc

DBCC\_SQLPERF(N’sys.dm\_os\_wait\_stats’,clear)

---------

dbcc sqlperf('sys.dm\_os\_wait\_stats', clear)

go

-------

select wait\_time\_ms / (case waiting\_tasks\_count when 0 then NULL else waiting\_tasks\_count end) as AvgWaitMS, \*

from sys.dm\_os\_wait\_stats

where wait\_type in ('PAGELATCH\_EX', 'PAGELATCH\_SH')

Go

--------

select object\_name(object\_id) as object\_name, page\_latch\_wait\_count, page\_latch\_wait\_in\_ms

from sys.dm\_db\_index\_operational\_stats(db\_id(), NULL, NULL, NULL)

where page\_latch\_wait\_count > 0

order by page\_latch\_wait\_in\_ms desc

* 1. Fill Factor
     1. Fill factor is the value that determines the percentage of space on each leaf-level page to be filled with data. In an SQL Server, the smallest unit is a page, which is made of Page with size 8K. Every page can store one or more rows based on the size of the row. The default value of the Fill Factor is 100, which is same as value 0.
     2. A correctly chosen fill-factor value can reduce potential page splits by providing enough space for index expansion as data is added to the underlying table.
     3. When a new row is added to a full index page, the Database Engine moves approximately half the rows to a new page to make room for the new row. This reorganization is known as a page split. A page split makes room for new records, but can take time to perform and is a resource intensive operation. Also, it can cause fragmentation that causes increased I/O operations. When frequent page splits occur, the index can be rebuilt by using a new or existing fill-factor value to redistribute the data.

---Here is the script to measure the Fill Factor at the server level:

SELECT \*

FROM sys.configurations

WHERE name ='fill factor (%)'

---And, here is the script to measure the Fill Factor at the table/index level:

USE YourDatabaseName;

SELECT OBJECT\_NAME(OBJECT\_ID) Name, type\_desc, fill\_factor

FROM sys.indexes

--Create index

CREATE INDEX IX\_Employee\_OrganizationLevel\_OrganizationNode ON HumanResources.Employee

(OrganizationLevel, OrganizationNode)

WITH (DROP\_EXISTING = ON, FILLFACTOR = 80);

GO

--Alter index

ALTER INDEX IX\_Employee\_OrganizationLevel\_OrganizationNode ON HumanResources.Employee

REBUILD WITH (FILLFACTOR = 80);

GO

### Statistics

* + 1. Statistics are objects that contain statistical information about the distribution of values in one or more columns of a table or indexed view. Query optimizer uses these statistics to estimate the number of rows or the cardinality returned in the query result. The query optimizer uses this cardinality estimates to choose either the index seek operator instead of the more resource intensive index scan operator and return the most efficient execution plan.
    2. By default, sql server has the **auto create statistics enabled** (don't turn it off). To check if stats is enabled for the database:
    3. **Select database -> Properties -> Options -> Ensure Auto Create Statistics = true & Auto Update Statistics = true**
    4. SQL server automatically creates stats when you create a table with PK or an index table.
    5. When you rebuild an index it automatically creates new statistics.
    6. logical reads -> Number of pages read from data cache
    7. physical read -> Number of pages read from disk
    8. Auto\_update\_statistics\_Async → background thread
    9. Auto\_create\_statistics → create stats on each column during query execution
    10. From sys.stats/ sys.object WHERE type=’U’ → U = User table
    11. SQL Server creates & updates stats automatically for all indexes & columns used in WHERE or JOIN ON clause.
    12. You can disable automatic stats update options & implement maintenance plan to update stats on demand or on schedule.

Set statistic IO ON

Set statistic Time ON

sys.sp\_xtp\_control\_proc\_exec\_stats → enables stats for SQL instance

sys.sp\_xtp\_control\_query\_exec\_stats → enables stats for query level

Primary key = cardianality high = 1= uniqueness= having high selectivity

DBCC show\_statistics(‘table’,index\_name);

ALTER DATABASE db SET AUTO\_CREATE\_STATISTICS OFF;

ALTER DATABASE db SET AUTO\_UPDATE\_STATISTICS OFF;

### Backup & Maintenance plan to avoid performance issues

* 1. Avoid backup jobs during production hours
  2. Determine minuets, hours or days it takes to do a backup
  3. Scatter full, differential backups
  4. Create backup jobs on replica
  5. From SQL Server 2008R2, you can tick a check box to use compression. Which means backups will be smaller, they take less time, and the restores will be quicker.
  6. For small databases choose full backup only
  7. Full+transaction log backup strategy is suitable for larger databases
  8. Full+differential backupstartegy for very large database
  9. **Take advantage of compression-> 25% quicker -> saves time and space**
     1. In Object Explorer, right-click a server and select Properties.
     2. Click the Database settings node.
     3. Under Backup and restore, Compress backup shows the current setting of the backup compression default option. This setting determines the server-level default for compressing backups.
     4. If the Compress backup box is checked, new backups are compressed by default.
     5. Checking backup compression via SQL, A value of 0 means that backup compression is off, and a value of 1 means that backup compression is enabled.
  10. Maintenance Plan
      1. It’s difficult to know how often you should be running index maintenance as it’s dependent on your workload and how quickly your data changes.
      2. If you’re unsure about the frequency, then start by checking for index fragmentation on a daily basis.
      3. If it takes a week for fragmentation to reach 30% or more, schedule in weekly index maintenance. If they fragment sooner, you should try to schedule a couple of index maintenance sessions a week, if not daily – this should include updating statistics.

SELECT value

FROM sys.configurations

WHERE name = 'backup compression default' ;

GO

EXEC sp\_configure 'backup compression default', 1 ;

RECONFIGURE;

GO

## Useful Tools

### Query Store

1. Available from SQL Server 2016 +, GUI based
2. Captures and stores historical data for queries, Execution plans, Statistics
3. 1st point of reference for troubleshooting, provides long running queries, performance related issues, & view changes to workload
4. Small footprint, you can execute on production server
5. You can enable for specific databases by right clicking on the database -> Query Store -> Operation mode -> RW
6. **Regressed queries** -> lists slow queries
7. **Top resource consuming queries** -> Plan it -> Force Plan -> Execution plan ->
   * + 1. Shows various stats ->Such as average logical reads, writes, execution counts etc
       2. Bold plan indicates forced plan
8. **Tracked** -> query ID -> tracked query option
9. Configure to change time interval
10. During backup query store info are retained
11. You can also compare 2 execution plans

--- Forcing a particular execution plan

ALTER DATABASE db\_name SET QUERY STORE CLEAN;

Exec sp\_query\_store\_force\_plan query\_id=1, execution\_plan\_id=13;

---View all execution plans

SELECT txt.query\_text\_id, txt.query\_sql\_text, pl.plan\_id, qry.\* FROM sys.query\_store\_plan AS pl JOIN sys.query\_store\_query as qry ON pl.query\_id=qry.query\_id JOIN sys.query\_store\_query\_text as txt ON qry.query\_text\_id = txt.query\_text\_id;

----To View Plan Cache:

SELECT decp.\*, dest.text, deqp.query\_plan FROM sys.dm\_exec\_cached\_plans decp CROSS APPLY sys.dm\_exec\_sql\_text(decp.plan\_handle) dest CROSS APPLY sys.dm\_exec\_query\_plan(decp.plan\_handle) deqp;

### Extended Events

* + 1. Ability to select Events & Columns for trouble shooting to find performance issues or bottlenecks in SQL Server
    2. SQL Server 2016 has about 564 extended events compared to profiler with 235.
    3. Light weight & uses little resources
    4. Easy graphical user interface
    5. Profiler V Ext Event ( Events= Events, Columns=Actions, Filter=Predicate)
    6. SSMS -> Management -> Ext Event-> Sessions -> Right Click -> New Session Wizard -> Name Session -> Next -> Template -> select relevant events -> Event Filter=SQLServer databasename=db\_name -> Next -> Summary -> Finish -> Start -> Turn Green -> Watch live data -> View Target -> stop Session
    7. Use wizard to select event + Actions -> then click script
    8. You can use ALTER EVENT SESSION to add additional events
    9. View Package0.Event\_file -> Grouping
    10. ADD Target Package0.event\_file(SET FileName=’Capture Waite Stats on Lock’)

----You can manually create Extended Event sessions as follows:

CREATE EVENT session [capture wait stats on lock]

ON SERVER

ADD EVENT sql.wait\_info(

ACTION (SqlServer.database\_name, nt\_username, session\_id, sql\_text, transaction\_id))

----Another example of manual event creation

CREATE EVENT SESSION [DELETE] ON SERVER

ADD EVENT sqlserver.object\_deleted(

ACTION(sqlserver.database\_name, sqlserver.nt\_username, sqlserver.SQL\_Text, sqlserver.username)

WHERE (sqlserver.database\_name]=N’dbname’))

ADD TARGET package0.event\_file(SET filename=N’Delete’)

WITH (Start\_State=off)

----look into events, actions & targets

SELECT \* FROM sys.server\_event\_sessions

### PerfMon

* + 1. CPU:
       1. Processor: % Processor Time -> 80 to 90% -> upgrade
       2. Processor: - Processor Queue Length -> if greater than no of processors then CPU issue
    2. Memory
       1. The Available Bytes -> Low level indicates insufficient or the app is not releasing memory
       2. Memory Page/sec -> pages retrieved from disk due to hard page faults or written to disk to free space in working set due to page faults
       3. High rate for Memory Page/sec -> could indicate excessive paging
       4. SQLServer: Buffer Manager: Buffer Cache hit ratio -> find data pages in its buffer cache. The higher this no the better. Data for queries obtained from memory instead of disk. Keep close to 100 as possible.
       5. SQLServer: Buffer Manager: Page Life Expectancy -> Measures how long pages stay in buffer cache in seconds. Below 300 or 5min means additional memory required.
    3. Compilation
       1. SQLServer: SQL Statistics: Batch Requests/ sec -> no of batches per second -> high means more queries being executed.
       2. SQLServer: SQL Statistics: SQL Compilations/ sec -> Resource intensive -> ideally 1 compile per every IO batch requests
       3. SQLServer: SQL Statistics: Re-Compilations/ sec -> minimize the number of re-compiles.
    4. Blocking
       1. SQLServer: General Statistics: User Connections -> no of user connections
       2. SQLServer: Locks: Lock Waites/ Sec: -Total -> The number of times per second SQL server is not able to retain a lock right away -> ideally 0 or close to 0.
       3. Locks
       4. SQLServer: Access Methods: Page Splits/ sec -> ideally <20% batch requests per second.
       5. SQLServer: General Statistics: Process Block -> no of blocked processes. Ideally, you don’t want to see any blocked processes.
    5. Disk
       1. Disks: Average Disk Sec/ Read -> Less than 8ms is good
       2. Disks: Average Disk Sec/ Write -> Less than 2ms is good
       3. DISK: Read Bytes/ Sec
       4. DISK: Write Bytes/ Sec
       5. For both Current and Avg. Disk Queue Length, 5 or more requests per disk could suggest that the disk subsystem is bottlenecked. if the Avg. Disk Queue Length is greater than 2 per hard disk for a prolonged period of time, it may produce a bottlenecked system.
    6. Network
       1. Network Interface: Network Bytes Received/ sec -> There is no specific threshold value.
       2. Network Interface: Network Bytes Sent/ sec -> There is no specific max old value.
    7. Save Counters for Re-Use
       1. Win+R -> MMC.Exe -> (MS Management Console) -> OK -> File -> Add/ Remove Snap-In -> Perf Monitor -> Add OK
       2. Open performance monitor -> add relevant counters -> Save -> once saved then these counters can be re-used.
       3. Perform Baseline measurement at regular intervals over time

### SQL Profiler

* + 1. Deprecated & no longer supported
    2. Or Trace -> Limit data by columns & use a filter
    3. Only trace the required data and nothing else
    4. Column filter -> AppName, databaseName, NTUserName, SPID, TextData
    5. Use Like Filter
    6. You can import Perfmon Counters with SQL Profiler trace side by side to analyse

### Database Engine Tuning Advisor

* + 1. This can be taxing
    2. Tools -> DataBase Tuning Adviser -> Connect -> WorkLoad=QueryStore -> choose relevant db -> Select table/ or File -> select db for workload -> Tuning options -> Indexes -> advanced options -> Session Name Demo -> Start Analysis -> Estimated Improvements = 52% -> Definition=T-SQL script
    3. Best to create indexes for the largest tables
    4. Action -> Apply recommendations -> Save recommendations to a file

### SQL Server Default Trace

* + 1. SQL Server provides a [Default Trace](https://www.mssqltips.com/sqlservertip/1739/using-the-default-trace-in-sql-server-2005-and-sql-server-2008/) of 34 selected events that can be accessed via tools like [SQL Profiler](https://www.mssqltips.com/sql-server-tip-category/83/profiler-and-trace/) or directly via T-SQL. The SQL Server **Default Trace** is enabled by default.

-- List all traces in the server

SELECT \* FROM sys.traces

-- List all default traces

SELECT \* FROM sys.traces WHERE is\_default = 1

-- You can add more events and check the complete list

DECLARE @id INT

SELECT @id=id FROM sys.traces WHERE is\_default = 1

SELECT DISTINCT eventid, name

FROM fn\_trace\_geteventinfo(@id) EI

JOIN sys.trace\_events TE

ON EI.eventid = TE.trace\_event\_id

-- Get all the events that have been captured and their occurrence

DECLARE @path NVARCHAR(260)

SELECT @path=path FROM sys.traces WHERE is\_default = 1

SELECT TE.name AS EventName, DT.DatabaseName, DT.ApplicationName,

DT.LoginName, COUNT(\*) AS Quantity

FROM dbo.fn\_trace\_gettable (@path, DEFAULT) DT

INNER JOIN sys.trace\_events TE

ON DT.EventClass = TE.trace\_event\_id

GROUP BY TE.name , DT.DatabaseName , DT.ApplicationName, DT.LoginName

ORDER BY TE.name, DT.DatabaseName , DT.ApplicationName, DT.LoginName

-- Data & Log File Auto Grow information

DECLARE @path NVARCHAR(260)

SELECT @path=path FROM sys.traces WHERE is\_default = 1

SELECT DatabaseName, [FileName],

CASE EventClass WHEN 92 THEN 'Data File Auto Grow'

WHEN 93 THEN 'Log File Auto Grow'END AS EventClass,

Duration, StartTime, EndTime, SPID, ApplicationName, LoginName

FROM sys.fn\_trace\_gettable(@path, DEFAULT)

WHERE EventClass IN (92,93)

ORDER BY StartTime DESC

--Database: Data & Log File Shrink

DECLARE @path NVARCHAR(260)

SELECT @path=path FROM sys.traces WHERE is\_default = 1

SELECT TextData, Duration, StartTime, EndTime, SPID, ApplicationName, LoginName

FROM sys.fn\_trace\_gettable(@path, DEFAULT)

WHERE EventClass IN (116) AND TextData like 'DBCC%SHRINK%'

ORDER BY StartTime DESC

--Security Audit: Audit DBCC CHECKDB, DBCC CHECKTABLE, DBCC CHECKCATALOG,

--DBCC CHECKALLOC, DBCC CHECKFILEGROUP Events, and more.

DECLARE @path NVARCHAR(260)

SELECT @path=path FROM sys.traces WHERE is\_default = 1

SELECT TextData, Duration, StartTime, EndTime, SPID, ApplicationName, LoginName

FROM sys.fn\_trace\_gettable(@path, DEFAULT)

WHERE EventClass IN (116) AND TextData like 'DBCC%CHECK%'

ORDER BY StartTime DESC

--Security Audit: Audit Backup Event

DECLARE @path NVARCHAR(260)

SELECT @path=path FROM sys.traces WHERE is\_default = 1

SELECT DatabaseName, TextData, Duration, StartTime, EndTime,

SPID, ApplicationName, LoginName

FROM sys.fn\_trace\_gettable(@path, DEFAULT)

WHERE EventClass IN (115) and EventSubClass=1

ORDER BY StartTime DESC

--Security Audit: Audit Restore Event

DECLARE @path NVARCHAR(260)

SELECT @path=path FROM sys.traces WHERE is\_default = 1

SELECT TextData, Duration, StartTime, EndTime, SPID, ApplicationName, LoginName

FROM sys.fn\_trace\_gettable(@path, DEFAULT)

WHERE EventClass IN (115) and EventSubClass=2

ORDER BY StartTime DESC

--Errors and Warnings: Hash Warning

DECLARE @path NVARCHAR(260)

SELECT @path=path FROM sys.traces WHERE is\_default = 1

SELECT TextData, Duration, StartTime, EndTime, SPID, ApplicationName, LoginName

FROM sys.fn\_trace\_gettable(@path, DEFAULT)

WHERE EventClass IN (55)

ORDER BY StartTime DESC

--Errors and Warnings: Missing Column Statistics

DECLARE @path NVARCHAR(260)

SELECT @path=path FROM sys.traces WHERE is\_default = 1

SELECT DatabaseName, TextData, Duration, StartTime, EndTime, SPID, ApplicationName, LoginName

FROM sys.fn\_trace\_gettable(@path, DEFAULT)

WHERE EventClass IN (79)

ORDER BY StartTime DESC;

--Errors and Warnings: Missing Join Predicate

DECLARE @path NVARCHAR(260)

SELECT @path=path FROM sys.traces WHERE is\_default = 1

SELECT DatabaseName,TextData, Duration, StartTime, EndTime, SPID, ApplicationName, LoginName

FROM sys.fn\_trace\_gettable(@path, DEFAULT)

WHERE EventClass IN (80)

ORDER BY StartTime DESC

--Errors and Warnings: Sort Warnings

DECLARE @path NVARCHAR(260)

SELECT @path=path FROM sys.traces WHERE is\_default = 1

SELECT DatabaseName, TextData, Duration, StartTime, EndTime,

SPID, ApplicationName, LoginName

FROM sys.fn\_trace\_gettable(@path, DEFAULT)

WHERE EventClass IN (69)

ORDER BY StartTime DESC

--Errors and Warnings: we can read more details directly from the [SQL Server Error Log](https://www.mssqltips.com/sqlservertip/1476/reading-the-sql-server-log-files-using-tsql/), but some events can also be queried from the Default Trace.

DECLARE @path NVARCHAR(260)

SELECT @path=path FROM sys.traces WHERE is\_default = 1

SELECT TextData, Duration, StartTime, EndTime, SPID, ApplicationName, LoginName

FROM sys.fn\_trace\_gettable(@path, DEFAULT)

WHERE EventClass IN (22)

ORDER BY StartTime DESC

--Auto Stats, Indicates an automatic updating of index statistics has occurred.

DECLARE @path NVARCHAR(260)

SELECT @path=path FROM sys.traces WHERE is\_default = 1

SELECT TextData, ObjectID, ObjectName, IndexID, Duration, StartTime, EndTime,

SPID, ApplicationName, LoginName

FROM sys.fn\_trace\_gettable(@path, DEFAULT)

WHERE EventClass IN (58)

ORDER BY StartTime DESC

### Activity Monitor

* + 1. Use the Activity Monitor Session in Microsoft SQL Management Studio. Activity Monitor displays information about SQL Server processes and how it affects the current instance of SQL Server. Also use to analyse schema locking issues, plan cache memory pressure, backup I/O problems.
    2. KILL Process\_ID → all the work that the transaction has done will be rolled back. The database must be put back in the state it was in, before the transaction started.
    3. SP\_WHO2, Activity Monitor, SQL Server Profiler, dbcc inputbuffer(spid)
    4. You should use the sys.database\_connection\_stats DMV to count the number of successful and failed connections.
    5. sys.dm\_db\_index\_physical\_stats DMF (Dynamic Management Function) to check the overall status of indexes in a database. See fragmentation section for usage

### DMVs

* + 1. There are many DMV & DMF with each version of SQL Server, more is being added
    2. Provides info on Performance, Indexes, Wait Stats, SQL Counters, Resource Usage & More
    3. Two groups -> Server or Database scope
    4. sys.dm\_db\_index\_usage\_stats DMV is used to review the use of indexes to resolve queries.

--To get a list of all DMO’s execute the following:

SELECT name, type\_desc FROM sys.system\_objects WHERE name like ‘dm\_%’

--lists all connection

SELECT \* FROM sys.dm\_exec\_connection

--lists all sessions

SELECT session\_id, login\_time, host\_name, program\_name, login\_name, nt\_user\_name FROM sys.dm\_exec\_session Order BY 5

--Find average CPU Time:

SELECT TOP 10 total\_worker\_time/ execution\_count AS [avg cpu time], SUBSTRING(st.text,(qs.statement\_start\_offset/2)+1, ((CASE qs.statement\_end\_offset WHEN -1 THEN datalength(st.text) ELSE qs.statement\_end\_offset END qs.statement\_start\_offset)/2)+1 AS Statement\_text FROM sys.dm\_exec\_query\_stats AS qs CROSS APPLY sys.dm\_exec\_sql\_text (qs.sql\_handle) AS st ORDER BY total\_worker\_time/ execution\_count DESC

--- sys.dm\_tran\_locks → view all current locks, resources, mode

Select \* from sys.dm\_tran\_locks WHERE resources\_database\_id=(‘dbname’)

Select object\_name(object\_id) as Resource, object\_id, hobt\_id From sys.partitions

SELECT \* from sys.dm\_tran\_locks as t1 Inner Join sys.dm\_os\_waiting\_tasks as t2 ON t1.lock\_owner\_address = t2.resource\_address;

---- sys.dm\_os\_waiting\_tasks → which tasks are waiting for resources

Select \* From sys.dm\_os\_waite\_stats WHERE waite\_type LIKE ‘LCK%’

DBCC SQLPERF (N’sys.dm\_os\_wait\_stats’,Clear)

* + 1. Dynamic Management Objects → clears info when SQL Server restarts
       1. sys.dm\_db\_index\_usage\_stats → renew index usage
       2. sys.dm\_db\_index\_physical\_stats →status of indexes → see fragmentation section for details
       3. sys.dm\_db\_missing\_index\_details →
       4. sys.dm\_db\_missing\_index\_group →
       5. sys.dm\_db\_missing\_index\_group \_stats →
       6. Find unused indexes → P277
       7. Find indexes that are maintained but never used → p278
       8. sys.indexes, sys.partition, sys.objects, sys.tables, sys.schemas, sys.index\_columns, sys.columns
    2. Performance & wait Stats
       1. SELECT \* FROM sys.dm\_os\_wait\_stats,
       2. SELECT \* FROM sys.dm\_os\_performance\_counters,
       3. SELECT \* FROM sys.dm\_os\_waiting\_tasks
       4. SELECT \* FROM sys.dm\_os\_schedulers
    3. I/O
       1. sys.dm\_io\_virtual\_file\_stats,
       2. sys.dm\_io\_pending\_io\_requests
    4. Transactions
       1. sys.dm\_tran\_lock
       2. sys.dm\_db\_index\_operational\_stats
       3. sys.dm\_db\_index\_usage\_stats
    5. SPID Activity & SQL Statements
       1. SELECT \* FROM sys.dm\_os\_exec\_requests
       2. SELECT \* FROM sys.dm\_exec\_requests
       3. sys.dm\_exec\_query\_stats
       4. sys.dm\_exec{procedure | trigger}\_stats
       5. sys.dm\_exec\_session\_wait\_stats
    6. Capture query plans
       1. sys.dm\_exec\_cached\_plans
       2. sys.dm\_exec\_sql\_text
       3. sys.dm\_exec\_query\_plan
    7. Other DMO
       1. sys.dm\_exec\_\*
       2. sys.dm\_os\_\*
       3. sys.dm\_tran\_\*
       4. sys.dm\_io\_\*
       5. sys.dm\_db\_\*
       6. sys.database\_connection\_stats → login failures
       7. sys.dm\_db\_resource\_stats → expensive queries
       8. sys.dm\_tran\_locks → discover blocked queries
       9. sys.event\_log → 30 days of database

## ISOLATION

* + 1. access to transactions, type of lock acquired during read, duration of lock, blocking transactions to resources with exclusive lock
    2. specify how read operations should behave when other concurrent transactions are changing database
    3. lowering ISO → increases concurrency but also increases risk of dirty reads
    4. raising ISO → minimize concurrency problems but transactions will block one another & performance will suffer
    5. Pessimistic ISO uses blocking, relies on locks
    6. Optimistic ISO uses snapshot, enables higher concurrency
    7. Default isolation is -> READ COMMITTED
    8. SNAPSHOT – OPTIMISTIC -> Allows Read & Write to run concurrently without blocking one another
       1. memory & space required
       2. No Locks used for read operations
       3. Cannot be used for distributed transactions

--- To check database information including Isolation level

DBCC useroptions

--- To change isolation level to SNAPSHOT

ALTER DATABASE db1

SET ALLOW\_SNAPSHOT\_ISOLATION ON;

SET TRANSACTION ISOLATION LEVEL SNAPSHOT

## Stored Procedures: Interpreted Vs CLR Vs Natively Compiled

* + A stored procedure code in SQL can be re-used many times.
  + With relational databases main issues involve around performance such as latching/locking, disk I/O, interpreting code into CPU. Most of these issues can be resolved by taking advantage of CLS & newly introduced natively compiled SPs. They execute direct from memory.
  + In-Memory OLTP is claimed to have the ability to improve performance by 20% or more, depending on workload, data, transaction type and hardware used.
  + There are three types of Stored Procedures that can be created to implement business logics against SQL Server databases. These are:
    1. Interpreted/ Standard Stored Procedure
    2. CLR/ Extended
    3. Natively compiled

### Interpreted/ Standard Stored Procedure

* + 1. Uses Transact-SQL scripts, self-contained logic → Compiled first time when executed -> Execution plan created in Cache -> Subsequent re-execution done from Cache
    2. It accesses both disk-based and memory-optimized table.
    3. Support ENCRYPTION option to encrypt the data.
    4. Support OUTPUT parameter in the procedure definition.
    5. Performance of the database improved by reducing the network traffic.

CREATE PROCEDURE procedure\_name

AS

sql\_statement

GO;

EXEC procedure\_name;

### CLR/ Extended Stored Procedure

* + 1. CLR integration includes stored procedures, triggers, user-defined types, user-defined functions, and user-defined aggregate functions.
    2. CLR Stored Procedures are managed codes to ensure type safety, memory management etc.
    3. Object oriented programming capability -> encapsulation, polymorphism & inheritance.
    4. Can be written in C#, VB or any other .NET Framework supported language.
    5. When functionality is CPU intensive, CLR Stored Procedures give better results as they are pre-compiled into a DLL which executes from memory of the database.
    6. Drawbacks:
       1. They should not be used to execute simple queries. In that case, standard stored procedures give better results.
       2. Deployment may be difficult in some scenarios.
       3. Maintenance of standard SP is much easier
       4. No real change since 2012, it may be deprecated in favor of Python & R languages according to the following website:

<https://sqlquantumleap.com/2018/08/09/sqlclr-vs-sql-server-2017-part-8-is-sqlclr-deprecated-in-favor-of-python-or-r-sp_execute_external_script/>

* + - 1. CLR Integration vs. Extended Stored Procedures.

<https://www.careerride.com/sqlserver-clr-integration-vs-extended-stored-procedures.aspx>

|  |  |
| --- | --- |
| **CLR Integration** | **Extended Stored Procedures** |
| Memory management, threads scheduling, and synchronization services are deeply integrated between managed code of CLR and SQL Server. | They are built to perform functionality which is not possible with T-SQL stored procedures. |
| Do not compromise integrity of SQL Server process as it is managed type safe code. | Compromise integrity of SQL Server process |
| Not supported by older versions of SQL Server | Supported by older versions of SQL Server |
| Can be written using any .NET compliant language. | Can be only written using c/c++ |
| Slower performance | Faster performance |

---Creating a CLR Stored Procedure:

--- From VS2015 -> File -> New project -> SQL Server -> Provide Project name -> OK

---Right Click project -> Add->New Item-> SQL CLR C# -> User Defined Function

---Build the following solution and note the location of DLL

using System;

using System.Data;

using System.Data.SqlClient;

using System.Data.SqlTypes;

using Microsoft.SqlServer.Server;

public partial class StoredProcedures

{

///

/// Prints a Message

///

[Microsoft.SqlServer.Server.SqlProcedure]

public static void myTestStoredProcedure()

{

//Simple proc

SqlPipe objSqlPipe = SqlContext.Pipe;

objSqlPipe.Send("Hi! I am simple CLR PROC");

}

///

/// Proc to Show Rows of [EmployeeDB]..[Roles] table

///

[Microsoft.SqlServer.Server.SqlProcedure]

public static void spGetRolesList()

{

//It returns rows from Roles table

SqlConnection conn = new SqlConnection();

conn.ConnectionString = "Context Connection=true";

SqlCommand cmd = new SqlCommand();

cmd.Connection = conn;

cmd.CommandText = @

"Select \* from [dbo].[Roles] Order By HireDate";

conn.Open();

SqlDataReader sqldr = cmd.ExecuteReader();

SqlContext.Pipe.Send(sqldr);

sqldr.Close();

conn.Close();

}

///

/// It shows rows from Employee table on basis of supplied age

///

/// a specified age

[Microsoft.SqlServer.Server.SqlProcedure]

public static void spGetEmployeeList(Int32 intAge)

{

//It returns rows from Employee table on basis of supplied age

SqlConnection conn = new SqlConnection();

conn.ConnectionString = "Context Connection=true";

SqlCommand cmd = new SqlCommand();

cmd.Connection = conn;

conn.Open();

cmd.CommandText = "Select \* from [dbo].[Employees] Where Age >=@intAge Order By Age";

SqlParameter paramAge = new SqlParameter();

paramAge.Value = intAge;

paramAge.Direction = ParameterDirection.Input;

paramAge.DbType = DbType.Int32;

paramAge.ParameterName = "@intAge";

cmd.Parameters.Add(paramAge);

SqlDataReader sqldr = cmd.ExecuteReader();

SqlContext.Pipe.Send(sqldr);

sqldr.Close();

conn.Close();

}

};

---Within SQL Server -> Select the DB -> perform the following:

sp\_configure 'clr enabled', 1

CREATE ASSEMBLY AssemblyName

FROM ‘C:\temp\ AssemblyName.dll

WITH PERMISSION\_SET = SAFE

GO

CREATE PROCEDURE usp\_AssemblyName AS

EXTERNAL NAME AssemblyName.ClassWithinAssembly.MethodName

EXEC usp\_NameOfClas

### Natively compiled stored procedures

* + 1. Natively compiled stored procedures contain Transact-SQL compiled to native code.
    2. Only used for Memory Optimized tables they cannot access disk based table
    3. When a database or server is brought online, natively compiled SP must be recompiled
    4. Transact-SQL stored procedures compiled to machine code, rather than interpreted by the query execution engine. First translated into C code, and then into machine language. Therefore, execution and access to data is faster and more efficient.
    5. Natively compiled stored procedures are recompiled on first execution of the procedure if the server is restarted.
    6. The machine language version of the code is stored as a dynamic link library (DLL) in the default data directory for your SQL Server instance.
    7. The query execution plan for a natively compiled stored procedure can be retrieved using **Estimated Execution Plan** in Management Studio

SET SHOWPLAN\_XML ON

GO

EXEC dbo.usp\_myproc

GO

SET SHOWPLAN\_XML OFF

GO

* + 1. When a table or stored procedure is compiled to a DLL, this DLL is immediately loaded into memory and linked to the sqlserver.exe process. A DLL cannot be modified while it is linked to a process.
    2. DLL files will not be included in any backups. DLL Files will be deleted automatically if the stored procedure or the in-memory table it uses is deleted.
    3. After a backup, all required DDLs will be created by SQL Server.
    4. Following requirements for Creating a natively compiled stored procedure:
       1. NATIVE\_COMPILATION – natively compiled.
       2. SCHEMABINDING –we cannot alter any objects referenced by the pre-compiled stored procedure without first dropping the stored procedure.
       3. BEGIN ATOMIC –This ensures that the entire stored procedure will execute within its own discrete transaction.
       4. TRANSACTION ISOLATION LEVEL = SNAPSHOT – in-memory OLTP engine only supports SNAPSHOT.
       5. LANGUAGE – must specify the language.

-- Create natively compiled stored procedure

CREATE PROCEDURE Examples.OrderInsert\_NC

@OrderID INT,

@CustomerCode NVARCHAR(10)

WITH NATIVE\_COMPILATION, SCHEMABINDING

AS

BEGIN ATOMIC

WITH (TRANSACTION ISOLATION LEVEL = SNAPSHOT, LANGUAGE = N'English')

DECLARE @OrderDate DATETIME = getdate();

INSERT INTO Examples.Order\_IM (OrderId, OrderDate, CustomerCode)

VALUES (@OrderID, @OrderDate, @CustomerCode);

END;

GO

* + 1. Beginning with SQL Server 2017 (14.x),
       1. CASE statements are now supported
    2. Beginning with SQL Server 2016 (13.x), the following are now supported:
       1. DISTINCT operator, UNION and UNION ALL
       2. LEFT OUTER JOIN, RIGHT OUTER JOIN, CROSS JOIN and INNER JOIN.
       3. Subqueries
       4. WHERE clause with OR/NOT/IN/EXISTS
       5. Aggregate functions AVG, COUNT, COUNT\_BIG, MIN, MAX, and SUM.
       6. In SQL Server 2016 & 2017 it now offers improved memory management and supports foreign keys, check and unique constraints, and parallelism
       7. Supports 2TB of durable tables (2014 -> 256GB)
       8. LOBs with large row size for a memory-optimized table
       9. OUTPUT clause in natively compiled stored procedures
       10. Reduced downtime during upgrade

See this website for details: https://blog.the3i.com/introduction-to-sql-server-in-memory-oltp-and-its-implementation

* + 1. Still not supported:
       1. tempdb -> You cannot create or access temporary tables, table variables, or table-valued functions in tempdb. You can create a non-durable memory-optimized table.
       2. Cursors, CASE, SELECT INTO -> use INSERT INTO <table> SELECT
       3. EXISTS, FROM clause or subqueries in an UPDATE statement
       4. MERGE, OUTER JOIN which includes LEFT JOINs
       5. ALTER PROCEDURE is not allowed
       6. INTO, INTERSECT, EXCEPT, APPLY, PIVOT, UNPIVOT, IN, LIKE, UNION, DISTINCT, PERCENT, WITH TIES, UDFs, WITH RECOMPILE, and views.
       7. MIN and MAX are not supported for types nvarchar, char, varchar, varchar, varbinary, and binary
       8. DISTINCT in aggregates or in the ORDER BY clause.
       9. The following link includes the complete list: <https://msdn.microsoft.com/en-us/library/dn246937.aspx>.

## SQL Standard Reports

* 1. In SQL Server Management Studio Object Explorer, right-click the SQL Server instance if you want the instance level reports, or the database object node for adequate standards reports
  2. In the context menu, select Reports -> Standard reports -> Select the report from the list
  3. If you have already viewed some of the reports, the most recent ones will be listed below Custom Reports
  4. If you select Custom reports, you’ll be prompted to navigate to the folder where custom reports are saved. These are the .rdl and .rdlc files. If you haven’t created or used any custom reports on this SQL Server instance, the menu will be empty.
  5. Right-click the report and print it or save as an Excel or PDF file
  6. If you’re looking for quick and basic details about SQL Server performance, SQL Server Management Studio standard reports can help. If you want to monitor your servers constantly, be able to select the performance metrics to monitor and configure their thresholds for alerting, keep history for later analysis, this is not the tool. A specialized SQL Server performance monitoring tool is highly recommended in these cases.
  7. Note that some of these reports use more hardware resources and need time to be created and displayed. Therefore, it is recommended to run them during off-peak hours.
  8. Besides the instance-level reports, there is also management-level **Tasks report**. This shows the tasks that are currently executing on your SQL Server instance, this information is useful for performance and blocking troubleshooting. Besides the executed statement, the reports also shows processor time, memory used, pending IO. Blocked and blocking tasks are shown in a separate section.

## Execution plans

* 1. Graphical representation using operators on each query -> Actual, Estimated, Live execution plan -> Graphical, Text, XML plan -> uses query optimizer to decide the most efficient plan based on statistics.
  2. Tool Tip & its meaning:
     1. Estimated operator cost -> % total cost of operation
     2. Estimated CPU cost
     3. Estimated I/O cost
     4. Estimated Row size
     5. Actual and Estimated number of rows
     6. Actual and Estimated execution mode -> whether rows are processed one at a time or in batches
     7. query plan optimization → timeout indicates not optimal
     8. Arrow width → indicates the number of rows affected
     9. Operator cost → locate high cost in %
     10. Operators → Sort or Blocking contribute to performance problems
     11. Warnings → tune your query to optimize environment
     12. Clustered Index Seek → with filter parameter, fastest data retrieval, direct from index. Can only be performed on a clustered or nonclustered index.
     13. A Scan reads an entire structure, which could be a heap, a clustered index, or a nonclustered index.
     14. Types of Operators:
         1. Table scan -> Heap table-> avoid
         2. Clustered index scan -> OK
         3. Clustered index seek -> Good ->
         4. Selectively finds rows in index rather than read all row
         5. Non-Clustered index seek -> Good
         6. Non-Clustered index scan -> OK
         7. Compute Scalar -> as a result of computed column
         8. Filter -> Adding a Where Clause -> helps reduce data set
         9. Sort -> avoid -> need relevant index -> expensive both in memory and on disk -> causes performance issues
         10. Table Joins
             1. Nested Loop

1. smaller table with no index to join large table with index
2. For each record from the **outer input** – find matching rows from the **inner input**.
3. The SQL Server Optimizer choose this operator type when the outer input is small and the inner input has an index on the column(s) by which the two data sets are joined.
4. Having indexes and up-to-date statistics is crucial for this join, because you don’t want SQL Server to accidentally think there’s a small number of rows in one of the inputs when in fact there are a lot.
   * + - 1. Merge Join
5. The optimizer chooses for medium workloads
6. Requires sorted inputs
7. Requires an equality operator
8. The Merge Join simultaneously reads a row from each input and compares them using the join key. If there’s a match, they are returned. Otherwise, the row with the smaller value can be discarded.
9. If the inputs are not both sorted on the join key, the SQL Server Optimizer will most likely not choose the Merge join type and instead prefer the Hash join
   * + - 1. Hash Match
10. not properly sorted, and/or not indexed
11. often negative effective
12. The query optimizer uses hash joins to process large, unsorted, non-indexed inputs efficiently.
13. Because both tables are scanned, the cost of a Hash Join is the sum of both inputs.
14. Higher cost in terms of memory consumption and disk IO utilization.
    * + - 1. Key LookUp -> If not covering index then key lookup (Clustered) Operators → more overhead. A key lookup occurs when data is found in a non-clustered index, but additional data is needed to satisfy the query and therefore a lookup occurs.
          2. RID LookUp -> If query accesses a heap table → RID lookup is used instead. RID lookup have a significant cost associated with them.
          3. SQL Server has two physical operators for implementing aggregations:
15. Stream Aggregate
    1. Is used for scalar aggregates, aggregates that return only a single value, e.g. the SUM, COUNT, AVG, etc.
    2. Very fast because it requires an input that has already been ordered by the columns specified in the GROUP statement.
16. Hash Aggregate 
    1. It’s implemented with the **Hash Match** physical operator.
    2. Works on unsorted data for big tables
    3. Cardinality estimates should produce few groups
    4. Builds a hash table in memory

## Data Collection

* 1. The data collector is integrated with SQL Server Agent and Integration Services, and uses both extensively. Is part of Data Warehouse Management? It provides info on schedule:
     1. Data Usage
     2. Query Statistics
     3. Server Activity
     4. Utility Information
     5. Different data points are collected at different intervals,
     6. Data is used to create Data Management Warehouse Reports
     7. Collected data can be copied to a Central Repository for Global reporting
  2. Configure the management data warehouse on a single instance or multiple instances of SQL Server
     1. Ensure that SQL Server Agent is running.
     2. In Object Explorer, expand the Management node.
     3. Right-click Data Collection, expand Tasks, and then click Configure Management Data Warehouse.
     4. Use the [Configure Management Data Warehouse Wizard](https://docs.microsoft.com/en-us/sql/relational-databases/data-collection/configure-the-management-data-warehouse-sql-server-management-studio?view=sql-server-2017#Wizard) to create a management data warehouse, configure logins, enable data collection, and start the System Data Collection Sets.

## Log Files

* 1. Main Areas to find errors and information messages
     1. database engine errors-> SQL Server error log and Windows Application Event log
     2. Logins/ Logouts errors-> SQL Server error log and Windows Security log
     3. Hardware/ OS errors/ Info -> Windows System Event log
  2. To find the location of SQL Server Error Log file used by the instance of SQL Server
     1. Using SQL Server Configuration Manager
        1. Go to Start > All Programs > Microsoft SQL Server (version) > Configuration Tools > SQL Server Configuration Manager -> In the SQL Server Properties window, click on Advanced Tab and go to Startup Parameters. The SQL Server error log file location will be given after ‘-e’ in the next column of the Startup Parameters.
     2. Using Windows Application Event Viewer
        1. Go to Start > All Programs > Administrative Tools > Server Manager -> Expand Diagnostics > Event Viewer > Windows Logs > Application -> Double-click on event to view the location of SQL Server Error Log
     3. SSMS -> Management -> SQL Logs -> right click on the current log file -> View

1. You can schedule the "DBA - Recycle SQL Server Agent Error Logs" SQL Server Agent Job to run once a week. It becomes easier to open and analyze the SQL Server Agent Error Log file when it is smaller in size
2. It is helpful to recycle old error logs.

----Provides location of SQL Server Agent log files You should now see that the SQLAGENT.OUT

USE MASTER

GO

EXEC msdb..sp\_get\_sqlagent\_properties

GO

EXEC sp\_cycle\_errorlog ; sp\_cycle\_agent\_errorlog

GO

USE master

GO

xp\_readerrorlog 0, 1, N'Logging SQL Server messages in file', NULL, NULL, N'asc'

----- Lists log files associated with Employee database

USE EMPLOYEE

SELECT name, size, max\_size, growth, is\_percent\_growth

FROM sys.database\_files

WHERE type\_desc =’LOG’

---- Returns information about transaction logs for each databases

---- Log size in MB, % of log space used, and status

DBCC SQLPERF(LOGSPACE)

---- Shrink Log file and target size

DBCC SHRINKFILE (Employee\_log, 1);

## Good Table Design

* + 1. Normalize to at least 3rd Normal form -> least redundancy & highest potential performance benefits.
    2. A table can have only one PRIMARY KEY. A column defining the PRIMARY KEY must be defined as NOT NULL. A Primary key constraint is automatically indexed. It creates unique clustered index.
    3. De-Normalize -> only to aid performance
    4. Every table to have at least 1 clustered index
    5. Foreign Key and Primary Key to be indexed
    6. Keep transaction short with simple queries
    7. IDENTITY(1,1) → 1ST being seed, 2nd being step

## Achieving high availability of SQL Server in case of disaster

* + 1. Clustering -> is applied to the entire database server
    2. Mirroring -> applied to the entire database server
    3. Log Shipping-> applied to the entire database server
    4. Replication -> applied to specific database objects
       1. Snapshot
       2. Transactional
       3. Peer to Peer -> no filtering
       4. Merge -> Dynamic filtering

## Summary

* 1. General Good Practices
     1. Separate reporting system from the OLTP
     2. Archive historical data
     3. Partition large tables
     4. Re-boot SQL database server once in a while.
     5. Good queries always specify the required columns by name
     6. OLTP → short transactions & simple queries
     7. Explicit → Begin...End → all commits or non-commits -> better performance
     8. SET NOCOUNT ON -> This can reduce network traffic, because your client will not receive the message indicating the number of rows affected by a Transact-SQL statement.
     9. Log file to investigate issues with application system-> Win App event log, SQL Server Error log & Agent Log
     10. The WITH (TABLOCK) allows to run insert statement in parallel. This reduce concurrency because it will immediately take the table lock on the destination table. So this hint will insert data quickly.
     11. Review the hardware to validate it, is it sufficient to meet the needs of the business.
     12. Review the database design for primary keys, foreign keys, correct data types, reasonable usage of NULL, etc.
     13. Validate that the system is not overloaded or taxed from a design perspective.
     14. Review the current indexes to ensure they are effectively utilized
         1. Clustered indexes
         2. Non clustered indexes
         3. Columnstore indexes
         4. Covering Indexes
         5. Filtered Indexes
         6. Unused Indexes
         7. Missing Indexes
     15. Capture and analyze the most expensive queries to improve.
         1. If possible remove LIKE statements, remove functions in the WHERE clause, remove temp table usage, etc.
         2. Once the indexes have been finalized, deploy them and continue to monitor the performance for no unexpected results.
     16. Make use of In-Memory technologies where possible
  2. Avoid
     1. Use of varchar(max)
     2. Select \* from tables
     3. GUID → 16 BYTES →36 CHAR -. increases page splits
     4. Avoid using functions in where clause because of IO overhead. WHERE CAST(gTime AS DATE) = @sdate; → cast stripes timestamp
        1. However every for gTime will be converted & checked. Index on gTime will not be used.
        2. Better to use WHERE gTime >= @sdate AND gTime < DateAdd(Day, 1, @sdate)
        3. When data types don’t match →implicit conversion performed → however index not used
     5. Shrinking the log file will reduce the physical log file size. Shrinking the log file is not best practice until you don’t have any other option.
        1. After the DBCC SHRINKFILE, use ALTER INDEX REORGANIZE to reduce the index fragmentation without growing the data file(s) again. Simply using REORGANIZE rather than REBUILD avoids the vicious circle of shrinking and growing.
        2. If you absolutely have no choice but to run data file shrink operation, be aware that you’re going to cause index fragmentation. The only way to remove index fragmentation without causing data file growth again is to use *DBCC INDEXDEFRAG* or *ALTER INDEX … REORGANIZE*. These commands only require a single 8KB page of extra space, instead of needing to build a whole new index in the case of an index rebuild operation (which will likely cause the file to grow).

## SSIS

### Balance Data Distributor

* 1. Balance Data Distributor transformer → divides process in half → 50/50 processing

### Parallel Processing

* 1. Parallel processing → Check cpu in Device manager → no of processors (4)+ 2
     1. Check properties of package → by right click in package space → MaxConcurrentExecutable = -1 → means, it will use total no of processors + 2

### Conditional Split

* 1. Conditional split is used to route data rows to different outputs based on conditions. This is similar to CASE statement in programming languages.
  2. Output 1
  3. SUBSTRING(FirstName,1,1) == "A"
  4. Output 2
  5. SUBSTRING(FirstName,1,1) == "B"

### Cache Transform

* 1. Cache transform in SSIS. The "Cache Transform" transformation creates a reference dataset for the Lookup Transformation that will be used in cache, without writing onto disk. It writes data from a data source in the data flow to a Cache connection manager
  2. Cache Transformation in [SSIS](https://www.tutorialgateway.org/ssis/) is used to read data from wide variety of sources such as flat files, Excel sheets and ADO.NET data sources and save data from those data sources in .caw file.

### Merge

* 1. Merge Transformation in [SSIS](https://www.tutorialgateway.org/ssis/) is used to merge two inputs (such as tables or files) and produce one output. Merge Transformation is very useful when we want to merge the error path data (after handling the errors) and normal data. This SSIS Merge transformation uses key column values to insert the data into destination columns.

### Merge Join

* 1. The difference is that with Merge join transformation you can support two inputs from two different data source, for example one from flat file and another from oracle DB, but with join in t-sql you can only join from one data source. The Merge transformation is similar to the Union All transformations
  2. The Merge Join transformation lets us join data from more than one data source, such as relational databases or text files, into a single data flow that can then be inserted into a destination such as a SQL Server database table, Excel spreadsheet, text file, or other destination type. The Merge Join transformation is similar to performing a join in a Transact-SQL statement. However, by using SSIS, you can pull data from different source types. In addition, much of the work is performed in-memory, which can benefit performance under certain condition.

### MultiCast

1. In nutshell, a Multicast transformation is used to create/distribute exact copies of the source dataset to one or more destination datasets.

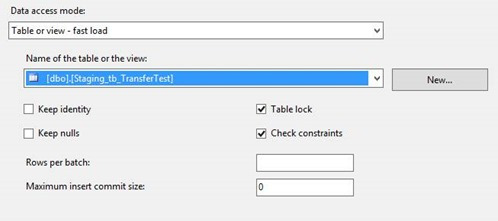
### OLEDB command

1. That being said, OLE DB, for most cases will have better performance than ADO.NET. This is due to ADO.NET being a managed façade, and providing more abstraction with a little more performance overhead.
2. OLE DB has better performance, and because the nifty 3rd upsert task is built on ADO.NET, I won't be using it nor recommending it for use in anything other than a small to mid-size company with small MB incremental loads.
3. The difference between the ADO NET source and ADO NET Destinations when compared to the OLE DB Source and OLE DB Destinations is significant. It was just over 8 x faster.
4. It was interesting to note, that the network card throughput was almost 10 x faster when changing from the ADO NET to OLE DB.
5. Now what sometimes makes the source query faster is using the TSQL Hint: Option (Fast 10000)
6. NOTE: This sometimes makes the query faster, and other times it can also slow the query down. So test first.

Select RowID,DateAdded

Fromdbo.tb\_TableNamewith (nolock)

Option (Fast 10000)

1. The final part is to configure the OLE DB Destination in the data flow task.
2. Double click or right click on the OLE DB Destination and go into the Properties.
3. Then ensure that you have configured with the following as shown below:
4. i.      [](http://gqbi.files.wordpress.com/2013/06/clip_image01011.jpg)
5. NOTE: that the Maximum insert commit size is set to zero.
6. You must ensure that you have enough memory in order to only commit the entire transaction once.
7. If you do NOT have enough memory, when SSIS tries to commit the transaction, it will then fail.
8. If this happens to you, then configure the Maximum insert commit size, to accommodate your memory allocation.

### Archiving Old data

* 1. Archiving → old data, not required for reporting reduces the volume of data which will increase performance.

### Avoid Implicit Data Conversion

* 1. By default source data type is string, this should be corrected to relevant data type instead to avoid implicit data conversion
  2. Avoid data conversion components in order to improve performance

### SCD

* 1. Stands for slowly changing dimension. Used to track master data table values as they rarely change.
  2. Fixed -> no change
  3. Changing -> column value that can change
  4. Historical -> retain old record as well as the new changed records
  5. Type 1 -> tracks only column changes -> No audit
  6. Type 2 -> Keeps history of changes -> There is audit

### Avoid using sort component

* 1. Avoid using sort component as they are overhead. Perform sort at source where possible

### Create cubes for reporting to improve performance

* 1. SSAS improve performance in Analysis calculation by performing calculation and stores them in a cube. This then can be access by various reporting tools.
  2. Cube contains pre-calculation. Centre of the cube contains the numbers/ measures, the sides will provide the dimensions.

### Choosing transaction

* 1. Supported transaction is enabled by default in SSIS.

### CDC is much faster than SCD

* 1. CDC performs incremental changes
  2. SCD provides history tracking
  3. CDC initially does a full load, there after it performs incremental load
  4. Change Data Capture CDC → which records to be extracted. Tracks table changes
     1. Add create/ last update fields
     2. Add flag fields to extracted record
     3. Enable CDC 1st for DB then for individual tables
     4. exec sys.sp\_cdc\_enable\_db
     5. Available for Enterprise or Developer 2016 only
     6. A change table is created for every enabled table
     7. cdc.schemaname\_tablename\_ct
     8. validity interval → sys.sp\_cdc\_enable\_db → removed old changes
  5. used to update master data -> avoid for frequent and large data changes

### Buffer size tuning

* 1. Ensure logging mode is enabled
  2. Optimized=true
  3. On property of Data Flow Task -> set max row & buffer size
  4. You should have a SSIS buffer calculation policy

### Three types of components to aid performance?

* 1. Blocking -> When primeoutput filling event appears in the log file then it will mean it is blocking or a semi blocking component as used some time for it to appear in the log file.
  2. Semi Blocking -> Merge, Merge join & Union all semi-blocking
  3. Non-Blocking

### Synchronous, Asynchronous, Full Blocking, Semi Blocking, non Blocking

* 1. Synchronous components use existing buffers
  2. Asynchronous create new buffers -> Asynchronous components create their own begin end sections

### Implementing multithreading

* 1. From Control flow need to set
  2. MaxConcurrentExecuteable =-1 -> no of processes+2
  3. RunInOptimizedMode=true
  4. From DataFlow = EngineThread number
  5. If 10 DFT each having 5 source & destination paths-> make them run in parallel

### Partitioning

* 1. Helps to separate data into several groups. Then each partitioned data can be retrieved and processed in parallel to increase performance.
  2. How to do SQL Server data partitioning:

CREATE PARTITION FUNCTION [MyFivePartition] (int)

AS RANGE LEFT

FOR VALUES (86747, 173494, 260241, 346988, 433735)

CREATE PARTITION SCHEME [psSample]

AS PARTITION [MyFivePartition]

TO ([PRIMARY],[PRIMARY],[PRIMARY],[PRIMARY],[PRIMARY])

ALTER TABLE [dbo].[TestTable]

ADD CONSTRAINT id

PRIMARY KEY CLUSTERED(id)

ON psSample(id)

* 1. Partitioning the source data and reading & processing each partition data under separate data flow tasks
  2. Conditional splitter
  3. SSIS threading enabled
  4. Range select source and then parallel processing
  5. How to select a particular partition from SQL Server?

SELECT count(\*) from [dbo].[TestTable]

WHERE $PARTITION.[MyFivePartition](id) = 1;

* 1. Partition on year date column (15000 partitions are possible). Pointers refer to the fact table (Not done for query performance, OK for ETL, partition can allow different storage & each partition can be on a separate file group, SAND is optimum storage device)

### Page split performance issues with SSIS

* 1. Page split happens only with clustered index. Sometimes it is better to separate clustered index instead and the primary key. Page split causes performance issues. Usually, developers drop indexes before performing a bulk insert and then re-attach indexes after within the control flow using SQL Execute Task. For this process it is better to use a separate column as clustered index rather than the primary key.
  2. Can page split happen for non-clustered indexes? No
  3. Page split per second help us to know page split issues
  4. We need to drop indexes and recreate them after a major bulk data import

### Cube Sub Aggregation

* 1. The cube contains one main aggregation which could be very slow as it contains all aggregations. Therefore, when creating various smaller sub aggregations, when queries use these they execute much faster.

### What are the different caching modes in Lookup

* 1. Full, loads all data first time and then always goes to the cache for subsequent lookups
  2. Partial updates cache for every lookup it performs
  3. no cache -> No cache always goes to SQL database

### Create Statistics Stat1

* 1. Create Statistics Stat1 onSales.Order (OrderDate) WITH SAMPLE 5 PERCENT
  2. Update Stats → Query optimizer relies on this to estimate how many rows to return. Out of date stats will make its decision making process inaccurate. SQL Server by default will only update stats when 500 rows or 20% of the records have changed.
  3. Filtered index → Helps Sql server to estimate more accurately
  4. Statistic on non index column, take less space of 8kb,

update statistics tab73 with fullscan

create statistics stats\_c1\_where\_c2\_closed on tab73 (c1) where c2 = 'closed'

* 1. specifically useful when joining fact tables with dimension tables

### Improving Bulk Insert performance

* 1. Break files into smaller pieces & run multiple BCP commands in parallel
  2. Source files to be sorted in the same order as the clustered index on the target table and use the ORDER option
  3. Drop any non-clustered indexes from the table before load

### Cube Architecture

* 1. OLAP Architecture
     1. Relational OLAP → Cube structures
     2. Preprocessed aggregates data (Relational Storage)
     3. Slower than Multidimensional storage
     4. Ideal for large volumes of data
     5. Poor query performance as aggregation performed on the fly
  2. Multidimensional OLAP → Cube structure & Preprocessed aggregation
     1. Performance better
  3. Hybrid OLAP → Data sources & Data Source Views,
     1. Proactive cache → Multidimensional & Preprocessed aggregation storage
     2. Combines advantages of ROLAP & MOLAP

## Index A: Useful Link

* 1. <https://www.microsoftpressstore.com/articles/article.aspx?p=2756486&seqNum=4>
  2. <https://msdn.microsoft.com/library/dn673538.aspx>.
  3. [*https://msdn.microsoft.com/en-us/library/bb510411.aspx#InMemory*](https://msdn.microsoft.com/en-us/library/bb510411.aspx#InMemory).
  4. <https://sqlquantumleap.com/2018/08/09/sqlclr-vs-sql-server-2017-part-8-is-sqlclr-deprecated-in-favor-of-python-or-r-sp_execute_external_script/>
  5. <https://blog.the3i.com/introduction-to-sql-server-in-memory-oltp-and-its-implementation>
  6. <https://www.sqlshack.com/sql-server-buffer-pool-action/> -> Buffer pool Extension
  7. <https://docs.microsoft.com/en-us/sql/database-engine/configure-windows/optimize-for-ad-hoc-workloads-server-configuration-option?view=sql-server-2017> -> Optimize for Ad-hoc Workload
  8. <https://docs.microsoft.com/en-us/sql/relational-databases/automatic-tuning/automatic-tuning?view=sql-server-2017> -> Automatic Plan Correction
  9. <https://www.red-gate.com/simple-talk/sql/t-sql-programming/converting-database-memory-oltp/> -> Converting to In-Memory Tables
  10. <https://sqlwithmanoj.com/tag/in-memory-tables/> -> Removed limitations in SQL Server 2017.

## DBCCs (Database Console Commands)

* 1. DBCC ShowContig (TableName); -> shows table stats info
  2. DBCC Show\_statistics (table\_name, Statistics Name)
  3. DBCC FREEPROCCACHE; -> do not do this in production
  4. DBCC DROP CLEANBUFFERS → Clears cache
  5. Do not shrink data file just shrink log files
  6. dbcc @@opentran()
  7. DBCC Shrinkdatabase(DBName, size) → cannot shrink database lower than its original size
  8. DBCC Shrinkfile(DBName, size) → shrink files lower than their original size
  9. DBCC OpenTran → shows oldest active transaction
  10. dbcc inputbuffer(spid) -> — Will give you the Event Info
  11. *DBCC INDEXDEFRAG*
  12. DBCC SHRINKFILE (DBName, TrucateOnly) → After archiving old data run this
  13. DBCC CHECKDB -> to check for corruption and to do repair
  14. DBCC CHECKDB(DB1) WITH EXTENDED\_LOGICAL\_CHECKS
  15. DBCC LOGINFO
  16. DBCC LOGSTATUS
  17. DBCC SQLPERF(“sys.dm\_os\_wait\_stats”, CLEAR) -> clearing existing wait stats
  18. DBCC LOGSTATUS
  19. DBCC SQLPERF(LOGSPACE) -> Provides transaction log info for each database
  20. DBCC TRACEON/OFF (1204, 1222, -1) -> turning deadlock trace on or off for all sessions
  21. DBCC TRACEON (3604) -> to view used pages
  22. DBCC IND ('DB\_Name',table\_name,-1)
  23. DBCC page('DB\_Name',1,page\_id,1) -> identifying objects involved in page locks
  24. DBCC useroptions -> To Check database information including Isolation level
  25. DBCC lock(StallReportThreshold,200) – To identify statements that wait for certain lock times

## Index B: Useful System Stored Procedures

* 1. Sp\_who/2,
  2. Sp\_who2 ‘Active’
  3. sp\_monitor
  4. sp\_spaceused
  5. exec sp\_helpindex 'tablename'
  6. Exec Sp\_helpfile -> current size & location
  7. sp\_configure setting (‘Cost Threshold For Parallelism’ and ‘Max degree of Paralleism’)
  8. sp\_flush\_log → manual request
  9. Exec sp\_readerrorlog
  10. EXEC sp\_lock spid
  11. sp\_configure ‘max server memory (MB)’,<Memory in MB>

## Index C: Useful SQLs

* 1. ----List the size of each tables

USE AdventureWorks2017 -- replace your dbname

GO

SELECT

s.Name AS SchemaName,

t.Name AS TableName,

p.rows AS RowCounts,

CAST(ROUND((SUM(a.used\_pages) / 128.00), 2) AS NUMERIC(36, 2)) AS Used\_MB,

CAST(ROUND((SUM(a.total\_pages) - SUM(a.used\_pages)) / 128.00, 2) AS NUMERIC(36, 2)) AS Unused\_MB,

CAST(ROUND((SUM(a.total\_pages) / 128.00), 2) AS NUMERIC(36, 2)) AS Total\_MB

FROM sys.tables t

INNER JOIN sys.indexes i ON t.OBJECT\_ID = i.object\_id

INNER JOIN sys.partitions p ON i.object\_id = p.OBJECT\_ID AND i.index\_id = p.index\_id

INNER JOIN sys.allocation\_units a ON p.partition\_id = a.container\_id

INNER JOIN sys.schemas s ON t.schema\_id = s.schema\_id

GROUP BY t.Name, s.Name, p.Rows

ORDER BY s.Name, t.Name

GO

* 1. --Find all blocking processes

SELECT

[s\_tst].[session\_id],

[s\_es].[login\_name] AS [Login Name],

DB\_NAME (s\_tdt.database\_id) AS [Database],

[s\_tdt].[database\_transaction\_begin\_time] AS [Begin Time],

[s\_tdt].[database\_transaction\_log\_bytes\_used] AS [Log Bytes],

[s\_tdt].[database\_transaction\_log\_bytes\_reserved] AS [Log Rsvd],

[s\_est].text AS [Last T-SQL Text],

[s\_eqp].[query\_plan] AS [Last Plan]

FROM

sys.dm\_tran\_database\_transactions [s\_tdt]

JOIN

sys.dm\_tran\_session\_transactions [s\_tst]

ON

[s\_tst].[transaction\_id] = [s\_tdt].[transaction\_id]

JOIN

sys.[dm\_exec\_sessions] [s\_es]

ON

[s\_es].[session\_id] = [s\_tst].[session\_id]

JOIN

sys.dm\_exec\_connections [s\_ec]

ON

[s\_ec].[session\_id] = [s\_tst].[session\_id]

LEFT OUTER JOIN

sys.dm\_exec\_requests [s\_er]

ON

[s\_er].[session\_id] = [s\_tst].[session\_id]

CROSS APPLY

sys.dm\_exec\_sql\_text ([s\_ec].[most\_recent\_sql\_handle]) AS [s\_est]

OUTER APPLY

sys.dm\_exec\_query\_plan ([s\_er].[plan\_handle]) AS [s\_eqp]

ORDER BY

[Begin Time] ASC;

GO

* 1. To find missing indexes use the following DMVs:

SELECT

(user\_seeks + user\_scans) \* avg\_total\_user\_cost \* (avg\_user\_impact \* 0.01) AS IndexImprovement,

id.statement,

id.equality\_columns,

id.inequality\_columns

From sys.dm\_db\_missing\_index\_group\_stats AS iqs

INNER JOIN sys.dm\_db\_missing\_index\_groups AS ig

ON iqs.group\_handle = ig.index\_group\_handle

INNER JOIN sys.dm\_db\_missing\_index\_details AS id

ON ig.index\_handle = id.index\_handle

ORDER BY IndexImprovement DESC;

* 1. Locking, locks lock waits Extended Events

--- To identify statements that wait for certain amount of time for ---locks

DBCC lock(StallReportThreshold,200)

use tempdb

go

if exists (select \* from sys.objects where name = 'sqlws\_xev\_locks\_lock\_waits')

drop view sqlws\_xev\_locks\_lock\_waits

go

create view sqlws\_xev\_locks\_lock\_waits as

with xevents (event\_data)

as

(

select event.query('.') as event\_data from

((select cast (xest.target\_data as xml) as target\_data

from sys.dm\_xe\_sessions as xes

inner join sys.dm\_xe\_session\_targets as xest on (xes.address = xest.event\_session\_address)

where xes.name = 'sqlws\_xevents\_locks\_lock\_waits' and xest.target\_name = 'ring\_buffer') as td

cross apply target\_data.nodes ('//event[@name="locks\_lock\_waits"]') as x (event))

)

select event\_data.value ('(event/@name)[1]', 'varchar(max)') as event\_name,

event\_data.value ('(event/@timestamp)[1]', 'datetime') as event\_timestamp,

event\_data.value ('(event/data[@name="count"]/value)[1]', 'bigint') as [count],

event\_data.value ('(event/data[@name="increment"]/value)[1]', 'bigint') as [increment],

event\_data.value ('(event/data[@name="lock\_type"]/value)[1]', 'bigint') as [lock\_type],

event\_data.value ('(event/action[@name="client\_app\_name"]/value)[1]', 'nvarchar(max)') as action\_client\_app\_name,

event\_data.value ('(event/action[@name="client\_hostname"]/value)[1]', 'nvarchar(max)') as action\_client\_hostname,

event\_data.value ('(event/action[@name="database\_name"]/value)[1]', 'nvarchar(max)') as action\_database\_name,

event\_data.value ('(event/action[@name="nt\_username"]/value)[1]', 'nvarchar(max)') as action\_nt\_username,

event\_data.value ('(event/action[@name="session\_id"]/value)[1]', 'int') as action\_session\_id,

event\_data.value ('(event/action[@name="sql\_text"]/value)[1]', 'nvarchar(max)') as action\_sql\_text

from xevents

go

* 1. Query Post Execution Showplan Extended Events→

----To identify expensive execution plan based on CPU usage and duration

----Collect fewer events in order not to impact performance

----Not possible with profiler

use tempdb

go

if exists (select \* from sys.objects where name = 'sqlws\_xev\_query\_post\_execution\_showplan')

drop view sqlws\_xev\_query\_post\_execution\_showplan

go

create view sqlws\_xev\_query\_post\_execution\_showplan as

with xevents (event\_data)

as

(

select event.query('.') as event\_data from

((select cast (xest.target\_data as xml) as target\_data

from sys.dm\_xe\_sessions as xes

inner join sys.dm\_xe\_session\_targets as xest on (xes.address = xest.event\_session\_address)

where xes.name = 'sqlws\_xevents\_query\_post\_execution\_showplan' and xest.target\_name = 'ring\_buffer') as td

cross apply target\_data.nodes ('//event[@name="query\_post\_execution\_showplan"]') as x (event))

)

select event\_data.value ('(event/@name)[1]', 'varchar(max)') as event\_name,

event\_data.value ('(event/@timestamp)[1]', 'datetime') as event\_timestamp,

event\_data.value ('(event/data[@name="source\_database\_id"]/value)[1]', 'bigint') as [source\_database\_id],

event\_data.value ('(event/data[@name="object\_type"]/value)[1]', 'int') as [object\_type],

(select map\_value from sys.dm\_xe\_map\_values xemv where xemv.object\_package\_guid = '03FDA7D0-91BA-45F8-9875-8B6DD0B8E9F2' and xemv.name = 'object\_type' and xemv.map\_key = event\_data.value ('(event/data[@name="object\_type"]/value)[1]', 'int')) as [object\_type\_map\_value],

event\_data.value ('(event/data[@name="object\_id"]/value)[1]', 'int') as [object\_id],

event\_data.value ('(event/data[@name="nest\_level"]/value)[1]', 'int') as [nest\_level],

event\_data.value ('(event/data[@name="cpu\_time"]/value)[1]', 'bigint') as [cpu\_time],

event\_data.value ('(event/data[@name="duration"]/value)[1]', 'bigint') as [duration],

event\_data.value ('(event/data[@name="estimated\_rows"]/value)[1]', 'int') as [estimated\_rows],

event\_data.value ('(event/data[@name="estimated\_cost"]/value)[1]', 'int') as [estimated\_cost],

event\_data.value ('(event/data[@name="serial\_ideal\_memory\_kb"]/value)[1]', 'bigint') as [serial\_ideal\_memory\_kb],

event\_data.value ('(event/data[@name="requested\_memory\_kb"]/value)[1]', 'bigint') as [requested\_memory\_kb],

event\_data.value ('(event/data[@name="used\_memory\_kb"]/value)[1]', 'bigint') as [used\_memory\_kb],

event\_data.value ('(event/data[@name="ideal\_memory\_kb"]/value)[1]', 'bigint') as [ideal\_memory\_kb],

event\_data.value ('(event/data[@name="granted\_memory\_kb"]/value)[1]', 'bigint') as [granted\_memory\_kb],

event\_data.value ('(event/data[@name="dop"]/value)[1]', 'bigint') as [dop],

event\_data.value ('(event/data[@name="object\_name"]/value)[1]', 'nvarchar(max)') as [object\_name],

event\_data.query ('(event/data[@name="showplan\_xml"]/value)[1]/node()') as [showplan\_xml],

event\_data.value ('(event/data[@name="database\_name"]/value)[1]', 'nvarchar(max)') as [database\_name],

event\_data.value ('(event/action[@name="client\_app\_name"]/value)[1]', 'nvarchar(max)') as action\_client\_app\_name,

event\_data.value ('(event/action[@name="client\_hostname"]/value)[1]', 'nvarchar(max)') as action\_client\_hostname,

event\_data.value ('(event/action[@name="database\_name"]/value)[1]', 'nvarchar(max)') as action\_database\_name,

event\_data.value ('(event/action[@name="nt\_username"]/value)[1]', 'nvarchar(max)') as action\_nt\_username,

event\_data.value ('(event/action[@name="session\_id"]/value)[1]', 'int') as action\_session\_id,

event\_data.value ('(event/action[@name="sql\_text"]/value)[1]', 'nvarchar(max)') as action\_sql\_text

from xevents

go

select text, query\_plan, execution\_count, query\_hash, query\_plan\_hash, sql\_handle, plan\_handle

from sys.dm\_exec\_query\_stats

outer apply sys.dm\_exec\_sql\_text(sql\_handle)

outer apply sys.dm\_exec\_query\_plan(plan\_handle)

where text like '%use sql' + 'workshops%'

go

select text, query\_plan, usecounts, size\_in\_bytes, cacheobjtype, objtype

from sys.dm\_exec\_cached\_plans

outer apply sys.dm\_exec\_sql\_text(plan\_handle)

outer apply sys.dm\_exec\_query\_plan(plan\_handle)

where text like '%use sql' + 'workshops%'

go

select sum(size\_in\_bytes) / 1024 as size\_in\_kb, count(\*) as count

from sys.dm\_exec\_cached\_plans

outer apply sys.dm\_exec\_sql\_text(plan\_handle)

outer apply sys.dm\_exec\_query\_plan(plan\_handle)

where text like '%use sql' + 'workshops%'

go

select sum(size\_in\_bytes) / 1024 as size\_in\_kb, count(\*) as count

from sys.dm\_exec\_cached\_plans

outer apply sys.dm\_exec\_sql\_text(plan\_handle)

outer apply sys.dm\_exec\_query\_plan(plan\_handle)

where text like '%use sql' + 'workshops%' and objtype = 'Adhoc' and usecounts = 1

go

select sum(size\_in\_bytes) / 1024 as size\_in\_kb, count(\*) as count

from sys.dm\_exec\_cached\_plans

outer apply sys.dm\_exec\_sql\_text(plan\_handle)

outer apply sys.dm\_exec\_query\_plan(plan\_handle)

where text like '%use sql' + 'workshops%' and objtype = 'Adhoc' and usecounts > 1

go

exec sp\_configure 'optimize for ad hoc workloads'

go

exec sp\_configure 'optimize for ad hoc workloads', 1

reconfigure

go

exec sp\_configure 'optimize for ad hoc workloads'

go

exec sp\_configure 'optimize for ad hoc workloads'

go

exec sp\_configure 'optimize for ad hoc workloads', 0

reconfigure

go

exec sp\_configure 'optimize for ad hoc workloads'

go

select query\_hash, query\_plan\_hash, count(\*) from sys.dm\_exec\_query\_stats

outer apply sys.dm\_exec\_sql\_text(plan\_handle)

outer apply sys.dm\_exec\_query\_plan(plan\_handle)

where text like '%use sql' + 'workshops%'

group by query\_hash, query\_plan\_hash

go

SELECT usecounts, cacheobjtype, plan\_handle

FROM sys.dm\_exec\_cached\_plans

WHERE cacheobjtype = 'Compiled Plan'

SELECT query\_plan

FROM sys.dm\_exec\_query\_plan(0x06000100BFBA052ED062C1655F02000001000000000000000000000000000000000000000000000000000000)

SELECT name, in\_use\_count, original\_cost, pages\_kb

FROM sys.dm\_os\_memory\_cache\_entries

WHERE type in ('CACHESTORE\_OBJCP', 'CACHESTORE\_SQLCP', 'CACHESTORE\_PHDR')

SELECT \* FROM Sales.SalesOrderHeader

WHERE SalesOrderID = 43666

OPTION (MAXDOP 1)

--We can view the statistics on the table with the following query :

SELECT s.name, c.name, auto\_created

FROM sys.stats s

JOIN sys.columns c

ON s.object\_id = c.object\_id and s.stats\_id = c.column\_id

WHERE s.object\_id = object\_id('HumanResources.Employee')

* 1. Temporarily disable automatic creation of statistics at the database level:

ALTER DATABASE AdventureWorks2012

SET AUTO\_CREATE\_STATISTICS OFF

ALTER DATABASE AdventureWorks2012 SET AUTO\_CREATE\_STATISTICS ON

* 1. A plan\_handle is a hash value that represents a specific execution plan

SELECT \* FROM sys.dm\_exec\_requests

CROSS APPLY

sys.dm\_exec\_query\_plan(plan\_handle)

* 1. The sys.dm\_exec\_query\_stats DMV contains one row per query statement within the cached plan

SELECT \* FROM sys.dm\_exec\_query\_stats

CROSS APPLY

sys.dm\_exec\_query\_plan(plan\_handle)

* 1. Retrieve top 10 queries by usage

SELECT TOP 10 total\_worker\_time/execution\_count AS avg\_cpu\_time,

plan\_handle, query\_plan

FROM sys.dm\_exec\_query\_stats

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

ORDER BY avg\_cpu\_time DESC

* 1. Removing Plans from the Plan Cache

DBCC FREEPROCCACHE --statement can be used to remove all the entries from the plan cache.

DBCC FREESYSTEMCACHE --statement can be used to remove all the elements from the plan cache or only the elements associated with a Resource Governor pool name.

DBCC FLUSHPROCINDB --can be used to remove all the cached plans for a particular database.

DBCC DROPCLEANBUFFERS --statement can be used to remove all the buffers from the buffer pool.

--SET STATISTICS TIME / IO

SET STATISTICS TIME --to see the number of milliseconds required to parse, compile, and execute each statement. For example, run

SET STATISTICS TIME ON

--and then run the following query:

SELECT DISTINCT(CustomerID)

FROM Sales.SalesOrderHeader

--and turn it off :

SET STATISTICS TIME OFF

SET STATISTICS IO --displays the amount of disk activity generated by a query. To enable it, run the following statement:

SET STATISTICS IO ON

--Run this next statement to clean all the buffers from the buffer pool to make sure that no pages for this table are loaded in memory:

DBCC DROPCLEANBUFFERS

SELECT \* FROM sys.dm\_exec\_query\_optimizer\_info;

* 1. How many optimizations are performed?

SELECT occurrence AS Optimizations FROM sys.dm\_exec\_query\_optimizer\_info

WHERE counter = 'optimizations';

* 1. What is the average elapsed time per optimization?

SELECT ISNULL(value,0.0) AS ElapsedTimePerOptimization

FROM sys.dm\_exec\_query\_optimizer\_info WHERE counter = 'elapsed time';

* 1. What fraction of optimized queries contained a subquery?

SELECT (SELECT CAST (occurrence AS float)

FROM sys.dm\_exec\_query\_optimizer\_info WHERE counter = 'contains subquery') /

(SELECT CAST (occurrence AS float) FROM sys.dm\_exec\_query\_optimizer\_info WHERE counter = 'optimizations')

AS ContainsSubqueryFraction;

* 1. Percentage of the total optimizations :

SELECT (SELECT CAST (occurrence AS float)

FROM sys.dm\_exec\_query\_optimizer\_info WHERE counter = 'contains subquery') /

(SELECT CAST (occurrence AS float) FROM sys.dm\_exec\_query\_optimizer\_info WHERE counter = 'optimizations') \* 100

AS ContainsSubqueryFraction ;

* 1. View percentage of optimization on query with hints

SELECT (SELECT occurrence FROM sys.dm\_exec\_query\_optimizer\_info WHERE counter = 'hints' ) \* 100.0 /

(SELECT occurrence FROM sys.dm\_exec\_query\_optimizer\_info WHERE counter = 'optimizations' )

* 1. Check optimizations for a specific workload

SELECT \*

INTO after\_query\_optimizer\_info

FROM sys.dm\_exec\_query\_optimizer\_info

GO

SELECT \*

INTO before\_query\_optimizer\_info

FROM sys.dm\_exec\_query\_optimizer\_info

GO

-- real execution starts

GO

SELECT \*

INTO before\_query\_optimizer\_info

FROM sys.dm\_exec\_query\_optimizer\_info

GO

-- insert your query here

SELECT \*

FROM Person.Address

-- keep this to force a new optimization

OPTION (RECOMPILE)

GO

SELECT \*

INTO after\_query\_optimizer\_info

FROM sys.dm\_exec\_query\_optimizer\_info

GO

SELECT a.counter,

(a.occurrence - b.occurrence) AS occurrence,

(a.occurrence \* a.value - b.occurrence \*

b.value) AS value

FROM before\_query\_optimizer\_info b

JOIN after\_query\_optimizer\_info a

ON b.counter = a.counter

WHERE b.occurrence <> a.occurrence

DROP TABLE before\_query\_optimizer\_info

DROP TABLE after\_query\_optimizer\_info

* 1. To check space used by tables tab72 and tab73 in each NUMA node

use test\_numa

select distinct db\_name(database\_id), object\_name(p.object\_id, bd.database\_id) as tabname, numa\_node, count(\*) \* 8 / 1024 as SizeMB from sys.dm\_os\_buffer\_descriptors bd

inner join sys.allocation\_units au on (bd.allocation\_unit\_id = au.allocation\_unit\_id)

inner join sys.partitions p on (au.container\_id = p.hobt\_id)

where bd.database\_id in(db\_id('test\_numa')) and object\_id in (object\_id('tab72'), object\_id('tab73'))

group by db\_name(database\_id), object\_name(p.object\_id, bd.database\_id), numa\_node

order by db\_name(database\_id), object\_name(p.object\_id, bd.database\_id), numa\_node

go

* 1. To check memory allocated per node in SQL Server

use master

select memory\_node\_id, pages\_kb / 1024 as pagesMB

from sys.dm\_os\_memory\_nodes

where memory\_node\_id < 64

go

* 1. Check wait stats and page\_latch\_wait\_count > 0

select wait\_time\_ms / (case waiting\_tasks\_count when 0 then NULL else waiting\_tasks\_count end) as AvgWaitMS, \*

from sys.dm\_os\_wait\_stats

where wait\_type in ('PAGELATCH\_EX', 'PAGELATCH\_SH')

go

select object\_name(object\_id) as object\_name, page\_latch\_wait\_count, page\_latch\_wait\_in\_ms

from sys.dm\_db\_index\_operational\_stats(db\_id(), NULL, NULL, NULL)

where page\_latch\_wait\_count > 0

order by page\_latch\_wait\_in\_ms desc

go