Implement Partitioning Solutions

Section Objective:

The objective may include but is not limited to: partitioned tables and indexes (constraints, partition functions, partition schemes, MERGE, SPLIT, SWITCH); distributed partitioned views (constraints, linked servers)

Database Partitioning Definition:

A partition is a division of a logical database or its constituting elements into distinct independent parts. Database partitioning is normally done for manageability, performance or availability reasons.

A popular and favourable application of partitioning is in a distributed database management system. Each partition may be spread over multiple nodes, and users at the node can perform local transactions on the partition. This increases performance for sites that have regular transactions involving certain views of data, whilst maintaining availability and security.

The partitioning can be done by either building separate smaller databases (each with its own tables, indices, and transaction logs), or by splitting selected elements, for example just one table.

Horizontal partitioning (also see shard) involves putting different rows into different tables. Perhaps customers with ZIP codes less than 50000 are stored in CustomersEast, while customers with ZIP codes greater than or equal to 50000 are stored in CustomersWest. The two partition tables are then CustomersEast and CustomersWest, while a view with a union might be created over both of them to provide a complete view of all customers.

Vertical partitioning involves creating tables with fewer columns and using additional tables to store the remaining columns. Normalization also involves this splitting of columns across tables, but vertical partitioning goes beyond that and partitions columns even when already normalized. Different physical storage might be used to realize vertical partitioning as well; storing infrequently used or very wide columns on a different device, for example, is a method of vertical partitioning. Done explicitly or implicitly, this type of partitioning is called "row splitting" (the row is split by its columns). A common form of vertical partitioning is to split (slow to find) dynamic data from (fast to find) static data in a table where the dynamic data is not used as often as the static. Creating a view across the two newly created tables restores the original table with a performance penalty, however performance will increase when accessing the static data e.g. for statistical analysis.

Table Partitioning

```
USE Master;
```

```
GO

SET NOCOUNT ON
GO

--- Step 1 : Create New Test Database with two different filegroups.
IF EXISTS (SELECT name
             FROM sys.databases
            WHERE name = N'TestDB')
  DROP DATABASE TestDB;
GO

CREATE DATABASE TestDB
ON PRIMARY
  ( NAME='TestDB_Part1'
  , FILENAME='C:\Data\First\TestDB_Part1.mdf'
  , SIZE=2
  , MAXSIZE=100
  , FILEGROWTH=1 )
  , FILEGROUP TestDB_Part2
      ( NAME = 'TestDB_Part2'
      , FILENAME = 'C:\Data\Second\TestDB_Part2.ndf'
      , SIZE = 2
      , MAXSIZE=100
      , FILEGROWTH=1 )
  , FILEGROUP TestDB_Part3
      ( NAME = 'TestDB_Part3'
      , FILENAME = 'C:\Data\Third\TestDB_Part3.ndf'
      , SIZE = 2
      , MAXSIZE=100
      , FILEGROWTH=1 )
  , FILEGROUP TestDB_Part4
      ( NAME = 'TestDB_Part4'
      , FILENAME = 'C:\Data\Forth\TestDB_Part4.ndf'
      , SIZE = 2
      , MAXSIZE=100
      , FILEGROWTH=1 )      ;
GO

USE TestDB;
GO

--- Step 2 : Create Partition Range Function
--  Ranges: -200 to 100, 101 to 200, 201 to 300, 301 to 400
CREATE PARTITION FUNCTION TestDB_PartitionRange (INT)
AS RANGE RIGHT FOR VALUES (0, 200, 300);
GO

--- Step 3 : Attach Partition Scheme to FileGroups
CREATE PARTITION SCHEME TestDB_PartitionScheme
AS PARTITION TestDB_PartitionRange
TO ([PRIMARY], TestDB_Part2, TestDB_Part3, TestDB_Part4);
GO

--- Step 4 : Create Table with Partition Key and Partition Scheme
CREATE TABLE TestTable
(
```

```
    ID        INT          NOT NULL
  , CreatedOn DATETIME
)
ON TestDB_PartitionScheme (ID);
GO

--- Step 5 : (Optional/Recommended) Create Index on Partitioned Table
CREATE UNIQUE CLUSTERED INDEX IX_TestTable
  ON TestTable(ID)
  ON TestDB_PartitionScheme (ID);
GO

--- Step 6 : Insert Data in Partitioned Table
DECLARE @Count INT = -200;
WHILE @Count < 500
  BEGIN
   INSERT INTO TestTable (ID, CreatedOn) VALUES (@Count,GETDATE());

   SET @Count = @Count + 1
  END

--- Step 7 : Verify Rows Inserted in Partitions
SELECT Object_ID
     , Partition_number
     , rows
  FROM sys.partitions
 WHERE OBJECT_NAME(OBJECT_ID)='TestTable';
GO

--- Step 8 : Test Data from TestTable
--SELECT *
--  FROM TestTable;
--GO

SET NOCOUNT OFF
GO
```

ID Range: -200 to 500

Results Partition Function(Right) :

```
Object_ID   Partition_number rows
----------- ---------------- -----
2105058535  1                200
2105058535  2                200
2105058535  3                100
2105058535  4                201
```

Results Partition Function(LEFT) :

```
Object_ID   Partition_number rows
----------- ---------------- -----
2105058535  1                201
2105058535  2                200
2105058535  3                100
2105058535  4                200
```

Split Partitioning:

Append this code to the above code

```
ALTER DATABASE [TestDB] ADD FILEGROUP [TESTDB_PART5]
GO

ALTER DATABASE [TestDB]
ADD FILE
       (NAME = 'TestDB_Part5',
        FILENAME =
         'C:\Data\fifth\TestDB_Part5.ndf',
        SIZE = 5MB,
        MAXSIZE=500,
        FILEGROWTH=1 )
TO FILEGROUP [TestDB_Part5]
Go

ALTER PARTITION SCHEME TestDB_PartitionScheme
NEXT USED [TestDB_Part5]
GO

-- Try to split the last partition
ALTER PARTITION FUNCTION TestDB_PartitionRange ()
SPLIT RANGE (400);
GO

SET NOCOUNT OFF
GO

--- Step 7 : Verify Rows Inserted in Partitions
SELECT Object_ID
     , Partition_number
     , rows
  FROM sys.partitions
 WHERE OBJECT_NAME(OBJECT_ID)='TestTable';
GO
```

Results:

```
Object_ID   Partition_number rows
----------- --------------- ----
2105058535  1               201
2105058535  2               200
2105058535  3               100
2105058535  5               100
2105058535  4               100
```

Merge Partitioning:

Append this code to the above code

```
ALTER PARTITION FUNCTION TestDB_PartitionRange()
MERGE RANGE(0)
```

```
GO

--- Step 7 : Verify Rows Inserted in Partitions
SELECT Object_ID
     , Partition_number
     , rows
  FROM sys.partitions
 WHERE OBJECT_NAME(OBJECT_ID)='TestTable';
GO
```

Results:

```
Object_ID   Partition_number rows
----------- ---------------- ----
2105058535  1                401
2105058535  2                100
2105058535  4                100
2105058535  3                100
```