Testing Columnstore Merge Policy

Test Case Scenario 1 – Merge smaller rowgroups into one 2

Test Case Description: 2

Steps taken to test the MERGE policy for smaller rowgroups: 2

Test Case Scenario 2 – Self-merge 5

Test Case Description: 5

Steps taken to test the MERGE policy for Self-Merge: 5

Test Case Scenario 3 – Merging of multiple rowgroups is preferred 8

Test Case Description: 8

Steps taken to test the MERGE policy for Self-Merge vs. Multiple RGs Merge: 8

Test Case Scenario 4 – Rowgroups merged in sequential order 10

Test Case Description: 10

Steps taken to test the MERGE in sequential order: 10

Test Case Scenario 5 – Rowgroups that won't qualify for MERGE 12

Test Case Description: 12

Steps taken to test the MERGE in sequential order: 12

Test Case Scenario 1 – Merge smaller rowgroups into one

Test Case Description:

Combine one or more compressed rowgroups such that total number of rows <= 1,024,576. For example, if you bulk import 5 batches of size 102400, you will get 5 compressed rowgroups. Now if you run REORGANIZE command, these rowgroups will get merged into 1 compressed rowgroup of size 512000 rows assuming there were no dictionary size or memory limitation.

Steps taken to test the MERGE policy for smaller rowgroups:

1.) Bulk load 5 batches of size 102400 and review the row groups physical stats:

```sql
SELECT
    OBJECT_NAME(object_id) AS table_name,
    row_group_id,
    total_rows,
    state_desc,
    transition_to_compressed_state_desc
FROM
    sys.dm_db_column_store_row_group_physical_stats
WHERE
    object_id = OBJECT_ID('Production.TransactionHistory_DST')
ORDER BY
    row_group_id ASC;
```

100 %

Results | Messages

	table_name	row_group_id	total_rows	state_desc	transition_to_compressed_state_desc
1	TransactionHistory_DST	0	102400	COMPRESSED	BULKLOAD
2	TransactionHistory_DST	1	102400	COMPRESSED	BULKLOAD
3	TransactionHistory_DST	2	102400	COMPRESSED	BULKLOAD
4	TransactionHistory_DST	3	102400	COMPRESSED	BULKLOAD
5	TransactionHistory_DST	4	102400	COMPRESSED	BULKLOAD

2.) Let's REORGANIZE the CCI index:

```sql
ALTER INDEX CCI_TransactionHistory_DST ON Production.TransactionHistory_DST
REORGANIZE WITH (COMPRESS_ALL_ROW_GROUPS = ON);
```

3.) Immediately after REORGANIZE operation finished, let's have a look at the row groups statistics again:

```
SELECT
    OBJECT_NAME(object_id) AS table_name,
    row_group_id,
    total_rows,
    state_desc,
    transition_to_compressed_state_desc
FROM
    sys.dm_db_column_store_row_group_physical_stats
WHERE
    object_id = OBJECT_ID('Production.TransactionHistory_DST')
ORDER BY
    row_group_id ASC;
```

100 %

Results | Messages

	table_name	row_group_id	total_rows	state_desc	transition_to_compressed_state_desc
1	TransactionHistory_DST	0	102400	TOMBSTONE	NULL
2	TransactionHistory_DST	1	102400	TOMBSTONE	NULL
3	TransactionHistory_DST	2	102400	TOMBSTONE	NULL
4	TransactionHistory_DST	3	102400	TOMBSTONE	NULL
5	TransactionHistory_DST	4	102400	TOMBSTONE	NULL
6	TransactionHistory_DST	5	512000	COMPRESSED	MERGE

As we can see, the original rowgroups have been deallocated by tuple mover and marked as **'TOMBSTONE'**.

4.) Now, we can review the extended event session **'Tuple_Mover_Xe'** and have a look what can we see there:

Event: columnstore_rowgroup_merge_complete (2018-07-11 21:34:52.5035805)

Details

Field	Value
cpu_time	4227590
database_id	5
index_id	1
is_aggressive_compression	False
is_in_memory	0
memory_available_kb	3526496
memory_required_kb	643888
merge_policy	Merging multiple rowgroups.
object_id	116195464
partition_number	1
source_deleted_row_count	0
source_row_count	512000
source_rowgroup_id_list	0,1,2,3,4
source_rowgroup_id_list_length	5
target_rowgroup_id_list	5
target_rowgroup_id_list_length	1

As we can see in the screenshot above, merge policy **'Merging multiple rowgroups'** has been applied. We can also see that source rowgroups 0, 1, 2, 3, 4 have been merged into the rowgroup 5.

5.) 'Let's wait a few minutes and review the rowgroups physical stats:

Results | Messages

	table_name	row_group_id	total_rows	state_desc	transition_to_compressed_state_desc
1	TransactionHistory_DST	5	512000	COMPRESSED	MERGE

As we can see, the **'TOMBSTONE'** rowgroups have been garbage collected and removed in the background by system.

Test Case Scenario 2 – Self-merge

Test Case Description:

The row group has 10% or more rows deleted. When > 102400 rows are marked deleted, a compressed RG is eligible for self-merge. For example, if a compressed row group of 1,048,576 million rows has 110k rows deleted, we can remove the deleted rows and recompress the rowgroup with the remaining rows. It saves on the storage by removing deleted rows.

Steps taken to test the MERGE policy for Self-Merge:

1.) Load 3 batches of 1,048,576 million rows and review the row groups physical stats and data distribution:

Results | Messages

	table_name	row_group_id	total_rows	deleted_rows	state_desc	transition_to_compressed_state_desc
1	TransactionHistory_DST	0	1048576	0	COMPRESSED	BULKLOAD
2	TransactionHistory_DST	1	1048576	0	COMPRESSED	BULKLOAD
3	TransactionHistory_DST	2	1048576	0	COMPRESSED	BULKLOAD

And see also the on_disk_size and data distribution (min, max data ID, based on the TransactionID):

Results | Messages

	table_name	segment_id	min_data_id	max_data_id	row_count	on_disk_size
1	TransactionHistory_DST	0	-2147483648	-2146435073	1048576	2796792
2	TransactionHistory_DST	1	-2146435072	-2145386497	1048576	2796792
3	TransactionHistory_DST	2	-2145386496	-2144337921	1048576	2796792

2.) Let's DELETE 10% of rows from the 1st rowgroup:

```
DELETE TOP (10) PERCENT
FROM
    Production.TransactionHistory_DST
WHERE
    TransactionID BETWEEN -2147483648 AND -2146435073; -- Range for the 1st group
```

3.) And REORGANIZE the CCI:

```
ALTER INDEX CCI_TransactionHistory_DST ON Production.TransactionHistory_DST
REORGANIZE WITH (COMPRESS_ALL_ROW_GROUPS = ON);
```

4.) After REORGANIZE, let's review RG physical stats and data distribution (based on the TransactionID column):

Results | Messages

	table_name	row_group_id	total_rows	deleted_rows	state_desc	transition_to_compressed_state_desc
1	TransactionHistory_DST	0	1048576	104858	TOMBSTONE	NULL
2	TransactionHistory_DST	1	1048576	0	COMPRESSED	BULKLOAD
3	TransactionHistory_DST	2	1048576	0	COMPRESSED	BULKLOAD
4	TransactionHistory_DST	3	943718	0	COMPRESSED	MERGE

As we can see, the 1st rowgroup with deleted records has been ”self-merged” and the new RG 3 has been created. We can also see, that on_disk_space has been reduced:

```sql
-- Rougroups data distribution:
SELECT
    OBJECT_NAME(p.object_id) AS table_name,
    cs.segment_id,
    cs.min_data_id,
    cs.max_data_id,
    cs.row_count,
    cs.on_disk_size
FROM
    sys.column_store_segments AS cs
    INNER JOIN sys.partitions AS p ON cs.hobt_id = p.hobt_id
WHERE
    p.object_id = OBJECT_ID('Production.TransactionHistory_DST') AND
    cs.column_id = 1
ORDER BY
    cs.segment_id ASC;
```

100 %

Results | Messages

	table_name	segment_id	min_data_id	max_data_id	row_count	on_disk_size
1	TransactionHistory_DST	0	-2147483648	-2146435073	1048576	2796792
2	TransactionHistory_DST	1	-2146435072	-2145386497	1048576	2796792
3	TransactionHistory_DST	2	-2145386496	-2144337921	1048576	2796792
4	TransactionHistory_DST	3	-2147483648	-2146435073	943718	2517576

5.) Checking the Extended Event session **'Tuple_Mover_Xe'**, we can see the Merge Policy applied, count of deleted records and also the source and target RG IDs:

Event: columnstore_rowgroup_merge_complete (2018-07-15 19:59:49.9784211)

Details

Field	Value
cpu_time	5855463
database_id	5
index_id	1
is_aggressive_compression	False
is_in_memory	0
memory_available_kb	3669032
memory_required_kb	738736
merge_policy	Removing deleted rows form single rowgroup.
object_id	292196091
partition_number	1
source_deleted_row_count	104858
source_row_count	1048576
source_rowgroup_id_list	0
source_rowgroup_id_list_length	1
target_rowgroup_id_list	3
target_rowgroup_id_list_length	1

Test Case Scenario 3 – Merging of multiple rowgroups is preferred

Test Case Description:

If there is a choice between self-merge or merging two or more rowgroups, **the merging of two or more rowgroups is favored**. For example, if there are two compressed rowgroups RG1 with 500k rows and RG2 with 1,048,576 rows but 60% of the rows are deleted. In this case, instead of self-merging RG2, the merge policy will combine RG1 and RG2 into one compressed rowgroup.

Steps taken to test the MERGE policy for Self-Merge vs. Multiple RGs Merge:

1.) Load 2 batches: first one that has 500k rows and a second one that has 1,048,576 million rows:

Results | Messages

	table_name	row_group_id	total_rows	deleted_rows	state_desc	transition_to_compressed_state_desc
1	TransactionHistory_DST	0	500000	0	COMPRESSED	BULKLOAD
2	TransactionHistory_DST	1	1048576	0	COMPRESSED	BULKLOAD

And see also the on_disk_size and data distribution (min, max data ID, based on the TransactionID):

Results | Messages

	table_name	segment_id	min_data_id	max_data_id	row_count	on_disk_size
1	TransactionHistory_DST	0	-2147483648	-2146983649	500000	1334384
2	TransactionHistory_DST	1	-2146983648	-2145935073	1048576	2796792

2.) Let's DELETE 60% of rows from the 2nd RG:

```
DELETE TOP (60) PERCENT
FROM
    Production.TransactionHistory_DST
WHERE
    TransactionID BETWEEN -2146983648 AND -2145935073; -- Range for the 2nd group
```

3.) And REORGANIZE the CCI:

```
ALTER INDEX CCI_TransactionHistory_DST ON Production.TransactionHistory_DST
REORGANIZE WITH (COMPRESS_ALL_ROW_GROUPS = ON);
```

4.) After REORGANIZE, let's review RG physical stats and data distribution (based on the TransactionID column):

Results | Messages

	table_name	row_group_id	total_rows	deleted_rows	state_desc	transition_to_compressed_state_desc
1	TransactionHistory_DST	0	500000	0	TOMBSTONE	NULL
2	TransactionHistory_DST	1	1048576	629146	TOMBSTONE	NULL
3	TransactionHistory_DST	2	919430	0	COMPRESSED	MERGE

As we can see, the 2 RGs have been merged into the one RG, means the merging of two rowgroups is favoured (rather than performing Self-Merge operation).

5.) In the **'Tuple_Mover_Xe' file**, we can see that merging of multiple RGs is favoured:

Event: columnstore_rowgroup_merge_complete (2018-07-15 20:25:06.6268797)

Details

Field	Value
cpu_time	5950104
database_id	5
index_id	1
is_aggressive_compression	False
is_in_memory	0
memory_available_kb	3469984
memory_required_kb	691160
merge_policy	Merging multiple rowgroups.
object_id	292196091
partition_number	1
source_deleted_row_count	629146
source_row_count	1548576
source_rowgroup_id_list	0,1
source_rowgroup_id_list_length	2
target_rowgroup_id_list	2
target_rowgroup_id_list_length	1

Test Case Scenario 4 – Rowgroups merged in sequential order

Test Case Description:

If you have three rowgroups that qualify for merge, they are considered for **merge in sequential order**. For example, rowgroup1 (500k), rowgroup2 (500k) and rowgroup3 (500k) we will merge the first two qualifying ones.

Steps taken to test the MERGE in sequential order:

1.) Load 3 batches of size 500 000, so we will get 3 compressed RGs:

```sql
-- Rowgroups physical stats:
SELECT
    OBJECT_NAME(object_id) AS table_name,
    row_group_id,
    total_rows,
    deleted_rows,
    state_desc,
    transition_to_compressed_state_desc
FROM
    sys.dm_db_column_store_row_group_physical_stats
WHERE
    object_id = OBJECT_ID('Production.TransactionHistory_DST')
ORDER BY
    row_group_id ASC;
```

100 %

Results | Messages

	table_name	row_group_id	total_rows	deleted_rows	state_desc	transition_to_compressed_state_desc
1	TransactionHistory_DST	0	500000	0	COMPRESSED	BULKLOAD
2	TransactionHistory_DST	1	500000	0	COMPRESSED	BULKLOAD
3	TransactionHistory_DST	2	500000	0	COMPRESSED	BULKLOAD

2.) Let's REORGANIZE the CCI:

```sql
ALTER INDEX CCI_TransactionHistory_DST ON Production.TransactionHistory_DST
REORGANIZE WITH (COMPRESS_ALL_ROW_GROUPS = ON);
```

3.) After REORGANIZE, we can see that RGs have been MERGed in sequential order:

Results | Messages

	table_name	row_group_id	total_rows	state_desc	transition_to_compressed_state_desc
1	TransactionHistory_DST	0	500000	TOMBSTONE	NULL
2	TransactionHistory_DST	1	500000	TOMBSTONE	NULL
3	TransactionHistory_DST	2	500000	COMPRESSED	BULKLOAD
4	TransactionHistory_DST	3	1000000	COMPRESSED	MERGE

4.) We can also observe this behaviour in the **'Tuple_Mover_Xe' file**:

Event: columnstore_rowgroup_merge_complete (2018-07-15 20:48:10.4877146)

Details

Field	Value
cpu_time	6370667
database_id	5
index_id	1
is_aggressive_compression	False
is_in_memory	0
memory_available_kb	3837792
memory_required_kb	718488
merge_policy	Merging multiple rowgroups.
object_id	292196091
partition_number	1
source_deleted_row_count	0
source_row_count	1000000
source_rowgroup_id_list	0,1
source_rowgroup_id_list_length	2
target_rowgroup_id_list	3
target_rowgroup_id_list_length	1

Test Case Scenario 5 – Rowgroups that won't qualify for MERGE

Test Case Description:

Amend the INSERT statement so it contains TransactionDescr of type NVARCHAR(4000). Bulk import 1,024,576 rows, due to DICTIONARY limit (**the size of dictionary is limited to 16MB**) we can observer the RGs with trim reason description **DICTIONARY_SIZE**, hence RGs won't qualify for MERGE.

Steps taken to test the MERGE in sequential order:

1.) Load 1 batch 3 batches of size 1,024,576 rows. But in this case, also include the TransactionDescr column, NVARCHAR(4000), and review the RGs physical stats. Normally, we would expect to have one compressed RG, but due to dictionary size limit (16 MB) we have many RGs with trim reason description DICTIONARY_SIZE:

Results | Messages

	table_name	row_group_id	total_rows	state_desc	transition_to...	trim_reason_desc
1	TransactionHistory_DST	0	68971	COMPRESSED	BULKLOAD	DICTIONARY_SIZE
2	TransactionHistory_DST	1	68971	COMPRESSED	BULKLOAD	DICTIONARY_SIZE
3	TransactionHistory_DST	2	68971	COMPRESSED	BULKLOAD	DICTIONARY_SIZE
4	TransactionHistory_DST	3	68971	COMPRESSED	BULKLOAD	DICTIONARY_SIZE
5	TransactionHistory_DST	4	68971	COMPRESSED	BULKLOAD	DICTIONARY_SIZE
6	TransactionHistory_DST	5	68971	COMPRESSED	BULKLOAD	DICTIONARY_SIZE
7	TransactionHistory_DST	6	68971	COMPRESSED	BULKLOAD	DICTIONARY_SIZE
8	TransactionHistory_DST	7	68971	COMPRESSED	BULKLOAD	DICTIONARY_SIZE
9	TransactionHistory_DST	8	68971	COMPRESSED	BULKLOAD	DICTIONARY_SIZE
10	TransactionHistory_DST	9	68971	COMPRESSED	BULKLOAD	DICTIONARY_SIZE
11	TransactionHistory_DST	10	68971	COMPRESSED	BULKLOAD	DICTIONARY_SIZE
12	TransactionHistory_DST	11	68971	COMPRESSED	BULKLOAD	DICTIONARY_SIZE
13	TransactionHistory_DST	12	68971	COMPRESSED	BULKLOAD	DICTIONARY_SIZE
14	TransactionHistory_DST	13	68971	COMPRESSED	BULKLOAD	DICTIONARY_SIZE
15	TransactionHistory_DST	14	68971	COMPRESSED	BULKLOAD	DICTIONARY_SIZE
16	TransactionHistory_DST	15	14011	OPEN	NULL	NULL

2.) Ok, so let's REORGANIZE the CCI:

```
ALTER INDEX CCI_TransactionHistory_DST ON Production.TransactionHistory_DST
REORGANIZE WITH (COMPRESS_ALL_ROW_GROUPS = ON);
```

3.) As you can see, nothing has really changed after we reorganized the CCI (except one RG moved from Open to Compressed state):

Results | Messages

	table_name	row_gr...	total_rows	state_desc	transition_to_com...	trim_reason_desc
1	TransactionHistory_DST	0	68971	COMPRESSED	BULKLOAD	DICTIONARY_SIZE
2	TransactionHistory_DST	1	68971	COMPRESSED	BULKLOAD	DICTIONARY_SIZE
3	TransactionHistory_DST	2	68971	COMPRESSED	BULKLOAD	DICTIONARY_SIZE
4	TransactionHistory_DST	3	68971	COMPRESSED	BULKLOAD	DICTIONARY_SIZE
5	TransactionHistory_DST	4	68971	COMPRESSED	BULKLOAD	DICTIONARY_SIZE
6	TransactionHistory_DST	5	68971	COMPRESSED	BULKLOAD	DICTIONARY_SIZE
7	TransactionHistory_DST	6	68971	COMPRESSED	BULKLOAD	DICTIONARY_SIZE
8	TransactionHistory_DST	7	68971	COMPRESSED	BULKLOAD	DICTIONARY_SIZE
9	TransactionHistory_DST	8	68971	COMPRESSED	BULKLOAD	DICTIONARY_SIZE
10	TransactionHistory_DST	9	68971	COMPRESSED	BULKLOAD	DICTIONARY_SIZE
11	TransactionHistory_DST	10	68971	COMPRESSED	BULKLOAD	DICTIONARY_SIZE
12	TransactionHistory_DST	11	68971	COMPRESSED	BULKLOAD	DICTIONARY_SIZE
13	TransactionHistory_DST	12	68971	COMPRESSED	BULKLOAD	DICTIONARY_SIZE
14	TransactionHistory_DST	13	68971	COMPRESSED	BULKLOAD	DICTIONARY_SIZE
15	TransactionHistory_DST	14	68971	COMPRESSED	BULKLOAD	DICTIONARY_SIZE
16	TransactionHistory_DST	15	14011	TOMBSTONE	NULL	NULL
17	TransactionHistory_DST	16	14011	COMPRESSED	REORG_FORCED	REORG

4.) We can also review the **'Tuple_Mover_Xe' file** and ensure that RGs did not qualify for MERGE:

	name	timestamp
▶	columnstore_no_rowgroup_qualified_for_merge	2018-07-15 2

Event: columnstore_no_rowgroup_qualified_for_merge (2018-07-15 21:16:59.9742651)

Details

Field	Value
database_id	5
index_id	1
is_in_memory	0
object_id	308196148
partition_number	1