**Problem**

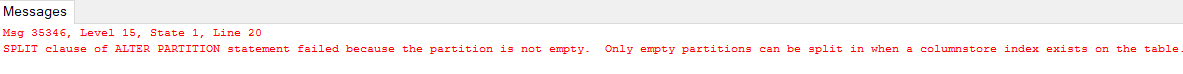
In SQL Server, partitioning allows large tables to be divided into smaller, more manageable parts called partitions. These partitions can be based on specific ranges of data, such as dates. However, one common limitation arises when you cannot extend the last partition of a partitioned table because it contains data, and that table has a CLUSTERED COLUMNSTORE INDEX (CCI). The presence of a columnstore index on partitioned tables can complicate the process of adding new partitions, as columnstore indexes require special handling when modifying partition boundaries.

If your partitioned table does not have any columnstore indexes, splitting an non-empty partition is possible but it will trigger physical data movement. If your table is large and SQL will have to deal with millions of rows then it can be time and resources consuming.

So lets assume you manage time series data and have number of objects across number of databases which are partioned by date or datetime type columns. When trying to create new partitions for upcoming year you get an error saying this:

Msg 35346, Level 15, State 1, Line 20

SPLIT clause of ALTER PARTITION statement failed because the partition is not empty. Only empty partitions can be split in when a columnstore index exists on the table. Consider an ALTER TABLE SWITCH operation from one of the nonempty partitions on the source table to a temporary staging table and then re-attempt the ALTER PARTITION SPLIT operation. Once completed, use ALTER TABLE SWITCH to move the staging table partition back to the original source table.



**Solution**

If you face the scenario where the last partition cannot be extended due to its non-empty status and the presence of a columnstore index, here are potential workarounds:

1. **Switch data out of the last partition**: You can temporarily move data out of the last partition by using the ALTER PARTITION or MERGE PARTITION commands to remove rows from the last partition.
2. **Recreate the columnstore index**: In some cases, you may need to drop and recreate the clustered columnstore index once partition split operation is completed. Again this opration may take quite a bit of time.

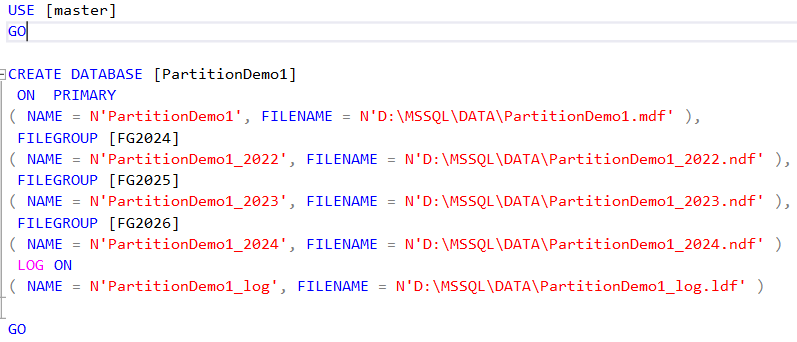
Lets focus on first option. Switching partition in or out works instantly as SQL server is not moving any data – it is just changing meta data and reassigns partition from one table to another. This seems quiet easy to do but you have to remember that following requirments have to be met for ALTER TABLE SWITCH to work:

* The source and target tables (or partitions) must have *identical* columns, indexes and use the same partition column
* The source and target tables (or partitions) must exist on the *same filegroup*
* The target table (or partition) must be *empty*

We can achieve all that using SQL Management Studio and script out objects we need but… it can be a challenge when you have considerable amount of objects and/or databases on your list.

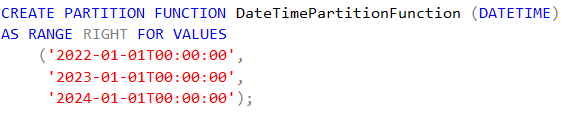
To try and automate the whole task lets create a database with partition scheme and function, partitioned table and some sample data.

**Step 1: Create database**



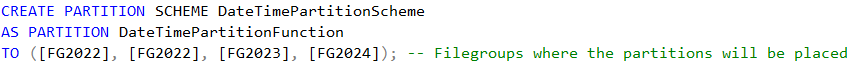
**Step 2: Create the Partition Function**

The partition function divides the data into ranges based on the DATETIME column. This function helps define how the data is distributed across partitions.



**Step 3: Create the Partition Scheme**

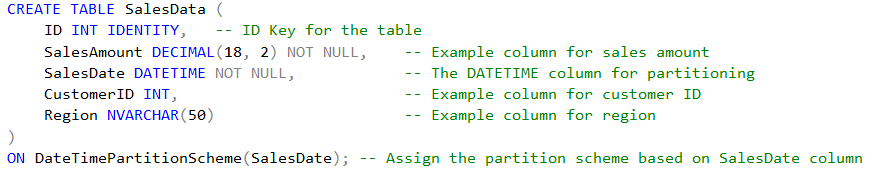
The partition scheme maps the partitions to specific filegroups, ensuring that the partitioned data is physically stored in the correct locations.



Filegroup FG2022 will contains all data from year 2022 and before hence why it appears twice.

**Step 4: Create the Table**

Next, we create a partitioned table SalesData with multiple columns, including a DATETIME column, which will be used for partitioning. The SalesDate column is assigned to the partition scheme.



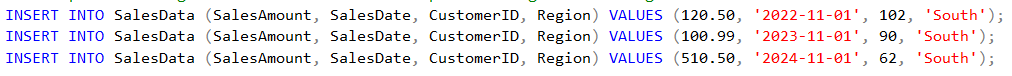
**Step 5: Create Clustered Columnstore Index on the Table**

Once the table is created, we define a CLUSTERED COLUMNSTORE INDEX on it. This index is efficient for large-scale data warehousing but requires special handling during partition operations.

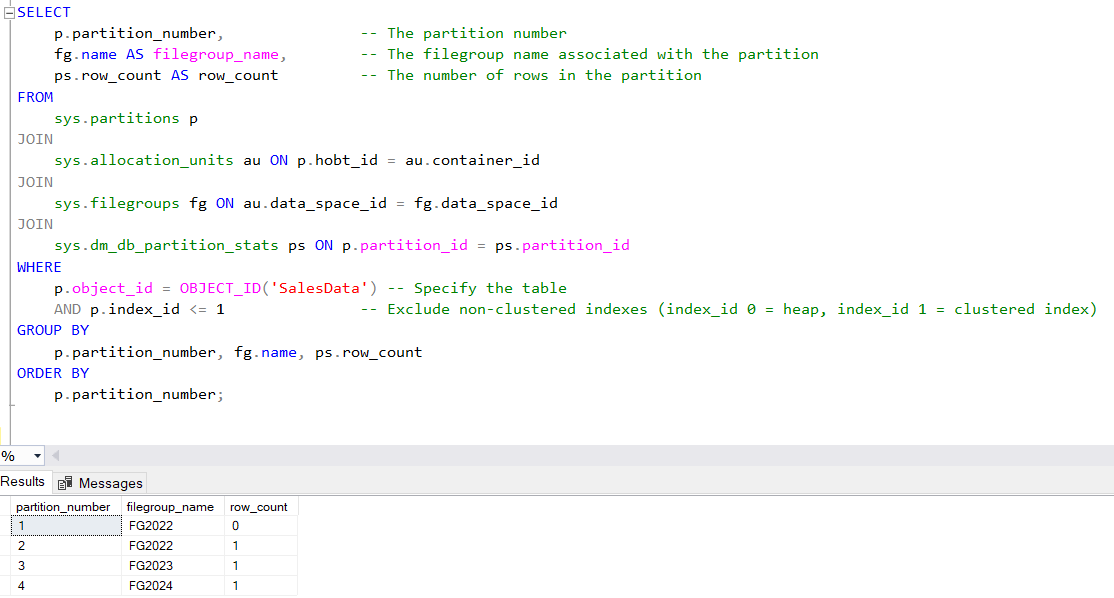


**Step 5: Insert Sample Data to Test the Partitioning and Indexing**

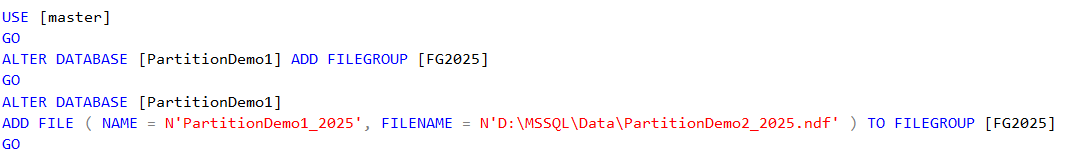
To test the partitioning and the columnstore index, we insert a 3 sample rows of data into the SalesData table. The data will automatically be placed in the appropriate partition based on the SalesDate column.



As result of the above insert statements we should end up with having one row in each file group. Use below query to check:



Now we need to create new file group and file for 2025 data:



Once added we wnat to ALTER our partition scheme and function:



And this is where we run into a problem mentioned at the very begining:



Here are main steps we need to take to workaround the problem without causing lenghty outage for our database:

1. **Create a temporary copy of the table** with the same structure.
2. **Add the necessary constraints** to the temporary table (including partitioning constraints).
3. **Switch the last partition** from the original table to the temporary table.
4. **Update the partition function** to split the partition at the specified boundary (@SplitDate).
5. **Switch the data back** from the temporary table to the original table after the partition boundaries have been adjusted.

Following code will generate a ready script which we can use to perform above steps. It will take care of all indexes and constraints. It also includes error handling so we don’t end up with our data switched to temp table, being unable to switch it back.

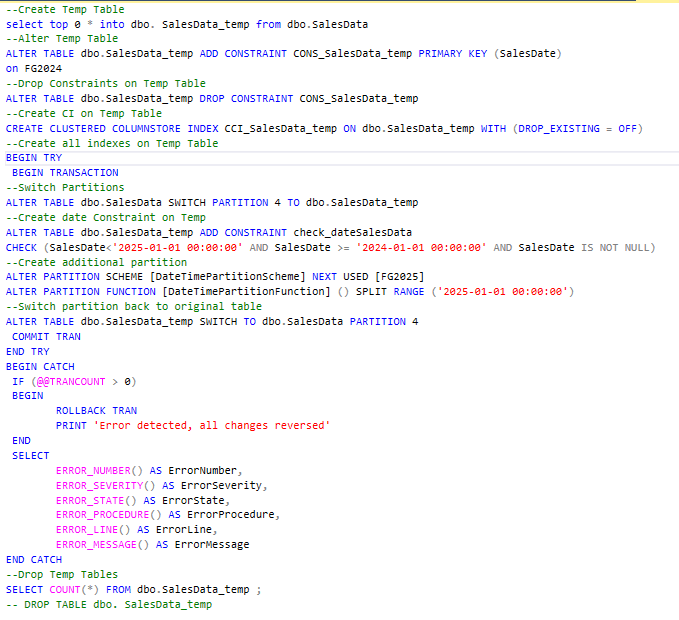
<https://github.com/mipardej/SQLPartitions/blob/main/SwitchPartitions.sql>

Script needs two parameters to be provided:



In our case we are create new partition for 2025 (for whole year = 12 months) so the split date is 1st Jan 2025.

For our demo database, code should generate following output:



**Code Breakdown**

Here is how your script executes the tasks required to move the data out of the last partition and prepare for a partition switch:

**1. Create Temporary Copy of the Table**

You create a temporary copy of the partitioned table. This temporary table has the same structure, including the necessary indexes, primary keys, and constraints. The CreateCopyTable part of your script generates the statement to create this temporary table with a SELECT TOP 0 approach:

This ensures that the temporary table is an empty copy of the original table.

**2. Add Primary Key and Constraints**

You add the necessary constraints to the temporary table. Specifically, you add the primary key and ensure that data inserted into the temporary table aligns with the partitioning rules (data date range). The AlterCopyTable part handles adding constraints to the temporary table:

Additionally, a constraint is added to ensure that only data within the desired date range is inserted into the temporary table

**3. Switch the Last Partition to the Temporary Table**

The **SWITCH** operation is the core of moving data between partitions. This is executed by the SwitchPartition line, which uses the ALTER TABLE SWITCH command to move data from the last partition of the original table to the temporary table

The partition number is dynamically identified based on the last partition, and the data from this partition is moved to the temporary table.

**4. Modify the Partition Scheme**

Next, you modify the partition scheme to accommodate the new partitions. The AlterPartition command prepares the partition scheme for adding a new partition by indicating the filegroup where the new partition will reside:

Then, the partition function is updated to split the range at the @SplitDate value:

**5. Switch Data Back to the Original Table**

After the partition has been split, the data can be switched back into the original table. This is done using the SwitchBack command, which moves the data from the temporary table back to the partitioned table.

This operation effectively places the data back into the partitioned table, following the new partition scheme.

**6. Drop Temporary Table**

Finally, the temporary table is dropped, as it is no longer needed

**Conclusion**

This script automates the process of switching data out of the last partition and modifying the partition scheme and partition function in a SQL Server environment. The steps ensure that partitions can be split dynamically without losing data and that the clustered columnstore index is handled properly. The use of temporary tables and partitioning constraints makes sure that the data remains consistent during the switch operation.

You can execute the script to generate the prepared code for all affected tables and then execute those statements to perform the partition switching operation.