Driving Data Efficiency with SQL Optimization

Objective

To improve database performance and scalability by implementing a structured SQL optimization framework. The project covers index management, statistics updates, fragmentation monitoring, and partitioning strategies to enable faster data retrieval and efficient query execution.

Content Outline with Index Numbers

Details of Each Section

1. Introduction

- **Purpose:** Provide a structured approach to SQL optimization for faster query execution and better resource utilization.
- **Benefit:** Improves data accessibility and scalability for analytics and business reporting.

2. Statistics Search and Update

- **Why:** Query execution plans rely on up-to-date statistics to optimize performance.
- **Actions**

Monitor

```
SELECT
    SCHEMA_NAME(t.schema_id) AS SchemaName,
    t.name AS TableName,
    s.name AS StatisticName,
    sp.last_updated AS LastUpdate,
    DATEDIFF(day, sp.last_updated, GETDATE()) AS DaysSinceLastUpdate,
    sp.rows AS [Row],
    sp.modification_counter AS Modification_Since_Last_Update
FROM sys.stats AS s
JOIN sys.tables AS t
    ON s.object_id = t.object_id
CROSS APPLY sys.dm_db_stats_properties(s.object_id, s.stats_id) AS sp
ORDER BY DaysSinceLastUpdate DESC ,Modification_Since_Last_Update DESC;
```

Result

Results | Messages

	SchemaName	TableName	StatisticName	LastUpdate	DaysSinceLastUpdate	Row	Modification_Since_Last_Update
1	dbo	DatabaseLog	PK_DatabaseLog_DatabaseLogID	2017-10-27 14:36:32.1300000	2623	49	47
2	dbo	DimAccount	PK_DimAccount	2017-10-27 14:36:31.9733333	2623	99	0
3	dbo	DimAccount	_WA_Sys_00000002_3B75D760	2017-10-27 14:36:33.1066667	2623	99	0
4	dbo	DimProductCategory	PK_DimProductCategory_ProductCategoryKey	2017-10-27 14:36:32.1000000	2623	4	0
5	dbo	DimProductCategory	AK_DimProductCategory_ProductCategoryAlternateKey	2017-10-27 14:36:33.0700000	2623	4	0
6	dbo	DimProductSubcategory	PK_DimProductSubcategory_ProductSubcategoryKey	2017-10-27 14:36:32.1066667	2623	37	0

Update Statistic -Table/Stats

```
UPDATE STATISTICS DatabaseLog
```

Update all Statistics

```
EXEC sp_updatestats
```

- **Benefit:** Reduces full table scans and enhances query execution speed.

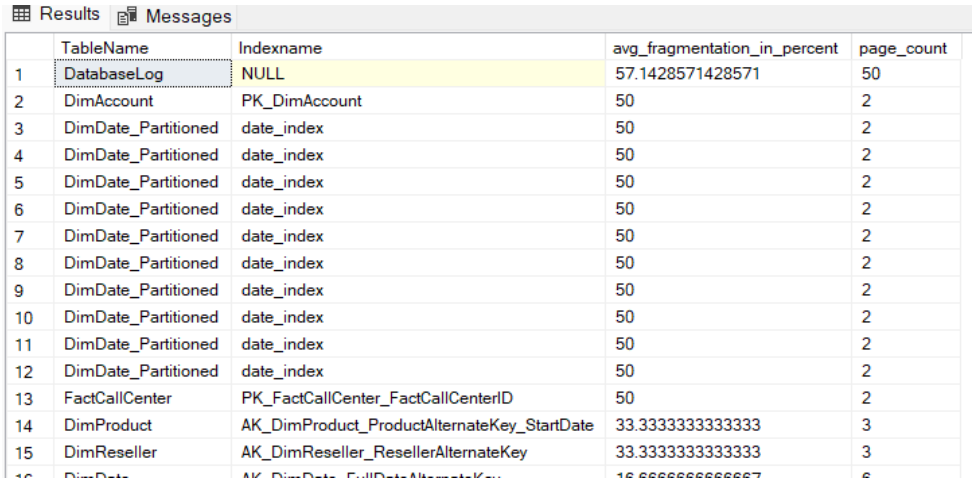
3. Fragmentation Monitoring and Handling

- **Why:** Fragmented indexes increase the I/O required for data retrieval.
- **Actions:**

Monitor

```
SELECT
    tbl.name AS TableName,
    idx.name AS Indexname,
    s.avg_fragmentation_in_percent,
    s.page_count
FROM sys.dm_db_index_physical_stats (DB_ID(),NULL,NULL,NULL,'LIMITED') s
INNER JOIN sys.tables tbl
ON s.object_id=tbl.object_id
INNER JOIN sys.indexes AS idx
ON idx.object_id=s.object_id
AND idx.index_id=s.index_id
ORDER BY avg_fragmentation_in_percent DESC;
```

Result



Results | Messages

	TableName	Indexname	avg_fragmentation_in_percent	page_count
1	DatabaseLog	NULL	57.1428571428571	50
2	DimAccount	PK_DimAccount	50	2
3	DimDate_Partitioned	date_index	50	2
4	DimDate_Partitioned	date_index	50	2
5	DimDate_Partitioned	date_index	50	2
6	DimDate_Partitioned	date_index	50	2
7	DimDate_Partitioned	date_index	50	2
8	DimDate_Partitioned	date_index	50	2
9	DimDate_Partitioned	date_index	50	2
10	DimDate_Partitioned	date_index	50	2
11	DimDate_Partitioned	date_index	50	2
12	DimDate_Partitioned	date_index	50	2
13	FactCallCenter	PK_FactCallCenter_FactCallCenterID	50	2
14	DimProduct	AK_DimProduct_ProductAlternateKey_StartDate	33.3333333333333	3
15	DimReseller	AK_DimReseller_ResellerAlternateKey	33.3333333333333	3

Reorganize for low fragmentation (<30%) / Rebuild for high fragmentation (>30%)

```
ALTER INDEX date_index ON DimDate_Partitioned REBUILD
```

Benefit: Enhances read and write performance by optimizing storage layout.

4. Index Usage Monitoring

- **Why:** Identifies indexes actively contributing to performance or those that are redundant.
- **Actions:** Analyze usage patterns using:

Monitor

```
SELECT
    tbl.name AS TableName,
    idx.name AS IndexName,
    idx.type_desc,
    idx.is_primary_key,
    idx.is_unique,
    s.user_seeks,
    s.user_scans,
    s.user_lookups,
    s.user_updates,
    s.last_user_seek,
    COALESCE(s.last_user_scan, s.last_user_seek) AS LastUpdate
FROM sys.indexes idx
JOIN sys.tables tbl ON idx.object_id = tbl.object_id
LEFT JOIN sys.dm_db_index_usage_stats s ON s.object_id = idx.object_id AND s.index_id = idx.index_id
ORDER BY  s.user seeks,s.user scans,s.user lookups;
```

Result

Results | Messages

	TableName	IndexName	type_desc	is_primary_key	is_unique	user_seeks	user_scans	user_lookups	user_updates	last_user_seek	LastUpdate
1	DimProduct	AK_DimProduct_ProductAlternateKey_StartDate	NONCLUSTERED	0	1	NULL	NULL	NULL	NULL	NULL	NULL
2	DimProductCategory	PK_DimProductCategory_ProductCategoryKey	CLUSTERED	1	1	NULL	NULL	NULL	NULL	NULL	NULL
3	DimProductCategory	AK_DimProductCategory_ProductCategoryAlternateKey	NONCLUSTERED	0	1	NULL	NULL	NULL	NULL	NULL	NULL
4	DimProductSubcategory	PK_DimProductSubcategory_ProductSubcategoryKey	CLUSTERED	1	1	NULL	NULL	NULL	NULL	NULL	NULL
5	DimProductSubcategory	AK_DimProductSubcategory_ProductSubcategoryAlter...	NONCLUSTERED	0	1	NULL	NULL	NULL	NULL	NULL	NULL
6	DimPromotion	PK_DimPromotion_PromotionKey	CLUSTERED	1	1	NULL	NULL	NULL	NULL	NULL	NULL
7	DimPromotion	AK_DimPromotion_PromotionAlternateKey	NONCLUSTERED	0	1	NULL	NULL	NULL	NULL	NULL	NULL
8	DimReseller	PK_DimReseller_ResellerKey	CLUSTERED	1	1	NULL	NULL	NULL	NULL	NULL	NULL
9	DimReseller	AK_DimReseller_ResellerAlternateKey	NONCLUSTERED	0	1	NULL	NULL	NULL	NULL	NULL	NULL
10	DimSalesReason	PK_DimSalesReason_SalesReasonKey	CLUSTERED	1	1	NULL	NULL	NULL	NULL	NULL	NULL
11	DimSalesTerritory	PK_DimSalesTerritory_SalesTerritoryKey	CLUSTERED	1	1	NULL	NULL	NULL	NULL	NULL	NULL

Benefit: Eliminates unnecessary maintenance on unused indexes.

5. Handling Missing and Duplicate Indexes

- **Why:** Missing indexes slow down queries; duplicate indexes waste storage.
- **Actions:**

Monitor Missing Indexes

```
SELECT
*
FROM sys.dm_db_missing_index_details
```

Monitor Duplicate Indexes

```
SELECT
    tbl.name AS TableName,
    col.name AS IndexColumn,
    idx.name AS IndexName,
    idx.type_desc AS IndexType,
    COUNT(*) OVER (PARTITION BY tbl.name,col.name)ColumnCount
FROM sys.indexes idx
JOIN sys.tables tbl ON idx.object_id=tbl.object_id
JOIN sys.index_columns ic ON idx.object_id=ic.object_id AND idx.index_id = ic.index_id
JOIN sys.columns col ON ic.object_id=col.object_id AND ic.column_id= col.column_id
ORDER BY ColumnCount DESC;
```

Result

Results | Messages

	TableName	IndexColumn	IndexName	IndexType	ColumnCount
1	ci	SalesOrderLineNumber	RI_INDEXsFFs	NONCLUSTERED	3
2	ci	SalesOrderLineNumber	RI_INDEXsFBJFs	NONCLUSTERED	3
3	ci	SalesOrderLineNumber	RI_INDEXNGVCS	CLUSTERED COLUMNSTORE	3
4	ci	SalesOrderNumber	RI_INDEXNGVCS	CLUSTERED COLUMNSTORE	2
5	ci	SalesOrderNumber	RI_INDEXsFFs	NONCLUSTERED	2
6	DimDate_Partitioned	FullDateAlternateKey	date_index	NONCLUSTERED	2
7	DimDate_Partitioned	FullDateAlternateKey	column_index	CLUSTERED COLUMNSTORE	2
8	DimDepartmentGroup	DepartmentGroupKey	PK_DimDepartmentGroup	CLUSTERED	1
9	DimEmployee	EmployeeKey	PK_DimEmployee_EmployeeKey	CLUSTERED	1

Benefit: Reduces query latency and storage costs.

6. Query Performance Optimization

- **Why:** Slow queries consume excessive resources, degrading overall performance.
- **Actions:**

Monitor

```
SELECT
    qs.total_elapsed_time,
    qs.execution_count,
    qs.total_worker_time AS Total_CPU_Time,
    qs.last_execution_time,
    qt.text AS query_test
FROM sys.dm_exec_query_stats qs
CROSS APPLY sys.dm_exec_sql_text(qs.sql_handle)qt;
```

Output

Results | Messages

	total_elapsed_time	execution_count	Total_CPU_Time	last_execution_time	query_test
1	11888	1	5200	2025-01-01 12:07:19.040	SELECT tbl.name AS TableName, col.name
2	1561	1	1559	2025-01-01 12:06:44.117	SELECT qs.total_elapsed_time, qs.execution_
3	2763	1	2761	2025-01-01 12:06:54.713	SELECT qs.total_elapsed_time, qs.execution_

- Optimize execution plans via indexing, joins, and reducing scans.
- **Benefit:** Improves query response times and resource efficiency.

7. Partitioning Setup

7.A Partition Functions

- **Why:** Logical segmentation of data for faster retrieval.
- **Action:**

```
CREATE PARTITION FUNCTION PartitionByYear (datetime)
AS RANGE LEFT FOR VALUES
('2005-12-31', '2006-12-31', '2007-12-31', '2008-12-31',
 '2009-12-31', '2010-12-31', '2011-12-31', '2012-12-31',
 '2013-12-31', '2014-12-31');
```

Monitor

```
SELECT
    name,
    function_id,
    type,
    type_desc,
    boundary_value_on_right
FROM sys.partition_functions
```

Result

Results | Messages

	name	function_id	type	type_desc	boundary_value_on_right
1	PartitionByYear	65538	R	RANGE	0

7.B Filegroups and Data Files

- **Why:** Distributes data across physical storage for scalability.
- **Actions:**

File groups

```
ALTER DATABASE AdventureWorksDW2022 ADD FILEGROUP FG_2005
ALTER DATABASE AdventureWorksDW2022 ADD FILEGROUP FG_2006
ALTER DATABASE AdventureWorksDW2022 ADD FILEGROUP FG_2007
ALTER DATABASE AdventureWorksDW2022 ADD FILEGROUP FG_2008
ALTER DATABASE AdventureWorksDW2022 ADD FILEGROUP FG_2009
ALTER DATABASE AdventureWorksDW2022 ADD FILEGROUP FG_2010
ALTER DATABASE AdventureWorksDW2022 ADD FILEGROUP FG_2011
ALTER DATABASE AdventureWorksDW2022 ADD FILEGROUP FG_2012
ALTER DATABASE AdventureWorksDW2022 ADD FILEGROUP FG_2013
ALTER DATABASE AdventureWorksDW2022 ADD FILEGROUP FG_2014
ALTER DATABASE AdventureWorksDW2022 ADD FILEGROUP FG_2015
ALTER DATABASE AdventureWorksDW2022 ADD FILEGROUP FG_2016
--REMOVE
ALTER DATABASE AdventureWorksDW2022 REMOVE FILEGROUP FG_2016
```

Monitor

```
SELECT
*
FROM sys.filegroups
```

Result

Results | Messages

	name	data_space_id	type	type_desc	is_default	is_system	filegroup_guid	log_filegroup
1	PRIMARY	1	FG	ROWS_FILEGROUP	1	0	NULL	NULL
2	FG_2005	2	FG	ROWS_FILEGROUP	0	0	A3A152B9-83D2-442E-8212-65A9FCB75223	NULL
3	FG_2006	3	FG	ROWS_FILEGROUP	0	0	F64D8589-C00C-4156-84AB-8A59C7679540	NULL
4	FG_2007	4	FG	ROWS_FILEGROUP	0	0	63F55CCB-97EE-4D07-9154-BBF4CAAC1179	NULL
5	FG_2008	5	FG	ROWS_FILEGROUP	0	0	FDBDB066-72CB-420A-A311-A2891B12A67A	NULL
6	FG_2009	6	FG	ROWS_FILEGROUP	0	0	404AFCBD-0A9F-438B-9BB3-D9F9F421A430	NULL
7	FG_2010	7	FG	ROWS_FILEGROUP	0	0	5BF52C94-8618-4EAF-9009-18F5AB6C9EDB	NULL
8	FG_2011	8	FG	ROWS_FILEGROUP	0	0	C6A652F8-61EC-4F73-8E39-80F899BAEC04	NULL
9	FG_2012	9	FG	ROWS_FILEGROUP	0	0	561B6B31-6A18-4D92-A935-4E485B640151	NULL
10	FG_2013	10	FG	ROWS_FILEGROUP	0	0	357E756C-1DAB-413F-9716-C975F3F6A362	NULL
11	FG_2014	11	FG	ROWS_FILEGROUP	0	0	061EDAE0-58EE-4FCF-83D9-93FFB2143A7C	NULL
12	FG_2015	12	FG	ROWS_FILEGROUP	0	0	8710A19D-FCDE-4885-A017-497CBDE4846E	NULL

Data Files

```
ALTER DATABASE AdventureWorksDW2022
ADD FILE
(
    NAME = P_2005,
    FILENAME = 'C:\Program Files\Microsoft SQL Server\MSSQL16.SQLEXPRESS\MSSQL\DATA\P_2005.ndf'
)
TO FILEGROUP FG_2005;

ALTER DATABASE AdventureWorksDW2022
ADD FILE
(
    NAME = P_2006,
    FILENAME = 'C:\Program Files\Microsoft SQL Server\MSSQL16.SQLEXPRESS\MSSQL\DATA\P_2006.ndf'
```

Monitor

```
SELECT
    fg.name AS FileGroupName,
    mf.name ASLogicalfilename,
    mf.physical_name AS physicalfilepath,
    mf.size/128 AS size_mb
FROM sys.filegroups fg
JOIN sys.master_files mf ON fg.data_space_id=mf.data_space_id
WHERE mf.database_id=DB_ID ('AdventureWorksDW2022');
```

Result

Results | Messages

	FileGroupName	ASLogicalfilename	physicalfilepath	size_mb
1	PRIMARY	AdventureWorksDW2022	C:\Program Files\Microsoft SQL Server\MSSQL16.SQL...	200
2	FG_2005	P_2005	C:\Program Files\Microsoft SQL Server\MSSQL16.SQL...	72
3	FG_2006	P_2006	C:\Program Files\Microsoft SQL Server\MSSQL16.SQL...	72
4	FG_2007	P_2007	C:\Program Files\Microsoft SQL Server\MSSQL16.SQL...	72
5	FG_2008	P_2008	C:\Program Files\Microsoft SQL Server\MSSQL16.SQL...	72
6	FG_2009	P_2009	C:\Program Files\Microsoft SQL Server\MSSQL16.SQL...	72
7	FG_2010	P_2010	C:\Program Files\Microsoft SQL Server\MSSQL16.SQL...	72
8	FG_2011	P_2011	C:\Program Files\Microsoft SQL Server\MSSQL16.SQL...	72
9	FG_2012	P_2012	C:\Program Files\Microsoft SQL Server\MSSQL16.SQL...	72
10	FG_2013	P_2013	C:\Program Files\Microsoft SQL Server\MSSQL16.SQL...	72
11	FG_2014	P_2014	C:\Program Files\Microsoft SQL Server\MSSQL16.SQL...	72
12	FG_2015	P_2015	C:\Program Files\Microsoft SQL Server\MSSQL16.SQL...	72

7.C Partition Scheme

- **Why:** Maps partitions to filegroups.
- **Action:**

```
CREATE PARTITION SCHEME SchemePartitionByYear
AS PARTITION PartitionByYear
TO (FG_2005,FG_2006,FG_2007,FG_2008,FG_2009,FG_2010,FG_2011,FG_2012,FG_2013,FG_2014,FG_2015)
```

Monitor

```
SELECT
    ps.name AS PARTITIONSHEMENAME,
    pf.name AS partitionfunctionname,
    ds.destination_id AS partitionnumber,
    fg.name AS filegroupname
FROM sys.partition_schemes ps
JOIN sys.partition_functions pf ON ps.function_id=pf.function_id
JOIN sys.destination_data_spaces ds ON ps.data_space_id=ds.partition_scheme_id
JOIN sys.filegroups fg ON ds.data_space_id=fg.data_space_id
```

Result

Results | Messages

	PARTITIONSHEMENAME	partitionfunctionname	partitionnumber	filegroupname
1	SchemePartitionByYear	PartitionByYear	1	FG_2005
2	SchemePartitionByYear	PartitionByYear	2	FG_2006
3	SchemePartitionByYear	PartitionByYear	3	FG_2007
4	SchemePartitionByYear	PartitionByYear	4	FG_2008
5	SchemePartitionByYear	PartitionByYear	5	FG_2009
6	SchemePartitionByYear	PartitionByYear	6	FG_2010
7	SchemePartitionByYear	PartitionByYear	7	FG_2011
8	SchemePartitionByYear	PartitionByYear	8	FG_2012
9	SchemePartitionByYear	PartitionByYear	9	FG_2013
10	SchemePartitionByYear	PartitionByYear	10	FG_2014
11	SchemePartitionByYear	PartitionByYear	11	FG_2015

7.D Partitioned Table Creation & Data Insert

- **Why:** Organizes large tables into manageable segments.
- **Action:**

Create Partition Table

```
CREATE TABLE dbo.DimDate_Partitioned
(
    DateKey INT ,
    FullDateAlternateKey DATETIME,
    DayNumberOfWeek INT,
    EnglishDayNameOfWeek NVARCHAR(20),

)
ON SchemePartitionByYear (FullDateAlternateKey)
```

Insert Data

```
INSERT INTO dbo.DimDate_Partitioned (DateKey, FullDateAlternateKey, DayNumberOfWeek, EnglishDayNameOfWeek)
SELECT DateKey, FullDateAlternateKey, DayNumberOfWeek, EnglishDayNameOfWeek
FROM #TEMP_TAB;
```

Monitor

```
SELECT
    p.partition_number AS PartitionNumber,
    fg.name AS PartitionFileGroup,
    p.rows AS NumberOfRows
FROM sys.partitions p
JOIN sys.destination_data_spaces dds ON p.partition_number = dds.destination_id
JOIN sys.filegroups fg ON dds.data_space_id = fg.data_space_id
WHERE OBJECT_NAME(p.object_id)='DimDate_Partitioned';
```

Result

Results | Messages

	PartitionNumber	PartitionFileGroup	NumberOfRows
1	1	FG_2005	365
2	2	FG_2006	365
3	3	FG_2007	365
4	4	FG_2008	366
5	5	FG_2009	365
6	6	FG_2010	365
7	7	FG_2011	365
8	8	FG_2012	366
9	9	FG_2013	365
10	10	FG_2014	365
11	11	FG_2015	1

Benefit:

- Enables efficient query performance for large datasets.
- Balances I/O workload and supports future data growth.
- Aligns data storage with query access patterns.
- Faster query performance for partitioned datasets.

8. Conclusion and Benefits

- **Key Outcomes:** Faster queries, efficient resource utilization, and improved scalability.
- **Impact:** This structured approach to SQL optimization ensures data systems are ready to handle growing demands in data analysis and reporting.