University of Windsor
Faculty of Science

COMP 8157 Advanced Database Topics
University of Windsor, School of Computer Science
Lab 2
Weight: 3.75 %

Aim: This assignment will assess your understanding of partitioning in SQL Server.

Instructions:

1. Construct SQL Queries for each of the steps on Page 6.
2. For each step, you are required to submit an SQL query, along with the screenshot.
3. Please submit .docx version along with its .pdf version of the updated file. A report to show the code and screenshots of all solutions.
4. Submit the lab during the lab session or maximum by tomorrow [11:59pm].

Part 1: Vertical Partitioning:

1. Create Required Table

```
CREATE TABLE EmployeeReports
(
ReportID int IDENTITY (1,1) NOT NULL,
ReportName varchar (100),
ReportNumber varchar (20),
ReportDescription varchar (max)
CONSTRAINT EReport_PK PRIMARY KEY CLUSTERED (ReportID)
)
```

2. Insert Required Data

```
DECLARE @i int
SET @i = 1
BEGIN TRAN
WHILE @i<=100000
BEGIN
INSERT INTO EmployeeReports(ReportName,ReportNumber,ReportDescription)
VALUES('ReportName',CONVERT (varchar (20), @i),REPLICATE ('Report', 1000))
SET @i=@i+1
END
COMMIT TRAN
GO
```

3. Check query performance before partitioning

```
SET STATISTICS IO ON
```

```
SET STATISTICS TIME ON
SELECT er.ReportID, er.ReportName, er.ReportNumber
FROM dbo.EmployeeReports er
WHERE er.ReportNumber LIKE '%33%'
SET STATISTICS IO OFF
SET STATISTICS TIME OFF
```

4. Divide EmployeeReports into two tables

```
-------------------------------------------------
CREATE TABLE ReportsDesc
( ReportID int FOREIGN KEY REFERENCES EmployeeReports (ReportID),
ReportDescription varchar(max)
CONSTRAINT PK_ReportDesc PRIMARY KEY CLUSTERED (ReportID)
)
-----------------------------------------------------------
CREATE TABLE ReportsData
(
ReportID int NOT NULL,
ReportName varchar (100),
ReportNumber varchar (20),
CONSTRAINT DReport_PK PRIMARY KEY CLUSTERED (ReportID)
)
```

---

5. Insert data into ReportsData table

```
INSERT INTO dbo.ReportsData(ReportID,ReportName,ReportNumber)
SELECT er.ReportID, er.ReportName, er.ReportNumber FROM dbo.EmployeeReports
er
```

6. Check query performance on new table

```
SET STATISTICS IO ON
SET STATISTICS TIME ON
SELECT er.ReportID, er.ReportName, er.ReportNumber
FROM ReportsData er
WHERE er.ReportNumber LIKE '%33%'
SET STATISTICS IO OFF
SET STATISTICS TIME OFF
```

Part 2: Horizontal Partitioning:

When data tables get very large with records numbering in the millions, it might be beneficial to consider creating partition tables. Partition tables are tables that have been split up horizontally so that collections of records can be spread out across multiple hard drives.

> This is accomplished by leveraging SQL Server filegroups, which serve as the physical location or containers that store the database objects. → spreading a single table across several filegroups, that, themselves, are stored on separate hard drives.

There are four steps to creating a partitioned table.

1. create a filegroup or filegroups within the database that'll hold the partitions.
2. create a partition function that divides the rows of a table into partitions based on the values of a specified column.
3. create a partition scheme that maps the partitions of a table to the new filegroups.
4. create or modify a table and specify the partition scheme as the storage location to segment the data out and store it within the appropriate filegroup.

Step 1: create a filegroup or filegroups within the database.

Create filegroups using SQL query

```
ALTER DATABASE PartitionTables ADD FILEGROUP
FG_2015; ALTER DATABASE PartitionTables ADD
FILEGROUP FG_2016; ALTER DATABASE
PartitionTables ADD FILEGROUP FG_2017; ALTER
DATABASE PartitionTables ADD FILEGROUP
FG_2018;
```

Assign the hardware location where each filegroup goes.

***Note that you should use the path of your local drive.

```
ALTER DATABASE [PartitionTables] ADD FILE (NAME = [Part2015],
FILENAME = 'C:\Program Files\Microsoft SQL
Server\MSSQL15.MSSQLSERVER\MSSQL\DATA\PartitionTables1.ndf', SIZE = 3072
KB, MAXSIZE = UNLIMITED, FILEGROWTH = 1024 KB
) TO FILEGROUP [FG_2015]

ALTER DATABASE [PartitionTables] ADD FILE (NAME = [Part2016],
FILENAME = 'C:\Program Files\Microsoft SQL
Server\MSSQL15.MSSQLSERVER\MSSQL\DATA\PartitionTables2.ndf', SIZE = 3072
KB, MAXSIZE = UNLIMITED, FILEGROWTH = 1024 KB
) TO FILEGROUP [FG_2016]

ALTER DATABASE [PartitionTables] ADD FILE (NAME = [Part2017],
FILENAME = 'C:\Program Files\Microsoft SQL
Server\MSSQL15.MSSQLSERVER\MSSQL\DATA\PartitionTables3.ndf', SIZE = 3072
KB, MAXSIZE = UNLIMITED, FILEGROWTH = 1024 KB
) TO FILEGROUP [FG_2017]

ALTER DATABASE [PartitionTables] ADD FILE (NAME = [Part2018],
FILENAME = 'C:\Program Files\Microsoft SQL
Server\MSSQL15.MSSQLSERVER\MSSQL\DATA\PartitionTables4.ndf', SIZE = 3072
```

```
KB, MAXSIZE = UNLIMITED, FILEGROWTH = 1024 KB
) TO FILEGROUP [FG_2018]
```

Step 2: create a partition function.

```
CREATE PARTITION FUNCTION PFYears (smallint)
AS RANGE LEFT
FOR VALUES (2015, 2016, 2017);
GO
```

The partition function will define how we want to divide up our data. In this example, we're going to store order records based on the year, and each year should sort out into its own table partition. We create a partition function called **PFYears**, and we'll feed it a small integer data type.

The `RANGE LEFT` line, indicates where records with exactly the boundary value will fall. In this case, records that are exactly 2015 will fall to the left-hand range, 2015 and earlier. Records that are exactly 2016, will fall into the 2015 to 2016 range and so on.

The other option that we could specify is `RANGE RIGHT`, which would be more appropriate for segmenting data that includes dates down to the day. For instance, dates that were exactly January 1, 2016, would fall into the right-hand group, between 2016 and 2017.

Step 3: create a partition scheme.

```
CREATE PARTITION SCHEME PSYears
AS PARTITION PFYears
TO (FG_2015, FG_2016, FG_2017, FG_2018);
GO
```

The partition scheme will define the filegroups each partition will get saved into. The partition scheme simply lifts out the filegroups in the same order as our partition function up above. The scheme name is going to be PSYears, and it's going to process the data coming out of the partition function called PFYears. Anything in this first range, up to 2015, will get put into filegroup_2015. Anything in the second group, which was 2016, will get put into the filegroup FG_2016. Anything in the range of 2017 will go into this filegroup. And anything after that will go into the filegroup FG_2018.Step 4: create or modify a table and specify the partition scheme as the storage location to segment the data out and store it within the appropriate filegroup.

```
CREATE TABLE dbo.Orders(
    OrderYear smallint NOT NULL,
    OrderID int IDENTITY(1,1) NOT NULL,
    PRIMARY KEY (OrderYear, OrderID))
    ON PSYears(OrderYear);
GO
```

`ON PSYears(OrderYear)-` this line creates the partition table. In this case, we're specifying that we want to create this table on a specific location, and the location is going to be the name of the partition scheme, PSYears. In order to properly divide up our data, we'll feed it the function parameter, OrderYear.

- Let's insert some values into table

```
INSERT dbo.Orders
    VALUES  (2015),
```

```
                (2016),
                (2017),
                (2018);
GO
```

- Look at the data

```
SELECT * from dbo.Orders;
GO
```

- Let's see how the records are getting stored on the hard drive. So, view the number of records in each partition using a system function.

```
SELECT $PARTITION.PFYears(OrderYear) AS Partition,
    COUNT(*) AS [COUNT] FROM dbo.Orders
    GROUP BY $PARTITION.PFYears(OrderYear)
    ORDER BY Partition;
GO
```

`$PARTITION` is a built-in system function, we'll use this function to process our PFYear's partition function.

Can you explain the meaning of the output?

Write a query to view the records in partition number 4?

```
SELECT *
FROM dbo.Orders
WHERE $PARTITION.PFYears(OrderYear) = 4;
```

Assignment

1. Create a database <yourfirstname>ADTlab2
2. Create filegroups within the database to divide them by month.
3. Create a partition function <yourfirstname>ByMonth (Note: consider the datatype of the month to be integer)
4. Create a partition scheme <yourfirstname>ByMonthADT
5. Create a table with following information and specify the partition scheme as the storage location to segment the data out and store it within the appropriate filegroup.

```
ReportID int IDENTITY (1,1) NOT NULL,
ReportMonth int NOT NULL,
PRIMARY KEY (ReportID, ReportMonth),
MonthlyReport varchar(max))
```

6. Insert 100000 records randomly to the table above.

   Hint: you can generate a loop to run 100000 times. Use the incremental value as ReportID, generate random number between 1 and 12, and finally replicate your first name for 100 times to generate MonthlyReport.

7. Write a query to check the number of records in each partition.
8. Execute the records in partition number 5.
