**Performance Tuning**

**DB – SQL Server 2008 R2**

**&**

**DB Maintenance Document**

**Revision History**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Ver. #** | **Description of change** | **Sections Affected** | **Created/ Modified By** | **Approved By** | **Date of issue** |
| 0.1 | Initial Draft |  | Alagarsamy T |  | 23-Jan-13 |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

**Table of Contents**

[1 DB Performance Tuning Guidelines 2](#_Toc346712141)

[2 Checklist for Analyzing Slow-Running Queries 10](#_Toc346712142)

[3 SQL Maintenance Activity 11](#_Toc346712143)

[3.1 To Check the memory Status 11](#_Toc346712144)

[3.2 To Check the unused cache 11](#_Toc346712145)

[3.3 To pull the physical memory & server information 11](#_Toc346712146)

[3.4 To Calculate the TPM 11](#_Toc346712147)

[3.5 To check the current concurrent users (Processes at that time). 12](#_Toc346712148)

[3.6 To check the RAM Memory 12](#_Toc346712149)

[3.7 To Identification of unused Index 13](#_Toc346712150)

[3.8 Driven Queries (Sample) 14](#_Toc346712151)

[3.9 Table Buffer Usage 14](#_Toc346712152)

[3.10 Re-Indexing 15](#_Toc346712153)

[3.11 Rebuild the Index 17](#_Toc346712154)

[3.12 Scripts for Defragmentation of the Index 18](#_Toc346712155)

[3.13 Scripts for generating Drop & Create Index 20](#_Toc346712156)

[3.14 To get the aggregate performance statistics from Plan Cache 23](#_Toc346712157)

# DB Performance Tuning Guidelines

1. Do not use OPENROWSET and OLEDB command anywhere in the SQL code.
2. For the bulk column fetch, first retrieve the main field based on the different condition and then fetch all required columns based on the derived main fields.
   1. E.g. In field status, first fetch the Work Orders & then fetch the details of Work Orders
   2. Similarly, fetch the Job Id first in Technician Job Details & then fetch all the other columns based on the Job id which was retrieved initially.
3. Always do not fetch all the grids in a screen. Analyse the screen and based on the result, proceed for coding
   1. E.g. In field status, there are three grids where we can display the output for
      1. Job Summary
      2. List View
      3. Total Count
   2. In this case, “Job Summary” & “Total Count” outputs will not vary unless we modify the main input filter conditions.
   3. For sorting any columns in the list view, we should not load the “Job Summary” & “Total Count” columns again, since they would have loaded already. (UI changes are needed)
   4. Based on the sorting, we can load only the List view.
4. Avoid unwanted “where” condition while framing SQL Query dynamically
   1. E.g, if the Order Date input is passed as “NULL”, then we should not send any filter for Order Date.

**Dont’s:**

and wo.priority\_id = isnull(pi\_priority\_id,wo.priority\_id)

**Do’s:**

if @pi\_priority\_id is not null

begin

select @l\_str = @l\_str + ' and wo.priority\_id = '+convert(nvarchar(200),@pi\_priority\_id) +' '

end

1. Do not use functions in the left side of the “where” condition
   1. E.g. where isnull(order\_date,’’) between @pi\_start and @pi\_end
   2. where cast(order\_date as date) between @pi\_start and @pi\_end
   3. where dbo.fn\_Xdate(order\_date) between @pi\_start and @pi\_end
2. All the images should be stored in Fileserver / Filestream. Also it would be better if it is read from UI side.
3. Avoid using functions in “where” clause.

**Dont’s:**

and isnull(reviewer\_id,0) = isnull(@pi\_reviewer\_id,0)

**Do’s:**

and (reviewer\_id is not null and reviewer\_id = isnull(@pi\_reviewer\_id,0))

1. Join the different tables based on the volume from small to large

**Dont’s:**

from #Work\_order\_Result wo

join work\_order wo1 (nolock) on wo1.work\_order\_id= wo.work\_order\_id

**Do’s:**

From #work\_order\_result wo

join work\_order wo1 (nolock) on wo.work\_order\_id= wo1.work\_order\_id

1. In a Join, keep ‘inner join” first, after that keep all the “left join” , It will reduce the IO Reads.

**Dont’s:**

from #Work\_order\_Result wo1

Join ref\_work\_item\_type rwit (nolock)

on rwit.work\_item\_type\_id = wo1.work\_item\_type\_id

left join ref\_organization ro (nolock) on ro.id\_org = wo1.vendor\_id

left join ref\_priority rp (nolock) on rp.priority\_id= wo1.priority\_id

join vw\_property\_details re on re.Property\_Id = wo1.Property\_Id

**Do’s:**

from #Work\_order\_Result wo1

Join ref\_work\_item\_type rwit (nolock)

on rwit.work\_item\_type\_id = wo1.work\_item\_type\_id

join vw\_property\_details re on re.Property\_Id = wo1.Property\_Id

left join ref\_organization ro (nolock) on ro.id\_org = wo1.vendor\_id

left join ref\_priority rp (nolock) on rp.priority\_id= wo1.priority\_id

1. Use Nolock for well known masters for fetching stored procedures.

**Dont’s:**

from #Work\_order\_Result wo1

join ref\_priority rp on rp.priority\_id= wo1.priority\_id

join vw\_property\_details re on re.Property\_Id = wo1.Property\_Id

**Do’s:**

from #Work\_order\_Result wo1

join ref\_priority rp (nolock) on rp.priority\_id= wo1.priority\_id

join vw\_property\_details re (noexpand) on re.Property\_Id= wo1.Property\_Id

1. Use Periodic Index, instead of reindex. With the help of Show Index Statistics, we should use index defragment.

**Eg:**

USE FPREO;

GO

DBCC SHOW\_STATISTICS ("ref\_meta\_data", pk\_ref\_meta\_data\_meta\_data\_id);

1. Always use covering index for non-clustered index.

**Dont’s:**

CREATE NONCLUSTERED INDEX IX\_OrderDetailDateProdSold ON dbo.OrderDetail

( ProductID, OrderDate)

**Do’s:**

CREATE NONCLUSTERED INDEX IX\_OrderDetailDateProdSold ON dbo.OrderDetail

( ProductID, OrderDate) INCLUDE (QtySold);

1. If there is an index for any column, then check whether the read and write process are performed by the respective index. It the read process does not happen for a long period, then we do not need that index.
2. Try to have views for addresses, since we need to map many tables to fetch the address in various queries. Please create an index for the respective views.
3. Also try to have views for Job details like Category, Service Type, Service Request in order to avoid the joins for different tables in many queries.
4. Test every Execute Statements with the option “with recompile”

**Eg:** exec get\_record\_search\_list\_FS @pi\_sp\_name=N'[get\_order\_status]',

@sp\_in\_params=N'"null~null~null~null~null~null~null~null~null~Null~Null~Null~0, 5, 11, 12, 7, 4, 3, 2, 6, 9, 47, 8, 10~4~Null~null~null~null~1~null~null~null~1~Null~1~1"'

With Recompile

1. For every SP execution time , we need to verify the IO. Depending upon the IO , CPU Utilization will differ.

**Ex:** Set Statistics IO on

EXEC ‘SP Name’

Set Statistics IO OFF

Logical read should be always less. Index for specific columns will help to reduce logical Reads.

1. If a condition contains “OR”, then put that condition first
   1. Where (wa.tech\_id = @pi\_tech\_id or @pi\_login\_id = 1)
   2. Please do write the above code as where (@pi\_login\_id = 1 or wa.tech\_id = @pi\_tech\_id)
2. If N select queries are mentioned in a SP, try to convert all the select queries into different SPs.
3. Remove “Not in” condition from all coding.

**Dont’s:**

and state\_id not in (31,32,33,34)

**Do’s:**

and state\_id in (25,26,27,28)

1. Avoid use ‘distinct’ key word
2. Create a function with ‘SCHEMABINDING’

**Ex:**

CREATE FUNCTION SchemaBinded(@INPUT INT)

RETURNS INT WITH SCHEMABINDING

BEGIN

RETURN @INPUT \* 2 + 50

END

GO

<http://www.mssqltips.com/sqlservertip/1692/using-schema-binding-to-improve-sql-server-udf->performance/

1. To Reduce the IO of each stored procedures with the help of user defined function (use Computed Columns /persisted Columns )

**Ex:**

alter table work\_order add wo\_category\_id as [dbo].[fn\_wo\_category\_id](work\_order\_id) persisted

go

**alter table work\_order add wo\_category\_id as [dbo].[fn\_wo\_category\_id](work\_order\_id)**

**go –- this is better than the above persited column**

It will help to avoid Demoralization of the table.

1. Without Group by we will be able to get the No. of records ‘count(\*)’ Using

count(\*) over( partition by (select 1)) as Total\_Count

‘Record Count’ in the specific select statement itself.

1. XML is always faster than the CSV command. It will not have parsing methodology.

**Ex:**

declare @l\_category nvarchar(max)

select @l\_category =

substring(category ,1,len(category) -1)

from (

select

( select convert(nvarchar(1000),wj1.work\_order\_category\_id ) + ','

from dbo.wo\_job wj1 (nolock)

where wj1.work\_order\_id = wj.work\_order\_id

order by work\_order\_id

for xml path('') ) as category

from dbo.wo\_job wj

where work\_order\_id = @pi\_work\_order\_id

)a

1. Always use “EXISTS” query instead of “NOT EXISTS”.

**Dont’s:**

and not exists(select 1 from work\_order wo1 (nolock)

where wo.work\_order\_id = wo1.work\_order\_id

and wo1.reviewer\_id = @pi\_login\_id

)

**Do’s:**

and exists(select 1 from work\_order wo1 (nolock)

where wo.work\_order\_id = wo1.work\_order\_id

and wo1.reviewer\_id = @pi\_login\_id )

1. If temporary tables are used in stored procedures, then drop temp table at the end of the sp.

**Ex:**

IF OBJECT\_ID('tempdb..#temp\_login') IS NOT NULL DROP TABLE #temp\_login

1. Avoid using sub queries / joins with the same table in a stored procedure. Instead use “EXISTS” which will improve the performance drastically.

**Dont’s:**

join ref\_property re (nolock)

join ref\_property\_stage re1 (nolock) on re1.property\_id= re.property\_id

and re1.property\_stage\_id in (1,2)

where re.property\_id = @pi\_property\_id

**Do’s:**

join ref\_property re (nolock)

where re.property\_id = @pi\_property\_id

and exists (select 1 ref\_property re1 (nolock)

where re1.property\_id = re.property\_id

and re1.property\_stage\_id in (1,2)

)

1. Avoid using global temporary “##temp” tables in a stored procedures. It will not support when concurrent users perform simultaneously. i.e., same value will be passed to all users which leads to wrong data entry.

**Dont’s:**

create table ##temp

( sl\_no int identity(1,1),

property\_id int

)

**Do’s:**

create table #temp

( sl\_no int identity(1,1),

property\_id int

)

1. Avoid using “SELECT INTO” keyword for creating temp tables. Instead, create a temporary table and then insert the records.

**Dont’s:**

Select id,designation

into #temp

from ref\_designation

where is\_active = 1

**Do’s:**

Create table #temp (id bigint, designation nvarchar(400))

Insert into #temp (id, desigantion)

Select id,designation from ref\_designation (nolock) where is\_active = 1

IF OBJECT\_ID('tempdb..#temp') IS NOT NULL DROP TABLE #temp

1. While insertion, use column list in the insert statement. It will avoid unwanted errors when a new column is added in the specified table.

**Dont’s:**

Insert into #temp

Select work\_order\_id, work\_order\_status\_work\_item\_type

from work\_order where work\_order\_id = @pi\_work\_order\_id

**Do’s:**

Insert into #temp (work\_order\_id, work\_order\_status\_work\_item\_type)

Select work\_order\_id, work\_order\_status\_work\_item\_type

from work\_order (nolock) where work\_order\_id = @pi\_work\_order\_id

1. Use “NOEXPAND” keyword, when views are used in joins.

**EX:**

from #Work\_order\_Result wo1

join ref\_priority rp (nolock) on rp.priority\_id= wo1.priority\_id

join vw\_property\_details re (noexpand) on re.Property\_Id= wo1.Property\_Id

1. Avoid selecting records by using “\*” keyword in stored procedures. Instead, provide the column list which needs to be given as output.

**Dont’s:**

Select \* from ref\_property (nolock) where property\_id = @pi\_property\_id

**Do’s:**

Select property\_id, property\_code, property\_number,priority\_id,is\_active

from ref\_property (nolock) where property\_id = @pi\_property\_id

34. Instead of including “Nolock” for all the tables in the stored procedure, we can set the isolation level as “SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED” in the top of stored procedures.

It will play the exact role of “Nolock” .It refers all the tables which referred in the respective stored procedure. It would be applicable for the connections (Session ID) even though the respective Stored procedures contains the nested Stored procedures/Remote Procedure. This should applicable for only fetch stored procedures.

# Checklist for Analyzing Slow-Running Queries

* Slow network communication.
* Inadequate memory in the server computer, or not enough memory available for SQL Server.
* Lack of useful statistics
* Lack of useful indexes.
* Lack of useful indexed views.
* Lack of useful data striping.
* Lack of useful partitioning.

Reference: <http://msdn.microsoft.com/en-us/library/ms177500.aspx>

# SQL Maintenance Activity

## To Check the memory Status

DBCC MEMORYSTATUS

## To Check the unused cache

DBCC FREESYSTEMCACHE ('ALL') WITH MARK\_IN\_USE\_FOR\_REMOVAL;

## To pull the physical memory & server information

select \* from sys.dm\_os\_sys\_info

select \* from sys.dm\_os\_sys\_memory

select convert(numeric(5,2),(total\_physical\_memory\_kb/1024.0/1024.0)) as Total, convert(numeric(5,2),(available\_physical\_memory\_kb/1024.0/1024.0)) as Available, system\_memory\_state\_desc ,( Select ((bpool\_committed\*8)/1024.0/1024.0) from sys.dm\_os\_sys\_info (nolock) ) As SQLUseage from sys.dm\_os\_sys\_memory  (nolock)

## To Calculate the TPM

DECLARE @cntr\_value1 bigint

DECLARE @cntr\_value2 bigint

t:

SELECT @cntr\_value1 = cntr\_value

FROM sys.dm\_os\_performance\_counters

WHERE counter\_name = 'transactions/sec'

AND object\_name = 'SQLServer:Databases'

AND instance\_name = 'FPREOPRO'

WAITFOR DELAY '00:00:01'

SELECT @cntr\_value2 = cntr\_value

FROM sys.dm\_os\_performance\_counters

WHERE counter\_name = 'transactions/sec'

AND object\_name = 'SQLServer:Databases'

AND instance\_name = 'FPREOPRO'

insert into mem\_transaction\_counter

Select @cntr\_value2-@cntr\_value1,getdate()

goto t

## To check the current concurrent users (Processes at that time).

select GETDATE() as 'Time',COUNT(\*) as 'Connection\_count'

from master.dbo.sysprocesses p

join master.dbo.sysdatabases d on p.dbID = d.dbID

where p.dbid = db\_id()

## To check the RAM Memory

SELECT GETDATE() As Time, physical\_memory\_in\_bytes/1073741824.0 as [Physical Memory\_GB] FROM sys.dm\_os\_sys\_info

## To Identification of unused Index

 Scan:   An index scan is a complete read of all of the leaf pages in the index.

 Seek:    An index seeks is an operation where SQL uses the b-tree structure to locate either a specific value or the beginning of a range of value

If both are 0 then the index is useless.

1)

WITH indexstats ([Table],[Index],[Reads],[Writes],[Rows]) AS ( SELECT usr.[name] + '.' + obj.[name] [Table], ixs.[name] [Index] ,

usage.user\_seeks + usage.user\_scans + usage.user\_lookups [Reads], usage.[user\_updates] [Writes],

(SELECT SUM(sp.[rows]) FROM sys.partitions sp

WHERE usage.OBJECT\_ID = sp.object\_id AND sp.index\_id = usage.index\_id) [Rows]

FROM sys.dm\_db\_index\_usage\_stats usage INNER JOIN sys.indexes ixs ON usage.[object\_id] = ixs.[object\_id]

AND ixs.[index\_id] = usage.[index\_id] INNER JOIN sys.objects obj ON usage.[object\_id] = obj.[object\_id]

INNER JOIN sys.sysusers usr ON obj.[schema\_id] = usr.[uid] WHERE usage.database\_id = DB\_ID()

AND usage.index\_id > 0 AND OBJECTPROPERTY(usage.[object\_id], 'IsUserTable') = 1 )

SELECT 'Drop Index ' +[INDEX] + ' on ' +[table],\* FROM indexstats WHERE Reads = 0 and ([index] not like '%pk\_%' and [index] not like '%uk\_%' )ORDER BY [Rows] DESC, [Index]

go

2)

DECLARE @dbid INT

SELECT @dbid = DB\_ID(DB\_NAME())

SELECT OBJECTNAME = OBJECT\_NAME(I.OBJECT\_ID),

INDEXNAME = I.NAME,

I.INDEX\_ID

FROM SYS.INDEXES I

JOIN SYS.OBJECTS O

ON I.OBJECT\_ID = O.OBJECT\_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)

ORDER BY OBJECTNAME,

I.INDEX\_ID,

INDEXNAME ASC

GO

## To Identification of Missing Index

SELECT mid.statement ,migs.avg\_total\_user\_cost \* (migs.avg\_user\_impact / 100.0) \* (migs.user\_seeks + migs.user\_scans) AS improvement\_measure,OBJECT\_NAME(mid.Object\_id),'CREATE INDEX [idx\_' + 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 JOIN sys.dm\_db\_missing\_index\_group\_stats migs ON migs.group\_handle = mig.index\_group\_handle  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

## Driven Queries (Sample)

Optimize Parameter Driven Queries with SQL Server OPTIMIZE FOR Hint:

DECLARE @Country VARCHAR(20)   
SET @Country = 'US'       
  
SELECT \*   
    FROM Sales.SalesOrderHeader h, Sales.Customer c,    
        Sales.SalesTerritory t   
    WHERE h.CustomerID = c.CustomerID   
        AND c.TerritoryID = t.TerritoryID   
        AND CountryRegionCode = @Country   
   OPTION (OPTIMIZE FOR (@Country = 'US'))

Reference for the above:

<http://www.mssqltips.com/sqlservertip/1354/optimize-parameter-driven-queries-with-sql-server-optimize-for-hint/>

## Table Buffer Usage

SELECT

      obj.[name],

      i.[name],

      i.[type\_desc],

      count(\*)AS Buffered\_Page\_Count ,

      count(\*) \* 8192.0 / (1024 \* 1024) as Buffer\_MB

FROM sys.dm\_os\_buffer\_descriptors AS bd

    INNER JOIN

    (

        SELECT object\_name(object\_id) AS name

            ,index\_id ,allocation\_unit\_id, object\_id

        FROM sys.allocation\_units AS au

            INNER JOIN sys.partitions AS p

                ON au.container\_id = p.hobt\_id

                    AND (au.type = 1 OR au.type = 3)

        UNION ALL

        SELECT object\_name(object\_id) AS name

            ,index\_id, allocation\_unit\_id, object\_id

        FROM sys.allocation\_units AS au

            INNER JOIN sys.partitions AS p

                ON au.container\_id = p.hobt\_id

                    AND au.type = 2

    ) AS obj

        ON bd.allocation\_unit\_id = obj.allocation\_unit\_id

LEFT JOIN sys.indexes i on i.object\_id = obj.object\_id AND i.index\_id = obj.index\_id

WHERE database\_id = db\_id()

GROUP BY obj.name, obj.index\_id , i.[name],i.[type\_desc]

ORDER BY Buffered\_Page\_Count DESC

## Re-Indexing

CREATE PROCEDURE upd\_reindex

AS

begin

set nocount on

declare @po\_error\_code nvarchar(4000),

@po\_severity tinyint ,

@l\_incr int ,

@l\_count int ,

@l\_table\_name nvarchar(200)

--if exists(select 1 from sys.tables where name = 'TableRowCount')

--begin

-- drop table [TableRowCount]

--end

--CREATE TABLE [TableRowCount](

-- TableName sysname,

-- [TableRowCount] int )

----EXEC sp\_MSForEachTable 'INSERT [TableRowCount](TableName, [TableRowCount]) SELECT ''?'', COUNT(\*) FROM ?'

--EXEC sp\_MSforeachtable @command1 = "print '?' DBCC DBREINDEX ('?', ' ', 80)"

--EXEC sp\_updatestats

--select \* from [TableRowCount]

create table #temp\_reindex

(

sl\_no int ,

table\_name nvarchar(200)

)

insert into #temp\_reindex

(

sl\_no,

table\_name

)

SELECT row\_number() over(order by name),

name

FROM sys.tables

where type = 'U'

if exists (select 1 from #temp\_reindex)

begin

select @l\_incr = 1

select @l\_count = COUNT(\*) from #temp\_reindex

while @l\_incr < = @l\_count

begin

select @l\_table\_name = table\_name

from #temp\_reindex tp

where sl\_no = @l\_incr

PRINT 'Reindexing Table: ' + @l\_table\_name

DBCC DBREINDEX(@l\_table\_name, '', 80)

select @l\_incr = @l\_incr + 1

select @l\_table\_name = null

end

end

EXEC sp\_updatestats

set nocount off

end

 go

## Rebuild the Index

create procedure rebuild\_index

as

begin

DECLARE @Database VARCHAR(255)

DECLARE @Table VARCHAR(255)

DECLARE @cmd NVARCHAR(500)

DECLARE @fillfactor INT

SET @fillfactor = 90

DECLARE DatabaseCursor CURSOR FOR

SELECT name FROM MASTER.dbo.sysdatabases

WHERE dbid = db\_id()

ORDER BY 1

OPEN DatabaseCursor

FETCH NEXT FROM DatabaseCursor INTO @Database

WHILE @@FETCH\_STATUS = 0

BEGIN

SET @cmd = 'DECLARE TableCursor CURSOR FOR SELECT ''['' + table\_catalog + ''].['' + table\_schema + ''].['' +

table\_name + '']'' as tableName FROM ' + @Database + '.INFORMATION\_SCHEMA.TABLES

WHERE table\_type = ''BASE TABLE'''

-- create table cursor

EXEC (@cmd)

OPEN TableCursor

FETCH NEXT FROM TableCursor INTO @Table

WHILE @@FETCH\_STATUS = 0

BEGIN

IF (@@MICROSOFTVERSION / POWER(2, 24) >= 9)

BEGIN

-- SQL 2005 or higher command

SET @cmd = 'ALTER INDEX ALL ON ' + @Table + ' REBUILD WITH (FILLFACTOR = ' + CONVERT(VARCHAR(3),@fillfactor) + ')'

EXEC (@cmd)

END

ELSE

BEGIN

-- SQL 2000 command

DBCC DBREINDEX(@Table,' ',@fillfactor)

END

FETCH NEXT FROM TableCursor INTO @Table

END

CLOSE TableCursor

DEALLOCATE TableCursor

FETCH NEXT FROM DatabaseCursor INTO @Database

END

CLOSE DatabaseCursor

DEALLOCATE DatabaseCursor

end

## Scripts for Defragmentation of the Index

create procedure defragment\_index

as

begin

SET NOCOUNT ON;

DECLARE @tablename varchar(255);

DECLARE @execstr varchar(400);

DECLARE @objectid int;

DECLARE @indexid int;

DECLARE @frag decimal;

DECLARE @maxfrag decimal;

-- Decide on the maximum fragmentation to allow for.

SELECT @maxfrag = 30.0;

-- Declare a cursor.

DECLARE tables CURSOR FOR

SELECT TABLE\_SCHEMA + '.' + TABLE\_NAME

FROM INFORMATION\_SCHEMA.TABLES

WHERE TABLE\_TYPE = 'BASE TABLE';

-- Create the table.

CREATE TABLE #fraglist (

ObjectName char(255),

ObjectId int,

IndexName char(255),

IndexId int,

Lvl int,

CountPages int,

CountRows int,

MinRecSize int,

MaxRecSize int,

AvgRecSize int,

ForRecCount int,

Extents int,

ExtentSwitches int,

AvgFreeBytes int,

AvgPageDensity int,

ScanDensity decimal,

BestCount int,

ActualCount int,

LogicalFrag decimal,

ExtentFrag decimal);

-- Open the cursor.

OPEN tables;

-- Loop through all the tables in the database.

FETCH NEXT

FROM tables

INTO @tablename;

WHILE @@FETCH\_STATUS = 0

BEGIN

-- Do the showcontig of all indexes of the table

INSERT INTO #fraglist

EXEC ('DBCC SHOWCONTIG (''' + @tablename + ''')

WITH FAST, TABLERESULTS, ALL\_INDEXES, NO\_INFOMSGS');

FETCH NEXT

FROM tables

INTO @tablename;

END;

-- Close and deallocate the cursor.

CLOSE tables;

DEALLOCATE tables;

-- Declare the cursor for the list of indexes to be defragged.

DECLARE indexes CURSOR FOR

SELECT ObjectName, ObjectId, IndexId, LogicalFrag

FROM #fraglist

WHERE LogicalFrag >= @maxfrag

AND INDEXPROPERTY (ObjectId, IndexName, 'IndexDepth') > 0;

-- Open the cursor.

OPEN indexes;

-- Loop through the indexes.

FETCH NEXT

FROM indexes

INTO @tablename, @objectid, @indexid, @frag;

WHILE @@FETCH\_STATUS = 0

BEGIN

PRINT 'Executing DBCC INDEXDEFRAG (0, ' + RTRIM(@tablename) + ',

' + RTRIM(@indexid) + ') - fragmentation currently '

+ RTRIM(CONVERT(varchar(15),@frag)) + '%';

SELECT @execstr = 'DBCC INDEXDEFRAG (0, ' + RTRIM(@objectid) + ',

' + RTRIM(@indexid) + ')';

EXEC (@execstr);

FETCH NEXT

FROM indexes

INTO @tablename, @objectid, @indexid, @frag;

END;

-- Close and deallocate the cursor.

CLOSE indexes;

DEALLOCATE indexes;

-- Delete the temporary table.

DROP TABLE #fraglist;

end

## Scripts for generating Drop & Create Index

SELECT

ixz.object\_id,

tablename = QUOTENAME(scmz.name) + '.' + QUOTENAME((OBJECT\_NAME(ixz.object\_id))),

tableid = ixz.object\_id,

indexid = ixz.index\_id,

indexname = ixz.name,

isunique = INDEXPROPERTY (ixz.object\_id,ixz.name,'isunique'),

isclustered = INDEXPROPERTY (ixz.object\_id,ixz.name,'isclustered'),

indexfillfactor = INDEXPROPERTY (ixz.object\_id,ixz.name,'indexfillfactor'),

--SQL2008+ Filtered indexes:

CASE

WHEN ixz.filter\_definition IS NULL

THEN ''

ELSE ' WHERE ' + ixz.filter\_definition

END Filter\_Definition

--For 2005, which did not have filtered indexes, comment out the above CASE statement, and uncomment this:

INTO #tmp\_indexes

FROM sys.indexes ixz

INNER JOIN sys.objects obz

ON ixz.object\_id = obz.object\_id

INNER JOIN sys.schemas scmz

ON obz.schema\_id = scmz.schema\_id

WHERE ixz.index\_id > 0

AND ixz.index\_id < 255 ---- 0 = HEAP index, 255 = TEXT columns index

AND INDEXPROPERTY (ixz.object\_id,ixz.name,'ISUNIQUE') = 0 -- comment out to include unique and

AND INDEXPROPERTY (ixz.object\_id,ixz.name,'ISCLUSTERED') = 0 -- comment out to include PK's

ALTER TABLE #tmp\_indexes ADD keycolumns VARCHAR(4000), includes VARCHAR(4000)

GO

DECLARE @isql\_key VARCHAR(4000),

@isql\_incl VARCHAR(4000),

@tableid INT,

@indexid INT

DECLARE index\_cursor CURSOR

FOR

SELECT

tableid,

indexid

FROM #tmp\_indexes

OPEN index\_cursor

FETCH NEXT FROM index\_cursor INTO @tableid, @indexid

WHILE @@FETCH\_STATUS <> -1

BEGIN

SELECT @isql\_key = '', @isql\_incl = ''

SELECT --ixz.name, colz.colid, colz.name, ixcolz.index\_id, ixcolz.object\_id, \*

--key column

@isql\_key = CASE ixcolz.is\_included\_column

WHEN 0

THEN CASE ixcolz.is\_descending\_key

WHEN 1

THEN @isql\_key + COALESCE(colz.name,'') + ' DESC, '

ELSE @isql\_key + COALESCE(colz.name,'') + ' ASC, '

END

ELSE @isql\_key

END,

--include column

@isql\_incl = CASE ixcolz.is\_included\_column

WHEN 1

THEN CASE ixcolz.is\_descending\_key

WHEN 1

THEN @isql\_incl + COALESCE(colz.name,'') + ', '

ELSE @isql\_incl + COALESCE(colz.name,'') + ', '

END

ELSE @isql\_incl

END

FROM sysindexes ixz

INNER JOIN sys.index\_columns AS ixcolz

ON (ixcolz.column\_id > 0

AND ( ixcolz.key\_ordinal > 0

OR ixcolz.partition\_ordinal = 0

OR ixcolz.is\_included\_column != 0)

)

AND ( ixcolz.index\_id=CAST(ixz.indid AS INT)

AND ixcolz.object\_id=ixz.id

)

INNER JOIN sys.columns AS colz

ON colz.object\_id = ixcolz.object\_id

AND colz.column\_id = ixcolz.column\_id

WHERE ixz.indid > 0 AND ixz.indid < 255

AND (ixz.status & 64) = 0

AND ixz.id = @tableid

AND ixz.indid = @indexid

ORDER BY

ixz.name,

CASE ixcolz.is\_included\_column

WHEN 1

THEN ixcolz.index\_column\_id

ELSE ixcolz.key\_ordinal

END

--remove any trailing commas from the cursor results

IF LEN(@isql\_key) > 1 SET @isql\_key = LEFT(@isql\_key, LEN(@isql\_key) -1)

IF LEN(@isql\_incl) > 1 SET @isql\_incl = LEFT(@isql\_incl, LEN(@isql\_incl) -1)

--put the columns collection into our temp table

UPDATE #tmp\_indexes

SET keycolumns = @isql\_key,

includes = @isql\_incl

WHERE tableid = @tableid

AND indexid = @indexid

FETCH NEXT FROM index\_cursor INTO @tableid,@indexid

END --WHILE

CLOSE index\_cursor

DEALLOCATE index\_cursor

--remove invalid indexes, ie ones without key columns

DELETE FROM #tmp\_indexes WHERE keycolumns = ''

SET NOCOUNT ON

SELECT

'IF NOT EXISTS(SELECT \* FROM sys.indexes WHERE object\_id = OBJECT\_ID(N''' + ti.TABLENAME + '''' + ') AND name = N' + '''' + ti.INDEXNAME + '''' + ')' + ' ' +

'CREATE '

+ CASE WHEN ti.ISUNIQUE = 1 THEN 'UNIQUE ' ELSE '' END

+ CASE WHEN ti.ISCLUSTERED = 1 THEN 'CLUSTERED ' ELSE '' END

+ 'INDEX ' + QUOTENAME(ti.INDEXNAME)

+ ' ON ' + (ti.TABLENAME) + ' '

+ '(' + ti.keycolumns + ')'

+ CASE

WHEN ti.INDEXFILLFACTOR = 0 AND ti.ISCLUSTERED = 1 AND INCLUDES = '' THEN ti.Filter\_Definition + ' WITH (SORT\_IN\_TEMPDB = ON) ON [' + fg.name + ']'

WHEN INDEXFILLFACTOR = 0 AND ti.ISCLUSTERED = 0 AND ti.INCLUDES = '' THEN ti.Filter\_Definition + ' WITH (ONLINE = ON, SORT\_IN\_TEMPDB = ON) ON [' + fg.name + ']'

WHEN INDEXFILLFACTOR <> 0 AND ti.ISCLUSTERED = 0 AND ti.INCLUDES = '' THEN ti.Filter\_Definition + ' WITH (ONLINE = ON, SORT\_IN\_TEMPDB = ON, FILLFACTOR = ' + CONVERT(VARCHAR(10),ti.INDEXFILLFACTOR) + ') ON [' + fg.name + ']'

WHEN INDEXFILLFACTOR = 0 AND ti.ISCLUSTERED = 0 AND ti.INCLUDES <> '' THEN ' INCLUDE (' + ti.INCLUDES + ') ' + ti.Filter\_Definition + ' WITH (ONLINE = ON, SORT\_IN\_TEMPDB = ON) ON [' + fg.name + ']'

ELSE ' INCLUDE(' + ti.INCLUDES + ') ' + ti.Filter\_Definition + ' WITH (FILLFACTOR = ' + CONVERT(VARCHAR(10),ti.INDEXFILLFACTOR) + ', ONLINE = ON, SORT\_IN\_TEMPDB = ON) ON [' + fg.name + ']'

END

FROM #tmp\_indexes ti

JOIN sys.indexes i ON ti.Object\_id = i.object\_id and ti.indexname = i.name

JOIN sys.filegroups fg on i.data\_space\_id = fg.data\_space\_id

WHERE LEFT(ti.tablename,3) NOT IN ('sys', 'dt\_') --exclude system tables

ORDER BY

ti.tablename,

ti.indexid,

ti.indexname

--makes the drop

SELECT

'DROP INDEX '

+ ' ' + (tablename) + '.'

+ (indexname) + ''

FROM #tmp\_indexes

WHERE LEFT(tablename,4) NOT IN ('[sys', 'dt\_')

----Drop the temp table again

DROP TABLE #tmp\_indexes

## To get the aggregate performance statistics from Plan Cache

SELECT total\_logical\_reads, total\_logical\_writes,

total\_physical\_reads, total\_worker\_time,

total\_elapsed\_time, sys.dm\_exec\_sql\_text.TEXT

FROM sys.dm\_exec\_query\_stats

CROSS APPLY sys.dm\_exec\_sql\_text(plan\_handle)

WHERE total\_logical\_reads <> 0

AND total\_logical\_writes <> 0

ORDER BY (total\_logical\_reads + total\_logical\_writes) DESC