

# Indexes

Last updated by | Vitor Tomaz | Feb 24, 2023 at 3:27 AM PST

---

## Contents


- [Issue](#)
- [Investigation / Analysis](#)
- [Mitigation](#)
  - [Find Index Fragmentation Levels](#)
  - [Find Missing indexes](#)
    - [Option 1 - Missing index DMVs](#)
    - [Option 2 - Query Store](#)
  - [Check the last time when indexes were rebuilt](#)
  - [Rebuild ALL Indexes](#)
  - [Use the ONLINE and RESUMABLE index options](#)
- [Public Doc Reference](#)
- [Internal Doc Reference](#)



## Indexes

### Issue

This is a "How To" TSG about maintaining and rebuilding indexes.

### Investigation / Analysis

The topic of indexes and their maintenance is discussed in depth in the article [Optimize index maintenance to improve query performance and reduce resource consumption](#) . It contains everything you and your customer need to know about indexes, and helps you decide when and how to perform index maintenance. It covers concepts such as index fragmentation and page density and their impact on query performance and resource consumption. It describes index maintenance methods like reorganizing and rebuilding an index and suggests an index maintenance strategy that balances potential performance improvements against resource consumption required for maintenance. It also discusses maintaining rowstore vs. columnstore indexes along with using an index rebuild to recover from data corruption.

For a complete, customer-ready index and statistics maintenance Stored Procedure, see Yochanan's blog article [How to maintain Azure SQL Indexes and Statistics](#)  and also consider the [Adaptive Index Defrag](#)  toolbox.

The information in the "Mitigation" section below provides you with sample SQL scripts to perform some manual analysis and maintenance steps.

### Mitigation

Make sure that you have read and understood the topics in:

[Optimize index maintenance to improve query performance and reduce resource consumption](#) 

before applying the steps in the sections below.

## Find Index Fragmentation Levels

-- Run against your database to check the current index fragmentation levels

```
SELECT
    idxs.[object_id]
    ,ObjectSchema = OBJECT_SCHEMA_NAME(idxs.object_id)
    ,ObjectName = OBJECT_NAME(idxs.object_id)
    ,IndexName = idxs.name
    ,idxs.type
    ,idxs.type_desc
    ,i.avg_fragmentation_in_percent
    ,i.page_count
    ,i.index_id
    ,i.partition_number
    ,i.avg_page_space_used_in_percent
    ,i.record_count
    ,i.ghost_record_count
    ,i.forwarded_record_count

FROM sys.indexes idxs
INNER JOIN sys.dm_db_index_physical_stats(DB_ID(),NULL, NULL, NULL, 'SAMPLED') i ON i.object_id = idxs.object_id

WHERE idxs.type IN (0 /*HEAP*/, 1/*CLUSTERED*/, 2/*NONCLUSTERED*/, 5/*CLUSTERED COLUMNSTORE*/, 6/*NONCLUSTERED
AND (alloc_unit_type_desc = 'IN_ROW_DATA' /*avoid LOB_DATA or ROW_OVERFLOW_DATA*/ OR alloc_unit_type_desc IS N
AND OBJECT_SCHEMA_NAME(idxs.object_id) != 'sys'
AND idxs.is_disabled = 0

ORDER BY i.avg_fragmentation_in_percent DESC, i.page_count DESC
```





Sample output:

Results Messages								
	object_id	ObjectSchema	ObjectName	IndexName	type	type_desc	avg_fragmentation_in_percent	page_count
1	274100017	Person	Person	IX_Person_LastName_FirstName_MiddleN...	2	NONCLUSTERED	55.6701030927835	194
2	178099675	Person	CountryRegion	PK_CountryRegion_CountryRegionCode	1	CLUSTERED	50	2
3	178099675	Person	CountryRegion	AK_CountryRegion_Name	2	NONCLUSTERED	50	2
4	402100473	Person	StateProvince	AK_StateProvince_Name	2	NONCLUSTERED	50	2
5	1026102696	Production	ProductReview	PK_ProductReview_ProductReviewID	1	CLUSTERED	50	2
6	1143675122	Person	vStateProvinceCo...	IX_vStateProvinceCountryRegion	1	CLUSTERED	50	2
7	1282103608	Purchasing	ProductVendor	IX_ProductVendor_BusinessEntityID	2	NONCLUSTERED	50	2

index_id	partition_number	avg_page_space_used_in_percent	record_count	ghost_record_count	forwarded_record_count
3	1	71.6961082283173	19972	0	NULL
1	1	89.8443291326909	238	0	NULL
2	1	75.1420805534964	238	0	NULL
2	1	53.4902396837163	181	0	NULL
1	1	63.8127007659995	4	0	NULL
1	1	95.9970348406227	181	0	NULL
2	1	82.3820113664443	460	0	NULL

## Find Missing indexes

The following example returns missing index suggestions for the current database. Missing index suggestions should be combined when possible with one another, and with existing indexes in the current database. Learn how to apply these suggestions in [Tune nonclustered indexes with missing index suggestions](#) .

In addition to the SQL query below, you can also see missing index suggestions in execution plans. See [View missing index suggestions in execution plans](#)  for step-by-step description.

### Option 1 - Missing index DMVs

```
-- generate commands to create missing indexes
SELECT
    CONVERT (varchar(30), getdate(), 126) AS runtime,
    CONVERT (decimal (28, 1), migs.avg_total_user_cost * migs.avg_user_impact * (migs.user_seeks + migs.user_s
    'CREATE INDEX [missing_index_' + CONVERT (varchar, mig.index_group_handle) + '_' + CONVERT (varchar, mid.i
    + ' 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,
    mig.index_group_handle,
    mid.index_handle,
    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 CONVERT (decimal (28, 1), migs.avg_total_user_cost * migs.avg_user_impact * (migs.user_seeks + migs.user
ORDER BY migs.avg_total_user_cost * migs.avg_user_impact * (migs.user_seeks + migs.user_scans) DESC
```

## Option 2 - Query Store







The following query retrieves the top 20 query plans containing missing index requests from query store based on a rough estimate of total logical reads for the query. The data is limited to query executions within the past 48 hours - adapt the Where clause as needed.

```
-- this index is slow due to its XML operations
SELECT TOP 20
    qsq.query_id,
    SUM(qrs.count_executions) * AVG(qrs.avg_logical_io_reads) AS est_logical_reads,
    SUM(qrs.count_executions) AS sum_executions,
    AVG(qrs.avg_logical_io_reads) AS avg_avg_logical_io_reads,
    SUM(qsq.count_compiles) AS sum_compiles,
    (SELECT TOP 1 qsqt.query_sql_text FROM sys.query_store_query_text qsqt
     WHERE qsqt.query_text_id = MAX(qsq.query_text_id)) AS query_text,
    TRY_CONVERT(XML, (SELECT TOP 1 qsp2.query_plan FROM sys.query_store_plan qsp2
     WHERE qsp2.query_id=qsq.query_id
     ORDER BY qsp2.plan_id DESC)) AS query_plan
FROM sys.query_store_query qsq
JOIN sys.query_store_plan qsp ON qsq.query_id=qsp.query_id
CROSS APPLY (SELECT TRY_CONVERT(XML, qsp.query_plan) AS query_plan_xml) AS qpx
JOIN sys.query_store_runtime_stats qrs ON
    qsp.plan_id = qrs.plan_id
JOIN sys.query_store_runtime_stats_interval qrsi ON
    qrs.runtime_stats_interval_id=qrsi.runtime_stats_interval_id
WHERE
    qsp.query_plan LIKE N'%<MissingIndexes>%'
    and qrsi.start_time >= DATEADD(HH, -48, SYSDATETIME())
GROUP BY qsq.query_id, qsq.query_hash
ORDER BY est_logical_reads DESC;
```

Sample output - you will see the missing index details by clicking on the execution plan link:

Results		Messages					
query_id	est_logical_reads	sum_executions	avg_avg_logical_io_reads	sum_compiles	query_text	query_plan	
1	6008	3043	1	3043	3	update person.person set FirstName = 'Arnold' where FirstName = 'Kevin'	<ShowPlanXML xmlns="http://schemas.microsoft.com...
2	6020	5	1	5	4	select LastName from person.person with (updlock) where FirstName = 'Kevin'	<ShowPlanXML xmlns="http://schemas.microsoft.com...

ExecutionPlan1.sqlplan*	SQLQuery3.sql - tcp...orks (holgerl (78))*	SQLQuery2.sql - tcp...orks (holgerl (70))*	SQLQuery1.sql - tcp...orks (holgerl (83))*
Query 1: Query cost (relative to the batch): 100%			
UPDATE [person].[person] set [FirstName] = @1 WHERE [FirstName]=@2			
Missing Index (Impact 52.6722): CREATE NONCLUSTERED INDEX [<Name of Missing Index, sysname,>] ON [Person].[Person] ([FirstName])			

T-SQL						
UPDATE	Clustered Index Update	Compute Scalar	Compute Scalar	Compute Scalar	Table Spool (Eager Spool)	Index Scan (NonClust [Person].[IX_Person_LastN
Cost: 0 %	Cost: 41 %	Cost: 0 %	Cost: 0 %	Cost: 0 %	Cost: 9 %	Cost: 50 %

## Check the last time when indexes were rebuilt

```

SELECT
    OBJECT_NAME(object_id) AS [TableName],
    name AS [IndexName],
    STATS_DATE(object_id, stats_id) AS [LastStatsUpdate]
FROM sys.stats
WHERE
    name NOT LIKE '_WA%'
    AND STATS_DATE(object_id, stats_id) IS NOT NULL
    AND OBJECTPROPERTY(object_id, 'IsMSShipped') = 0
ORDER BY [LastStatsUpdate] DESC
--ORDER BY TableName, IndexName

```

Sample output:

Results		Messages	
	TableName	IndexName	LastStatsUpdate
1	SpecialOffer	PK_SpecialOffer_SpecialOfferID	2022-09-15 10:12:37.390
2	SalesTerritoryHistory	PK_SalesTerritoryHistory_BusinessEntityID_StartDate_Te...	2022-09-15 10:12:36.823
3	SalesPersonQuotaHistory	PK_SalesPersonQuotaHistory_BusinessEntityID_QuotaD...	2022-09-15 10:12:36.253
4	SalesReason	PK_SalesReason_SalesReasonID	2022-09-15 10:12:35.663
5	SalesOrderHeaderSalesReason	PK_SalesOrderHeaderSalesReason_SalesOrderID_Sales...	2022-09-15 10:12:35.560
6	SalesPerson	PK_SalesPerson_BusinessEntityID	2022-09-15 10:12:35.277
7	Customer	PK_Customer_CustomerID	2022-09-15 10:12:35.197

## Rebuild ALL Indexes

Use the stored procedure from Yochanan's blog article [How to maintain Azure SQL Indexes and Statistics](#) for rebuilding all indexes. The proc has build-in logic to perform online rebuilds if the index supports it, thus has less impact on the concurrent workload.

If you don't care about blocking other workloads and simply want to rebuild everything, then run the following SQL script:

```

-- This will rebuild all the indexes on all the tables in your database.
-- remove the comments from EXEC sp_executesql in order to have the commands actually rebuild the indexes, ins

SET NOCOUNT ON

DECLARE rebuildindexes CURSOR FOR
    SELECT table_schema, table_name FROM information_schema.tables where TABLE_TYPE = 'BASE TABLE'

OPEN rebuildindexes

DECLARE @tableSchema NVARCHAR(128)
DECLARE @tableName NVARCHAR(128)
DECLARE @Statement NVARCHAR(300)

FETCH NEXT FROM rebuildindexes INTO @tableSchema, @tableName

WHILE (@@FETCH_STATUS = 0)
BEGIN
    SET @Statement = 'ALTER INDEX ALL ON ' + '[' + @tableSchema + ']' + '.' + '[' + @tableName + ']' + ' REBUI
    PRINT @Statement -- comment this print statement to prevent it from printing whenever you are ready to exec
    --EXEC sp_executesql @Statement -- remove the comment on the beginning of this line to run the commands
    FETCH NEXT FROM rebuildindexes INTO @tableSchema, @tableName
END

CLOSE rebuildindexes
DEALLOCATE rebuildindexes
SET NOCOUNT OFF

```

But keep in mind that rebuilding indexes is not the only option, and is not always the best option. See [Optimize index maintenance to improve query performance and reduce resource consumption](#) to learn why.

## Use the ONLINE and RESUMABLE index options

The ONLINE rebuild option has the advantage that no long-term table locks are held during the index operation. During the main phase of the index operation, only an Intent Share (IS) lock is held on the source table. This allows queries or updates to the underlying table and indexes to continue. At the start of the operation, a Shared (S) lock is very briefly held on the source object. At the end of the operation, an S lock is very briefly held on the source if a nonclustered index is being created. A Schema Modification (Sch-M) lock is acquired when a clustered index is created or dropped online and when a clustered or nonclustered index is being rebuilt.

See [Guidelines for online index operations](#) and [Perform Index Operations Online](#) for further details and examples.

```

-- Execute an online index rebuild as resumable operation with MAXDOP=1
ALTER INDEX AK_Person_rowguid ON Person.Person REBUILD WITH (ONLINE=ON, MAXDOP=1, RESUMABLE=ON);

-- To pause immediately the index operation, do either of the following:
<cancel the rebuild statement on the client, e.g. Ctrl-C or Cancel button in SSMS>
ALTER INDEX AK_Person_rowguid ON Person.Person PAUSE;
KILL <session_id>;

-- Check for resumable index operations
select * from sys.index_resumable_operations;








-- Resume an online index rebuild operation for an index rebuild that was executed as resumable. Set MAXDOP
ALTER INDEX AK_Person_rowguid ON Person.Person RESUME WITH (MAXDOP=4);

```

Sample output for sys.index\_resumable\_operations:

Results		Messages											
	object_id	index_id	name	sql_text	last_max_dop_used	partition_number	state	state_desc	start_time	last_pause_time	total_execution_time	percent_complete	page_count
1	274100017	2	AK_Person_rowguid	ALTER INDEX AK_Person_rowguid ON Person.Person R...	1	NULL	1	PAUSED	2022-10-06 14:55:07.123	2022-10-06 14:55:07.133	0	5.00700981373924	8

## Public Doc Reference

- [ALTER INDEX \(Transact-SQL\)](#) 
- [Index maintenance methods: reorganize and rebuild](#) 
- [REORGANIZE a rowstore index](#) 
- [REORGANIZE a columnstore index](#) 
- [RESUMABLE index operations](#) 
- [ONLINE index operations](#) 
- [Guidelines for online index operations](#) 

## Internal Doc Reference

- [Statistics](#)

How good have you found this content?

