# Analysing Indexes Part 1

Actually what started as a small document has continued to grow. I'm working on performance tuning a well known application and as I document as I work so others can follow what I'm doing I thought I'd add it to my blog. There's nothing particularly startling in here but if you want to drill into index performance on your database this may help. As I remarked on Tony or maybe Simon's blog it's fine looking at small tables but when row counts are in hundreds of millions and secondary indexes hit 10Gb ( each ) then performance and storage become more critical - and difficult to test - it's not a quick task to drop and create a handful of indexes on a 250 million row table.

- SQL Server 2005 maintains system views which contain information concerning indexes.
- As these are views the information is not carried forward from a server restart
- The information returned is cumulative so great care must be taken in evaluating the results.
- To obtain consistent data and results we must store the information from the system views in permanent tables for later analysis.
- The presented queries assume no partitioning

## 1.0 Operational cost on indexes ( assuming no RFI )

| Operation | Read | Write |
|---|---|---|
| Select | Always | No |
| Insert | No | Always – every index |
| Update | Always | Only if row qualifies |
| Delete | Always | Only if row qualifies |

## 2.0 Index Information

| Object | Description | Notes |
|---|---|---|
| dbo.sysindexes | This is the SQL 2000 system table taken forward as a view. This table may not be supported going forward. | |
| sys.dm_db_index_physical_stats | This is actually a function and replaces dbcc showcontig | |
| sys.indexes | 2005 system table contains property information only for each index. | |
| sys.dm_db_index_usage_stats | System view which records the access usage of every index in the database | We can view the number of times this index has been used, the type of access and the last access date |
| sys.dm_db_index_operational_stats | System function which records | We can view |

| | | |
|---|---|---|
| | the operational cost of access to the index | information such as the number of pages, locks, latches and waits. |
| sys.objects | Contains information on database objects | |
| | | |
| sys.dm_db_missing_index_details | System view which stores information on indexes the optimiser considers are missing. | |
| sys.dm_db_missing_index_group_stats | System view which stores usage and access details for the missing indexes similar to sys.dm_db_index_usage_stats | We can also view the improvement that the index is computed to make, this is essentially the same type of output as shown by the Tuning Advisor. |
| sys.dm_db_missing_index_groups | Presumably this view will be of more use in SQL2008 | |
| sys.dm_db_missing_index_columns | A system function which returns the columns for a missing index | Requires the index handle as a parameter |

## 3.0 sys.dm_db_index_usage_stats information

http://msdn2.microsoft.com/en-us/library/ms188755.aspx

- There are groups of data columns which interest us
  - click on the link ( above ) for full table details
- User access counts
- User access dates
- System access counts ( e.g. update statistics, index rebuilds )
- System access dates

## 4.0 List unused tables/indexes

- This will largely indicate empty tables as maintenance tasks will create entries within sys.dm_db_index_usage_stats where there is data in the table/index
  - Update statistics *tablename*
  - Dbcc dbreindex(*tablename*)
- I'd normally full qualify queries with three part naming; this is omitted for this document.

```
--
-- run in database to analyse
--
Select object_name(i.object_id) as TableName,isnull(i.name,'HEAP') as
IndexName, i.index_id
from sys.indexes i join sys.objects o on o.object_id = i.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 = db_id() )
```

```
order by object_name(i.object_id),i.index_id;
```

## 5.0 Query to filter by user access

- This sorts by lack of user access
- This is a count of the times there was a seek, scan or lookup on the index/table
- I've discounted updates from my criteria as an update will occur regardless of the index being used for selects
- Essentially any index with zero as a value for seek, scan and lookup will be unused.

```
--
-- run in database to be analysed
--
select object_name(s.object_id) as TableName,isnull(i.name,'HEAP') as IndexName,
case i.index_id
when 0 then 'HEAP'
when 1 then 'CLUS'
else 'NC'
end,
user_seeks as Seeks, user_scans as Scans, user_lookups as Lookups, user_updates
as Updates
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
where database_id = db_id() and objectproperty(s.object_id,'IsUserTable') = 1
order by (user_seeks + user_scans + user_lookups ) asc;
```

- Remember that these values are cumulative since start up so we must snapshot the data on a regular basis to enable proper analysis
- We must also collect data over a suitable time period based upon the usages of your database.
- In an environment which runs, say, month end processes, it is vital that we capture the full work life cycle.
- This is especially important if the CEO or MD has a favourite report they run once a month and you remove the index(es) which make this run quickly.

## 6.0 Capture snapshot of index activity

```
--
-- Run this is the database to be analysed
--
Insert into dbo.Unused_indexes
SELECT getdate(),* FROM sys.dm_db_index_usage_stats
WHERE database_id = DB_ID()
and (last_user_seek < convert(char(11),getdate()-1) or last_user_seek is null)
and (last_user_scan < convert(char(11),getdate()-1) or last_user_scan is null);
```

- I would schedule this to run every night just after midnight

- I’m collecting information where an index has not been used in the preceding 24 hours for a user seek or user scan. ( see **Working with snapshots** for explanation of this )
- The collection date is added for clarity.
- A primary key could be defined on the columns TheDate,database_id,object_id,index_id
- The table is a Heap

## 6.1 Table to capture index activity

```
CREATE TABLE dbo.Unused_Indexes
(
	TheDate datetime NOT NULL,
	database_id smallint NOT NULL,
	[object_id] int NOT NULL,
	index_id int NOT NULL,
	user_seeks bigint NOT NULL,
	user_scans bigint NOT NULL,
	user_lookups bigint NOT NULL,
	user_updates bigint NOT NULL,
	last_user_seek datetime NULL,
	last_user_scan datetime NULL,
	last_user_lookup datetime NULL,
	last_user_update datetime NULL,
	system_seeks bigint NOT NULL,
	system_scans bigint NOT NULL,
	system_lookups bigint NOT NULL,
	system_updates bigint NOT NULL,
	last_system_seek datetime NULL,
	last_system_scan datetime NULL,
	last_system_lookup datetime NULL,
	last_system_update datetime NULL
)
```

## 7.0 sys.dm_db_index_operational_stats

http://msdn2.microsoft.com/en-us/library/ms174281.aspx

- There are groups of data columns which interest us
  - click on the link ( above ) for full table details
- This is a very important table which contains a wealth of information
- We can view waits, page splits, and volume of access at leaf and non-leaf levels

## 8.0 View index usage by work - Selects

- This query shows all indexes, reads to top , writes to bottom

```
--
-- Run this is the database to be analysed
--
select object_name(s.object_id) as TableName, isnull(i.name,'HEAP') as IndexName,
case i.index_id
```

```
when 0 then 'HEAP'
when 1 then 'CLUS'
else 'NC'
end as IndexType
, reads=range_scan_count + singleton_lookup_count
, 'leaf_writes'=leaf_insert_count+leaf_update_count+ leaf_delete_count
, 'leaf_page_splits' = leaf_allocation_count
, 'nonleaf_writes'=nonleaf_insert_count + nonleaf_update_count +
nonleaf_delete_count
, 'nonleaf_page_splits' = nonleaf_allocation_count
from sys.dm_db_index_operational_stats (db_id(),NULL,NULL,NULL) s join
sys.indexes i
on i.object_id = s.object_id and i.index_id = s.index_id
where objectproperty(s.object_id,'IsUserTable') = 1
order by reads desc, leaf_writes, nonleaf_writes;
```

## 8.1 This query only shows those indexes which have reads

```
--
-- Run this is the database to be analysed
--
select object_name(s.object_id) as TableName, isnull(i.name,'HEAP') as IndexName,
case i.index_id
when 0 then 'HEAP'
when 1 then 'CLUS'
else 'NC'
end as IndexType
, reads=range_scan_count + singleton_lookup_count
, 'leaf_writes'=leaf_insert_count+leaf_update_count+ leaf_delete_count
, 'leaf_page_splits' = leaf_allocation_count
, 'nonleaf_writes'=nonleaf_insert_count + nonleaf_update_count +
nonleaf_delete_count
, 'nonleaf_page_splits' = nonleaf_allocation_count
from sys.dm_db_index_operational_stats (db_id(),NULL,NULL,NULL) s join
sys.indexes i
on i.object_id = s.object_id and i.index_id = s.index_id
where objectproperty(s.object_id,'IsUserTable') = 1
and (range_scan_count + singleton_lookup_count)>0
order by reads desc, leaf_writes, nonleaf_writes;
```

## 8.2 Indexes without reads but with high writes may be being maintained unnecessarily

```
--
-- Run this is the database to be analysed
--
select object_name(s.object_id) as TableName, isnull(i.name,'HEAP') as IndexName,
case i.index_id
when 0 then 'HEAP'
when 1 then 'CLUS'
else 'NC'
end as IndexType
, reads=range_scan_count + singleton_lookup_count
, 'leaf_writes'=leaf_insert_count+leaf_update_count+ leaf_delete_count
, 'leaf_page_splits' = leaf_allocation_count
, 'nonleaf_writes'=nonleaf_insert_count + nonleaf_update_count +
nonleaf_delete_count
, 'nonleaf_page_splits' = nonleaf_allocation_count
from sys.dm_db_index_operational_stats (db_id(),NULL,NULL,NULL) s join
sys.indexes i
on i.object_id = s.object_id and i.index_id = s.index_id
where objectproperty(s.object_id,'IsUserTable') = 1
and (range_scan_count + singleton_lookup_count)=0
and (leaf_update_count+ leaf_delete_count+ nonleaf_update_count +
nonleaf_delete_count)>0
order by leaf_writes desc, nonleaf_writes desc;
```

## 9.0 Order by writes to find your most heavily updated index

- The inserts have been separated as we cannot avoid this io
- Indexes without writes are excluded
- Secondary indexes appearing high on the list may benefit from being placed on a separate filegroup
- High levels of leaf writes on a HEAP may indicate the possibility of fragmentation which cannot be removed

```
--
-- Run this is the database to be analysed
--
select object_name(s.object_id) as TableName, isnull(i.name,'HEAP') as IndexName,
case i.index_id
when 0 then 'HEAP'
when 1 then 'CLUS'
else 'NC'
end as IndexType
,'total_writes'=leaf_update_count+ leaf_delete_count+nonleaf_update_count +
nonleaf_delete_count
,'total_insert_writes'=leaf_insert_count+nonleaf_insert_count
, 'leaf_writes'=leaf_update_count+ leaf_delete_count
, 'nonleaf_writes'=nonleaf_update_count + nonleaf_delete_count
, 'insert_leaf_writes'=leaf_insert_count
, 'insert_nonleaf_writes'=nonleaf_insert_count
, 'leaf_page_splits' = leaf_allocation_count
, 'nonleaf_page_splits' = nonleaf_allocation_count
, reads=range_scan_count + singleton_lookup_count
from sys.dm_db_index_operational_stats (db_id(),NULL,NULL,NULL) s join
sys.indexes i
on i.object_id = s.object_id and i.index_id = s.index_id
where objectproperty(s.object_id,'IsUserTable') = 1
and (leaf_update_count+ leaf_delete_count+nonleaf_update_count +
nonleaf_delete_count+leaf_insert_count+nonleaf_insert_count)>0
order by total_writes desc,total_insert_writes desc;
```

## 10.0 Simple query for user access reads and writes

```
--
-- Run this is the database to be analysed
--
select object_name(s.object_id) as TableName, isnull(i.name,'HEAP') as IndexName,
case i.index_id
when 0 then 'HEAP'
when 1 then 'CLUS'
else 'NC'
end as IndexType
,reads=user_seeks + user_scans + user_lookups
,writes =  user_updates
```

```
from sys.dm_db_index_usage_stats s join sys.indexes i
on s.object_id = i.object_id and i.index_id = s.index_id
where objectproperty(s.object_id,'IsUserTable') = 1
and s.database_id = db_id()
order by reads desc;
go
```

In much the same way as **::fn_virtualfilestats** or **sys_dm_io_virtual_file_stats** return information concerning file activity **sys.dm_db_index_operational_stats** allows us to view similar information for our indexes
http://msdn2.microsoft.com/en-us/library/ms190326.aspx
http://msdn2.microsoft.com/en-us/library/ms187309.aspx

## 11.0 Query to display waits and blocks

- this query is cumulative so is of questionable value

```
--
-- Run this is the database to be analysed
--
Select object_name(s.object_id) as TableName
,isnull(i.name,'HEAP') as IndexName
,case i.index_id
when 0 then 'HEAP'
when 1 then 'CLUS'
else 'NC'
end as IndexType
,row_lock_count, row_lock_wait_count
,[block %]=cast (100.0 * row_lock_wait_count / (1 + row_lock_count) as
numeric(15,2))
, row_lock_wait_in_ms
, [avg row lock waits in ms]=cast (1.0 * row_lock_wait_in_ms / (1 +
row_lock_wait_count) as numeric(15,2))
from sys.dm_db_index_operational_stats (db_id(), NULL, NULL, NULL) s join
sys.indexes i
on i.object_id = s.object_id and i.index_id = s.index_id
where objectproperty(s.object_id,'IsUserTable') = 1
order by row_lock_wait_count desc
```

## 11.1 Typical sample output

| Table | Index | Index Type | Row lock count | Row lock wait count | block % | Row lock wait in ms | avg row lock waits in ms |
|---|---|---|---|---|---|---|---|
| Sales | $21 | NC | 32679 | 34 | 0.10 | 104076 | 2973.60 |
| Employees | PK_Employees | CLUS | 958510 | 32 | 0.00 | 1055219 | 31976.33 |
| Invoices | PK_Invoices | CLUS | 100332 | 27 | 0.03 | 19049 | 680.32 |
| Returns | Clx_Returns_Date | CLUS | 1771122 | 21 | 0.00 | 168156 | 7643.45 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Areas | PK_Areas | CLUS | 50762 | 12 | 0.02 | 937 | 72.08 |

- In general terms as this is from start up the values are likely diluted, however we can see that on the Employees table ( none of these tables are real, although the data is ) we have some serious waits when a wait does occur.
- You would want to snapshot and compare to extract operational data values.

## 12.0 Find your worst scans

- An index scan may not be an issue so we'll concentrate on table scans
- A clustered index scan is a table scan so we'll select on index id 0 and 1

```
--
-- run in database to be analysed
--
select object_name(s.object_id) as TableName,isnull(i.name,'HEAP') as IndexName,
case i.index_id
when 0 then 'HEAP'
when 1 then 'CLUS'
else 'NC'
end as IndexType,
user_seeks as Seeks, user_scans as Scans, user_lookups as Lookups
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
where database_id = db_id() and objectproperty(s.object_id,'IsUserTable') = 1
and user_scans>0 and i.index_id<2
order by user_scans desc;
```

## 12.1 Add a calculation and sort by proportion of scans to seeks

```
--
-- run in database to be analysed
--
select object_name(s.object_id) as TableName,isnull(i.name,'HEAP') as IndexName,
case i.index_id
when 0 then 'HEAP'
when 1 then 'CLUS'
else 'NC'
end as IndexType,
user_seeks as Seeks, user_scans as Scans, user_lookups as Lookups
,cast((user_scans*1.0/(user_seeks+user_scans))*100 as numeric(5,2)) as '%age'
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
where database_id = db_id() and objectproperty(s.object_id,'IsUserTable') = 1
and user_scans>0 and i.index_id<2
order by '%age' desc;
```

## 13.0 Working with snapshots

In section 6.0 ( above ) we devised a simple query to collect index data
This method allows us to analyse usage far better.

Each index will have a row in the table if it were not used in the previous 24 hours
The snapshot taken just after midnight
After collecting data for, say, 32 days if an index has 32 entries in the table then it was not used in this time at all

## 13.1 This query shows counts for index entries

- It assumes that the table **Unused_Indexes** was created in the user database

```
--
-- run in database to be analysed
--
select object_name(ui.[object_id]),si.name,ui.index_id,count(*) as Days
from dbo.Unused_Indexes ui join sys.indexes si
on ui.[object_id] = si.[object_id] and ui.index_id = si.index_id
where objectproperty(ui.[object_id],'IsUserTable') = 1
group by ui.[object_id],ui.index_id,si.name
order by count(*) desc,ui.[object_id],ui.index_id asc;
```

## 13.2 How large are indexes in my database

- Great care must be exercised when running a query which includes **sys.dm_db_index_physical_stats** as for large tables this may well run for some time.
- An alternative would be to gather the information as a separate process .. see **13.4**

```
--
-- use with great care !!!
--
SELECT object_name(a.[object_id]) as TableName,a.index_id, isnull(b.name,'HEAP')
as IndexName, sum(a.page_count) as pages,sum(a.page_count)*1.0/1024 as Mb
FROM sys.dm_db_index_physical_stats (DB_ID(), NULL,NULL, NULL, 'DETAILED') AS
a
   JOIN sys.indexes AS b ON a.object_id = b.object_id AND a.index_id = b.index_id
group by a.[object_id],a.index_id, b.name
order by pages desc;
GO
```

| **TableName** | **index_id** | **IndexName** | **pages** | **Mb** |
|---|---|---|---|---|
| Employees | 1 | PK_Employees | 86897 | 84.860351 |
| Sales | 1 | PK_Sales | 72222 | 70.529296 |
| TableInfo | 0 | HEAP | 13909 | 13.583007 |
| Areas | 1 | PK_Areas | 5753 | 5.618164 |

## 13.3 How large are my unused indexes

- Great care must be exercised when running a query which includes **sys.dm_db_index_physical_stats** as for large tables this may well run for some time.

- An alternative would be to gather the information as a separate process .. see 13.4
- This expands upon the query in 13.1
- One of the joys of SQL 2005 is the ability to join using a table valued function, sadly some of microsoft's functions don't work, **sys.dm_db_index_physical_stats** for example, however this is not a problem as all we need to do is create our own table valued function then we can happily join **sys.dm_db_index_physical_stats** to any table of our choice.
- **IMPORTANT** queries using this may be long running, especially on a large database – be very careful.

## 13.4 Table Valued Function for sys.dm_db_index_physical_stats

- I personally would place this in the master database and might be tempted to make this part of the sys schema, for the examples that follow I've placed the function in master.

```
Create function dbo.fn_db_index_physical_stats
--
-- sys.dm_db_index_physical_stats is actually a function not a view
-- sadly you can't use CROSS APPLY with it, however, drop it in a table
-- valued function and you can!
--
(
@db_id int
,@object_id int
,@index_id int
,@partition_number int
,@mode nvarchar(16)
)
Returns @table TABLE
(
[database_id] [smallint] NULL,
[object_id] [int] NULL,
[index_id] [int] NULL,
[partition_number] [int] NULL,
[index_type_desc] [nvarchar](60) NULL,
[alloc_unit_type_desc] [nvarchar](60) NULL,
[index_depth] [tinyint] NULL,
[index_level] [tinyint] NULL,
[avg_fragmentation_in_percent] [float] NULL,
[fragment_count] [bigint] NULL,
[avg_fragment_size_in_pages] [float] NULL,
[page_count] [bigint] NULL,
[avg_page_space_used_in_percent] [float] NULL,
[record_count] [bigint] NULL,
[ghost_record_count] [bigint] NULL,
[version_ghost_record_count] [bigint] NULL,
[min_record_size_in_bytes] [int] NULL,
[max_record_size_in_bytes] [int] NULL,
[avg_record_size_in_bytes] [float] NULL,
[forwarded_record_count] [bigint] NULL
)
BEGIN
        insert into @table
        select *
        from sys.dm_db_index_physical_stats (@db_id, @object_id,
@index_id ,@partition_number,@mode )
```

```
        return
END;
--end function
go
```

## 13.5 Sample Query to show size of 25 unused indexes

- This query is selecting indexes which have 30 entries in the table **Unused_Indexes**

```
declare @table table(object_id int,ind_name sysname,index_id int)
insert into @table
select top 25 ui.[object_id],si.name,ui.index_id
from dbo.Unused_Indexes ui join sys.indexes si
on ui.[object_id] = si.[object_id] and ui.index_id = si.index_id
where objectproperty(ui.[object_id],'IsUserTable') = 1
group by ui.[object_id],ui.index_id,si.name
having count(*) >30
order by count(*) desc,ui.[object_id],ui.index_id asc
--
select  object_name(ui.[object_id]) as TableName,ui.ind_name as IndexName,
max(fps.index_level)as IndexDepth,sum(fps.page_count) as
TotalPages,max(fps.record_count) as LeafRows, sum(fps.page_count)/1024 as Mb
from @table ui
CROSS APPLY master.dbo.fn_db_index_physical_stats
(DB_ID(),ui.[object_id],ui.index_id,NULL, 'Detailed') AS fps
group by ui.[object_id],ui.ind_name
order by sum(fps.page_count) desc ;
```

Abridged results

| **TableName** | **IndexName** | **IndexDepth** | **TotalPages** | **LeafRows** | **Mb** |
|---|---|---|---|---|---|
| Employees | Idx_Employees_123 | 4 | 54811 | 11159545 | 53.526367 |
| Invoices | Idx_Invoices_paper | 4 | 44532 | 5162428 | 43.488281 |
| Invoices | Idx_Invoices_ink | 4 | 39539 | 5162428 | 38.612304 |
| DepartmentSales | Idx_DS_Promotions | 3 | 16791 | 3199181 | 16.397460 |
| Sales2006 | Idx_sales2007_opened | 2 | 116 | 31291 | 0.113281 |
| Sales2006 | Idx_sales2007_area | 2 | 103 | 31291 | 0.100585 |
| Sales2006 | Idx_sales2007_size | 2 | 87 | 31291 | 0.084960 |

- The value for index depth designates the depth of the b-tree, a depth of 4 means that there will be 4 io to return a value from that index.
- Total pages is the sum of used pages reported fro all levels of the index, not just the leaf level
- Leaf Rows should indicate the number of rows in the index, normally the number of rows in the table.
- Mb is the calculation of the physical storage space the index is using