Yes, **you can defragment indexes** on **SQL Server publisher replicated tables,** but there are a few important things

to keep in mind depending on the type of replication and the method you use.

**✅ General Rules:**

* **Index maintenance** (rebuild/reorganize) on publisher tables is allowed and often necessary for performance.
* It **does not affect replication** directly (replication doesn’t track index changes).
* **Transactional replication** and **merge replication** behave slightly differently in terms of how schema changes are handled—but index defragmentation isn’t considered a schema change.

**✅ Safe Ways to Defragment:**

You can use either:

* ALTER INDEX ... REBUILD
* ALTER INDEX ... REORGANIZE

These operations:

* **Do not replicate** to subscribers (index changes are local to the publisher).
* **Do not break replication**.

**⚠️ Caution:**

If you're doing **online index rebuilds**, check whether:

* Your SQL Server version supports **online operations** for the edition you use (e.g., not all online options are supported in **Standard Edition**).
* You’re not using **replicated indexed views**—changing indexed views can impact replication.

**💡 Best Practices:**

* **Schedule during low activity** windows to avoid replication latency.
* Consider **updating statistics** afterward, especially if index maintenance changes data distribution.
* Monitor replication performance after index maintenance just to be safe.

**🔧 1. Assess the Table Size and Index Fragmentation**

Start by checking fragmentation:

SELECT

OBJECT\_NAME(ps.object\_id) AS TableName,

i.name AS IndexName,

ps.index\_id,

ps.avg\_fragmentation\_in\_percent,

ps.page\_count

FROM sys.dm\_db\_index\_physical\_stats (DB\_ID(), NULL, NULL, NULL, 'LIMITED') AS ps

JOIN sys.indexes AS i ON ps.object\_id = i.object\_id AND ps.index\_id = i.index\_id

WHERE ps.database\_id = DB\_ID()

AND ps.page\_count > 1000 -- Only large indexes

AND ps.avg\_fragmentation\_in\_percent > 10;

**🔁 2. Decide: REBUILD vs REORGANIZE**

| **Option** | **When to Use** | **Notes** |
| --- | --- | --- |
| ALTER INDEX ... REBUILD | Fragmentation > 30% | More intensive, may cause blocking, but faster for heavily fragmented indexes |
| ALTER INDEX ... REORGANIZE | Fragmentation 10–30% | Lighter, online by default, good for minimizing impact |

🔐 3. **Use ONLINE = ON for Large Tables**

ALTER INDEX [IndexName] ON [dbo].[YourTable]

REBUILD WITH (ONLINE = ON, MAXDOP = 4, SORT\_IN\_TEMPDB = ON);

**Options explained:**

* ONLINE = ON: Minimizes blocking. *(Only in Enterprise/Developer editions)*
* MAXDOP: Controls CPU usage—tune this to avoid overload.
* SORT\_IN\_TEMPDB = ON: Reduces contention on user database files.

**📅 5. Schedule Maintenance Windows**

* **Avoid peak hours.** Index rebuilds can interfere with replication agent jobs.
* Use SQL Server Agent jobs to schedule during off-hours.

**🔍 6. Monitor Impact on Replication**

Index operations don’t replicate, but if you lock or block DML operations, it could:

* Delay log reader agent
* Increase replication latency

Monitor:

-- Check latency

exec sp\_replmonitorsubscriptionpendingcmds

-- Or use Replication Monitor for a live view

**✅ After Rebuilds**

Always update stats

SQL Server automatically updates the statistics after the index rebuild. It is equivalent to update statistics with FULL SCAN however; it does not update the column statistics. We should update column statistics after index rebuild as well.

Update STATISTICS <schemaName.tableName> WITH FULLSCAN, COLUMNS

SQL Server does not update statistics with the index reorganize activity. We should manually update the statistics, if required or need to rely on the automatically updated statistics.

**Summary**

| **Step** | **Description** |
| --- | --- |
| 1 | Analyze fragmentation |
| 2 | Choose REBUILD or REORGANIZE |
| 3 | Use ONLINE=ON for minimal blocking |
| 4 | Rebuild via dynamic T-SQL or maintenance plans |
| 5 | Schedule and monitor replication latency |
| 6 | Consider partitioning for very large tables |
| 7 | Update statistics |

**🧩 1. Benefits of Partitioning for Index Maintenance**

* ✅ **Maintain one partition at a time** (rather than the whole table).
* ✅ **Minimize blocking** and transaction log usage.
* ✅ Reduces **replication impact** by isolating DML/DDL.
* ✅ Can **automate sliding window** partition strategies for archiving/historical data.

**🔧 3. Rebuild Indexes by Partition**

Instead of rebuilding the entire index:

-- Rebuild only partition 3 (e.g., Jan–Jul 2024)

ALTER INDEX IX\_LargeFactTable\_CreatedDate

ON dbo.LargeFactTable

REBUILD PARTITION = 3

WITH (ONLINE = ON, SORT\_IN\_TEMPDB = ON, MAXDOP = 4);

**🧠 4. Find Fragmented Partitions Dynamically**

Here's a script to find which **partitions are fragmented**:

SELECT

OBJECT\_NAME(ps.object\_id) AS TableName,

i.name AS IndexName,

ps.index\_id,

ps.partition\_number,

ps.avg\_fragmentation\_in\_percent,

ps.page\_count

FROM sys.dm\_db\_index\_physical\_stats(DB\_ID(), NULL, NULL, NULL, 'LIMITED') ps

JOIN sys.indexes i ON ps.object\_id = i.object\_id AND ps.index\_id = i.index\_id

WHERE ps.page\_count > 10000 -- Large partitions only

AND ps.avg\_fragmentation\_in\_percent > 10

ORDER BY ps.avg\_fragmentation\_in\_percent DESC;

**🔁 5. Automated Rebuild Script per Fragmented Partition**

Generate dynamic SQL for rebuilding **only the fragmented partitions**:

DECLARE @SQL NVARCHAR(MAX) = ''

SELECT @SQL +=

'ALTER INDEX [' + i.name + '] ON [' + s.name + '].[' + o.name + '] ' +

'REBUILD PARTITION = ' + CAST(ps.partition\_number AS VARCHAR) +

' WITH (ONLINE = ON, SORT\_IN\_TEMPDB = ON, MAXDOP = 4);' + CHAR(13)

FROM sys.dm\_db\_index\_physical\_stats(DB\_ID(), NULL, NULL, NULL, 'LIMITED') ps

JOIN sys.indexes i ON ps.object\_id = i.object\_id AND ps.index\_id = i.index\_id

JOIN sys.objects o ON ps.object\_id = o.object\_id

JOIN sys.schemas s ON o.schema\_id = s.schema\_id

WHERE ps.page\_count > 10000

AND ps.avg\_fragmentation\_in\_percent > 10;

EXEC sp\_executesql @SQL;

**📈 6. Impact on Replication**

* These index rebuilds do **not replicate**.
* Since partition-level rebuilds are more lightweight, **they reduce blocking**, which means **lower replication latency**.
* Still monitor the **Log Reader Agent** during maintenance windows.

💡 7. **Advanced: Sliding Window Pattern**

If you're using **partition switching** to manage data (e.g., archiving old data), this plays perfectly with index maintenance:

* Switch out old partition to staging table.
* Rebuild index offline on the staging table (if needed).
* Archive or load somewhere else.

Example:

-- Switch out partition 1

ALTER TABLE dbo.LargeFactTable

SWITCH PARTITION 1 TO dbo.StagingTable;

**✅ Best Practices Summary**

| **Action** | **Benefit** |
| --- | --- |
| Rebuild per partition | Targeted, fast, reduces blocking |
| Use ONLINE = ON | Minimizes impact on concurrent sessions |
| Use SORT\_IN\_TEMPDB | Reduces user DB I/O pressure |
| Monitor replication | Ensure agents aren’t delayed |
| Automate per-fragmentation | Smart maintenance, avoids unnecessary rebuilds |

Awesome—let’s build a **smart, targeted index maintenance solution** for your **partitioned tables (>100 partitions)** in a **transactional replication** setup.

**🧠 Goal**

Create a system that:

1. Detects fragmentation on partitioned indexes.
2. Rebuilds **only** the **fragmented partitions**.
3. Works in a **replication-safe** and **low-impact** way.
4. Can be scheduled via a **SQL Server Agent job**.

**✅ Step 1: Dynamic T-SQL Script – Fragmented Partition Rebuilder**

Here’s a **rebuild script** that:

* Scans all partitioned indexes.
* Rebuilds only the **partitions** with high fragmentation.
* Keeps things **replication-safe** and efficient.

USE [YourDatabaseName]; -- Replace with your DB name

GO

DECLARE @ThresholdLow INT = 10; -- Reorganize if between 10–30%

DECLARE @ThresholdHigh INT = 30; -- Rebuild if >30%

DECLARE @SQL NVARCHAR(MAX) = '';

-- Cursor to loop through fragmented partitions

DECLARE PartitionCursor CURSOR FOR

SELECT

s.name AS SchemaName,

o.name AS TableName,

i.name AS IndexName,

ps.index\_id,

ps.partition\_number,

ps.avg\_fragmentation\_in\_percent

FROM sys.dm\_db\_index\_physical\_stats(DB\_ID(), NULL, NULL, NULL, 'LIMITED') ps

JOIN sys.indexes i ON ps.object\_id = i.object\_id AND ps.index\_id = i.index\_id

JOIN sys.objects o ON i.object\_id = o.object\_id

JOIN sys.schemas s ON o.schema\_id = s.schema\_id

WHERE ps.page\_count > 10000

AND ps.avg\_fragmentation\_in\_percent > @ThresholdLow

AND i.type\_desc IN ('CLUSTERED', 'NONCLUSTERED')

AND i.data\_space\_id IN (

SELECT data\_space\_id FROM sys.partition\_schemes

);

DECLARE

@SchemaName SYSNAME,

@TableName SYSNAME,

@IndexName SYSNAME,

@PartitionNumber INT,

@Fragmentation FLOAT;

OPEN PartitionCursor;

FETCH NEXT FROM PartitionCursor INTO @SchemaName, @TableName, @IndexName, @index\_id, @PartitionNumber, @Fragmentation;

WHILE @@FETCH\_STATUS = 0

BEGIN

SET @SQL +=

CASE

WHEN @Fragmentation >= @ThresholdHigh THEN

'PRINT ''Rebuilding partition ' + CAST(@PartitionNumber AS VARCHAR) + ' on index [' + @IndexName + '] of table [' + @SchemaName + '].[' + @TableName + ']'';' +

'ALTER INDEX [' + @IndexName + '] ON [' + @SchemaName + '].[' + @TableName + '] ' +

'REBUILD PARTITION = ' + CAST(@PartitionNumber AS VARCHAR) +

' WITH (ONLINE = ON, SORT\_IN\_TEMPDB = ON, MAXDOP = 4);' + CHAR(13)

WHEN @Fragmentation < @ThresholdHigh THEN

'PRINT ''Reorganizing partition ' + CAST(@PartitionNumber AS VARCHAR) + ' on index [' + @IndexName + '] of table [' + @SchemaName + '].[' + @TableName + ']'';' +

'ALTER INDEX [' + @IndexName + '] ON [' + @SchemaName + '].[' + @TableName + '] ' +

'REORGANIZE PARTITION = ' + CAST(@PartitionNumber AS VARCHAR) + ';' + CHAR(13)

END;

FETCH NEXT FROM PartitionCursor INTO @SchemaName, @TableName, @IndexName, @index\_id, @PartitionNumber, @Fragmentation;

END

CLOSE PartitionCursor;

DEALLOCATE PartitionCursor;

-- Execute the final SQL

EXEC sp\_executesql @SQL;  
  
**🕒 Step 2: Create SQL Agent Job**

You can create a SQL Agent job to run this nightly/weekly/etc.

Here’s how:

**🔧 SQL Agent Job Steps:**

1. **Step Name:** Rebuild Fragmented Partitions
2. **Type:** Transact-SQL script (T-SQL)
3. **Database:** Your replicated database (e.g., YourDatabaseName)
4. **Command:** Paste the full script from Step 1

**🗓️ Suggested Schedule:**

* **Frequency:** Weekly or every few days depending on insert/update volume.
* **Time:** Off-hours (e.g., 2 AM – 5 AM).

**✅ Optional Add-ons**

* 📊 **Email notifications** on completion or failure
* 📁 **Logging** to a maintenance log table
* 🔄 **Rotate** through specific partitions (e.g., if using a sliding window)
* 🔒 **Wrap in TRY/CATCH** blocks for fault tolerance
* A version that writes to a log table with timestamps
* Below is the enhanced version of your **partitioned index maintenance script** that:
* ✅ Detects fragmented partitions  
  ✅ Rebuilds/reorganizes accordingly  
  ✅ Writes detailed **logging info** to a table with timestamps, fragmentation, action taken, and status  
  ✅ Is safe for **transactional replication** environments
* **📦 Step 1: Create the Logging Table**
* Run this once in your **replicated database**:

CREATE TABLE dbo.IndexMaintenanceLog (

LogID INT IDENTITY(1,1) PRIMARY KEY,

SchemaName SYSNAME,

TableName SYSNAME,

IndexName SYSNAME,

PartitionNumber INT,

FragmentationPercent FLOAT,

ActionTaken VARCHAR(20), -- REBUILD or REORGANIZE

Status VARCHAR(20), -- SUCCESS or FAILED

ErrorMessage NVARCHAR(1000) NULL,

ExecutedAt DATETIME DEFAULT GETDATE()

);

**🧠 Step 2: Rebuild Script with Logging**

This script now:

* Captures fragmentation per partition
* Executes REBUILD or REORGANIZE
* Logs result with timestamps

USE [YourDatabaseName]; -- Replace with your DB name

GO

DECLARE @ThresholdLow INT = 10; -- Reorganize if between 10–30%

DECLARE @ThresholdHigh INT = 30; -- Rebuild if >30%

DECLARE @SQL NVARCHAR(MAX) = '';

DECLARE @Error NVARCHAR(1000);

-- Cursor to loop through fragmented partitions

DECLARE PartitionCursor CURSOR FOR

SELECT

s.name AS SchemaName,

o.name AS TableName,

i.name AS IndexName,

ps.index\_id,

ps.partition\_number,

ps.avg\_fragmentation\_in\_percent

FROM sys.dm\_db\_index\_physical\_stats(DB\_ID(), NULL, NULL, NULL, 'LIMITED') ps

JOIN sys.indexes i ON ps.object\_id = i.object\_id AND ps.index\_id = i.index\_id

JOIN sys.objects o ON i.object\_id = o.object\_id

JOIN sys.schemas s ON o.schema\_id = s.schema\_id

WHERE ps.page\_count > 10000

AND ps.avg\_fragmentation\_in\_percent > @ThresholdLow

AND i.type\_desc IN ('CLUSTERED', 'NONCLUSTERED')

AND i.data\_space\_id IN (

SELECT data\_space\_id FROM sys.partition\_schemes

);

DECLARE

@SchemaName SYSNAME,

@TableName SYSNAME,

@IndexName SYSNAME,

@PartitionNumber INT,

@Fragmentation FLOAT,

@IndexSQL NVARCHAR(MAX),

@Action VARCHAR(20);

OPEN PartitionCursor;

FETCH NEXT FROM PartitionCursor INTO @SchemaName, @TableName, @IndexName, @index\_id, @PartitionNumber, @Fragmentation;

WHILE @@FETCH\_STATUS = 0

BEGIN

BEGIN TRY

SET @IndexSQL = '';

SET @Error = NULL;

IF @Fragmentation >= @ThresholdHigh

BEGIN

SET @Action = 'REBUILD';

SET @IndexSQL = 'ALTER INDEX [' + @IndexName + '] ON [' + @SchemaName + '].[' + @TableName + '] ' +

'REBUILD PARTITION = ' + CAST(@PartitionNumber AS VARCHAR) +

' WITH (ONLINE = ON, SORT\_IN\_TEMPDB = ON, MAXDOP = 4);';

END

ELSE

BEGIN

SET @Action = 'REORGANIZE';

SET @IndexSQL = 'ALTER INDEX [' + @IndexName + '] ON [' + @SchemaName + '].[' + @TableName + '] ' +

'REORGANIZE PARTITION = ' + CAST(@PartitionNumber AS VARCHAR) + ';';

END

EXEC sp\_executesql @IndexSQL;

-- Log success

INSERT INTO dbo.IndexMaintenanceLog (SchemaName, TableName, IndexName, PartitionNumber, FragmentationPercent, ActionTaken, Status)

VALUES (@SchemaName, @TableName, @IndexName, @PartitionNumber, @Fragmentation, @Action, 'SUCCESS');

END TRY

BEGIN CATCH

SET @Error = ERROR\_MESSAGE();

INSERT INTO dbo.IndexMaintenanceLog (SchemaName, TableName, IndexName, PartitionNumber, FragmentationPercent, ActionTaken, Status, ErrorMessage)

VALUES (@SchemaName, @TableName, @IndexName, @PartitionNumber, @Fragmentation, @Action, 'FAILED', @Error);

END CATCH

FETCH NEXT FROM PartitionCursor INTO @SchemaName, @TableName, @IndexName, @index\_id, @PartitionNumber, @Fragmentation;

END

CLOSE PartitionCursor;

DEALLOCATE PartitionCursor;

**🗓️ Step 3: Schedule via SQL Agent Job**

Use this script as a **step in your SQL Agent job**:

* **Job Name:** Partitioned Index Maintenance
* **Step Type:** Transact-SQL
* **Database:** Your replicated DB
* **Run As:** SQL Agent account (must have appropriate permissions)
* **📊 Sample Log Output**

| **LogID** | **SchemaName** | **TableName** | **IndexName** | **PartitionNumber** | **FragmentationPercent** | **ActionTaken** | **Status** | **ExecutedAt** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | dbo | SalesFact | IX\_Sales\_Date | 102 | 65.3 | REBUILD | SUCCESS | 2025-04-10 02:00:01 |
| 2 | dbo | Orders | IX\_Order\_Date | 99 | 18.7 | REORGANIZE | SUCCESS | 2025-04-10 02:00:03 |
| 3 | dbo | Shipments | IX\_Ship\_Col | 100 |  |  |  |  |

Let’s make it as production-ready as you need.

let’s complete your solution with:

1. ✅ **Email report** summarizing the job execution (Successes + Failures)
2. ✅ **Retention policy** to purge log entries older than 90 days

**📧 Step 1: Email Summary After Job Runs**

We'll create a **second step in the SQL Agent job** to send an email summary using sp\_send\_dbmail.

**✅ Prerequisites**

Make sure:

* **Database Mail** is enabled and configured.
* A working **profile name** (e.g., DBA\_Mail\_Profile) is set up.
* SQL Agent has **DatabaseMailUserRole** access in msdb.
* **📨 Email Summary Script**
* Here’s a job step you can add **after the index maintenance step**:

DECLARE @Body NVARCHAR(MAX);

-- Generate summary

SELECT @Body =

'Partitioned Index Maintenance Report - ' + CONVERT(VARCHAR, GETDATE(), 120) + CHAR(13) + CHAR(13) +

'Total entries: ' + CAST(COUNT(\*) AS VARCHAR) + CHAR(13) +

'Successful: ' + CAST(SUM(CASE WHEN Status = 'SUCCESS' THEN 1 ELSE 0 END) AS VARCHAR) + CHAR(13) +

'Failed: ' + CAST(SUM(CASE WHEN Status = 'FAILED' THEN 1 ELSE 0 END) AS VARCHAR) + CHAR(13) + CHAR(13) +

'Recent Failures:' + CHAR(13) +

ISNULL(STRING\_AGG(CONCAT(SchemaName, '.', TableName, '.', IndexName,

' (P', PartitionNumber, '): ', ErrorMessage), CHAR(13)), 'None')

FROM dbo.IndexMaintenanceLog

WHERE ExecutedAt >= DATEADD(HOUR, -6, GETDATE()); -- adjust window if needed

-- Send the email

EXEC msdb.dbo.sp\_send\_dbmail

@profile\_name = 'DBA\_Mail\_Profile', -- Change to your profile

@recipients = 'dba-team@example.com', -- Change to your recipients

@subject = 'Partitioned Index Maintenance Report',

@body = @Body;

**🧹 Step 2: Retention Policy – Delete Logs Older Than 90 Days**

You can either:

* Add a **third SQL Agent job step**, or
* Add this **at the end of the maintenance script**

Here’s the cleanup logic:

DELETE FROM dbo.IndexMaintenanceLog

WHERE ExecutedAt < DATEADD(DAY, -90, GETDATE());

**🧱 Job Step Layout**

| **Step Order** | **Name** | **Type** | **Notes** |
| --- | --- | --- | --- |
| 1 | Index Partition Maintenance | Transact-SQL | Main logic w/ logging |
| 2 | Email Summary | Transact-SQL | Sends report via DB mail |
| 3 (opt.) | Cleanup Old Log Entries | Transact-SQL | Deletes logs older than 90d |

**📦 Full SQL Agent Job Script**

Paste the entire script into a SQL query window in SSMS and run it to deploy everything at once.

🔁 Before running:  
🔧 Update the following variables at the top:

* 'YourDatabaseName' – name of your replicated DB
* 'DBA\_Mail\_Profile' – your configured Database Mail profile
* 'dba-team@example.com' – your email recipients

USE msdb;

GO

-- ============================================

-- CONFIGURATION

-- ============================================

DECLARE @DBName SYSNAME = 'YourDatabaseName';

DECLARE @MailProfileName SYSNAME = 'DBA\_Mail\_Profile';

DECLARE @EmailRecipients NVARCHAR(500) = 'dba-team@example.com';

-- ============================================

-- STEP 0: Create logging table in the target DB (if not exists)

-- ============================================

DECLARE @CreateLogTableSQL NVARCHAR(MAX) = '

IF NOT EXISTS (

SELECT \* FROM [' + @DBName + '].sys.tables WHERE name = ''IndexMaintenanceLog''

)

BEGIN

USE [' + @DBName + '];

CREATE TABLE dbo.IndexMaintenanceLog (

LogID INT IDENTITY(1,1) PRIMARY KEY,

SchemaName SYSNAME,

TableName SYSNAME,

IndexName SYSNAME,

PartitionNumber INT,

FragmentationPercent FLOAT,

ActionTaken VARCHAR(20),

Status VARCHAR(20),

ErrorMessage NVARCHAR(1000),

ExecutedAt DATETIME DEFAULT GETDATE()

);

END';

EXEC sp\_executesql @CreateLogTableSQL;

-- ============================================

-- STEP 1: Create the SQL Agent Job

-- ============================================

IF EXISTS (SELECT \* FROM msdb.dbo.sysjobs WHERE name = N'Partitioned Index Maintenance')

BEGIN

EXEC msdb.dbo.sp\_delete\_job @job\_name = N'Partitioned Index Maintenance';

END

EXEC msdb.dbo.sp\_add\_job

@job\_name = N'Partitioned Index Maintenance',

@enabled = 1,

@description = N'Maintains partitioned indexes with logging and email report';

-- ============================================

-- STEP 2: Add job steps

-- ============================================

-- STEP 1: Index Maintenance with Logging

EXEC msdb.dbo.sp\_add\_jobstep

@job\_name = N'Partitioned Index Maintenance',

@step\_name = N'Maintain Partitioned Indexes',

@subsystem = N'TSQL',

@database\_name = @DBName,

@command = N'

DECLARE @ThresholdLow INT = 10;

DECLARE @ThresholdHigh INT = 30;

DECLARE @SQL NVARCHAR(MAX), @Error NVARCHAR(1000);

DECLARE PartitionCursor CURSOR FOR

SELECT

s.name, o.name, i.name, ps.index\_id, ps.partition\_number, ps.avg\_fragmentation\_in\_percent

FROM sys.dm\_db\_index\_physical\_stats(DB\_ID(), NULL, NULL, NULL, ''LIMITED'') ps

JOIN sys.indexes i ON ps.object\_id = i.object\_id AND ps.index\_id = i.index\_id

JOIN sys.objects o ON i.object\_id = o.object\_id

JOIN sys.schemas s ON o.schema\_id = s.schema\_id

WHERE ps.page\_count > 10000

AND ps.avg\_fragmentation\_in\_percent > @ThresholdLow

AND i.type\_desc IN (''CLUSTERED'', ''NONCLUSTERED'')

AND i.data\_space\_id IN (SELECT data\_space\_id FROM sys.partition\_schemes);

DECLARE

@Schema SYSNAME, @Table SYSNAME, @Index SYSNAME,

@Partition INT, @Fragmentation FLOAT, @IndexSQL NVARCHAR(MAX), @Action VARCHAR(20);

OPEN PartitionCursor;

FETCH NEXT FROM PartitionCursor INTO @Schema, @Table, @Index, @Partition, @Partition, @Fragmentation;

WHILE @@FETCH\_STATUS = 0

BEGIN

BEGIN TRY

IF @Fragmentation >= @ThresholdHigh

BEGIN

SET @Action = ''REBUILD'';

SET @IndexSQL = ''ALTER INDEX ['' + @Index + ''] ON ['' + @Schema + ''].['' + @Table + ''] REBUILD PARTITION = '' + CAST(@Partition AS VARCHAR) + '' WITH (ONLINE = ON, SORT\_IN\_TEMPDB = ON, MAXDOP = 4);'';

END

ELSE

BEGIN

SET @Action = ''REORGANIZE'';

SET @IndexSQL = ''ALTER INDEX ['' + @Index + ''] ON ['' + @Schema + ''].['' + @Table + ''] REORGANIZE PARTITION = '' + CAST(@Partition AS VARCHAR) + '';'';

END

EXEC sp\_executesql @IndexSQL;

INSERT INTO dbo.IndexMaintenanceLog (SchemaName, TableName, IndexName, PartitionNumber, FragmentationPercent, ActionTaken, Status)

VALUES (@Schema, @Table, @Index, @Partition, @Fragmentation, @Action, ''SUCCESS'');

END TRY

BEGIN CATCH

SET @Error = ERROR\_MESSAGE();

INSERT INTO dbo.IndexMaintenanceLog (SchemaName, TableName, IndexName, PartitionNumber, FragmentationPercent, ActionTaken, Status, ErrorMessage)

VALUES (@Schema, @Table, @Index, @Partition, @Fragmentation, @Action, ''FAILED'', @Error);

END CATCH

FETCH NEXT FROM PartitionCursor INTO @Schema, @Table, @Index, @Partition, @Partition, @Fragmentation;

END

CLOSE PartitionCursor;

DEALLOCATE PartitionCursor;

',

@on\_success\_action = 3; -- go to next step

-- STEP 2: Send Email Summary (HTML Format)

EXEC msdb.dbo.sp\_add\_jobstep

@job\_name = N'Partitioned Index Maintenance',

@step\_name = N'Send Email Report',

@subsystem = N'TSQL',

@database\_name = @DBName,

@command = N'

DECLARE @Body NVARCHAR(MAX);

SELECT @Body =

''<h2>Partitioned Index Maintenance Report - '' + CONVERT(VARCHAR, GETDATE(), 120) + ''</h2>'' +

''<p><b>Total Entries:</b> '' + CAST(COUNT(\*) AS VARCHAR) + ''<br>'' +

''<b>Successful:</b> '' + CAST(SUM(CASE WHEN Status = ''SUCCESS'' THEN 1 ELSE 0 END) AS VARCHAR) + ''<br>'' +

''<b>Failed:</b> '' + CAST(SUM(CASE WHEN Status = ''FAILED'' THEN 1 ELSE 0 END) AS VARCHAR) + ''</p>'' +

ISNULL(

(

SELECT

''<h4>Recent Failures:</h4><ul>'' +

STRING\_AGG(''<li>'' + SchemaName + ''.'' + TableName + ''.'' + IndexName + '' (P'' + CAST(PartitionNumber AS VARCHAR) + ''): '' + ErrorMessage + ''</li>'', '''') +

''</ul>''

FROM dbo.IndexMaintenanceLog

WHERE ExecutedAt >= DATEADD(HOUR, -6, GETDATE()) AND Status = ''FAILED''

),

''<p>No recent failures.</p>''

);

EXEC msdb.dbo.sp\_send\_dbmail

@profile\_name = ''' + @MailProfileName + ''',

@recipients = ''' + @EmailRecipients + ''',

@subject = ''Partitioned Index Maintenance Report'',

@body = @Body,

@body\_format = ''HTML'';

',

@on\_success\_action = 3;

-- STEP 3: Retention – Delete Logs Older Than 90 Days

EXEC msdb.dbo.sp\_add\_jobstep

@job\_name = N'Partitioned Index Maintenance',

@step\_name = N'Cleanup Old Logs',

@subsystem = N'TSQL',

@database\_name = @DBName,

@command = N'

DELETE FROM dbo.IndexMaintenanceLog

WHERE ExecutedAt < DATEADD(DAY, -90, GETDATE());

';

-- ============================================

-- STEP 3: Schedule the job

-- ============================================

EXEC msdb.dbo.sp\_add\_schedule

@schedule\_name = N'Weekly Partitioned Index Maintenance',

@enabled = 1,

@freq\_type = 8, -- Weekly

@freq\_interval = 1, -- Sunday

@active\_start\_time = 020000; -- 2:00 AM

EXEC msdb.dbo.sp\_attach\_schedule

@job\_name = N'Partitioned Index Maintenance',

@schedule\_name = N'Weekly Partitioned Index Maintenance';

-- ============================================

-- FINAL: Enable job

-- ============================================

EXEC msdb.dbo.sp\_update\_job

@job\_name = N'Partitioned Index Maintenance',

@enabled = 1;

PRINT '✅ SQL Agent Job "Partitioned Index Maintenance" created successfully.';  
  
**💬 Done!**

**This Job:**

* 🧠 Rebuilds only fragmented partitions (smart)
* 📊 Logs each action with status and error (if any)
* 📧 Sends a clean HTML summary email
* 🧹 Cleans up logs older than 90 days
* 🕒 Runs **every Sunday at 2:00 AM** (edit