Indexing Workshop

1. **Create a table**. Let us create an Employee table
   - EmployeeID INT (Primary Key)
   - EmployeeName VARCHAR(100)
   - Salary INT
   - Gender VARCHAR(6)
   - City VARCHAR(50)

2. **Create a Clustered Index**
   a. In order to check the list of indexes on table we can use the statement in SQL Server:
   ```
   EXEC sp_helpindex Employee.
   ```
   What do you observe?
   b. Now let us insert some unsorted rows (with respect to primary key) by executing the queries mentioned below.

   ```
   INSERT INTO Employee VALUES (3, 'John', 4500, 'Male', 'New York')
   INSERT INTO Employee VALUES (1, 'Sam', 2500, 'Male', 'London')
   INSERT INTO Employee VALUES (4, 'Sara', 5500, 'Female', 'Tokyo')
   INSERT INTO Employee VALUES (5, 'Todd', 3100, 'Male', 'Toronto')
   INSERT INTO Employee VALUES (2, 'Pam', 6500, 'Female', 'Sydney')
   ```

   c. Select the table to see all records. What do you observe with the Primary Key column?

   d. Can we create clustered index on multiple columns? Let us create another clustered index on Salary column. Note down the observations and explain.

3. **Insert data**
   a. Let's run the stored procedure to insert random 100000 rows of data to the Employee

   ```
   CREATE PROCEDURE sp_InsertEmployees
   AS
   BEGIN
       SET NOCOUNT ON;

       DECLARE @Counter INT = 6;

       WHILE @Counter <= 100000
       BEGIN
           DECLARE @EmployeeName VARCHAR(100) = 'Employee ' + CAST(@Counter AS
   VARCHAR(10));
           DECLARE @Salary INT = 50000 + (@Counter % 100) * 1000;
           DECLARE @Gender VARCHAR(6) = CASE WHEN @Counter % 2 = 0 THEN 'Male' ELSE
   'Female' END;
           DECLARE @City VARCHAR(50) = 'City ' + CAST((@Counter % 10) + 1 AS
   VARCHAR(10));

           INSERT INTO dbo.Employee (EmployeeID, EmployeeName, Salary, Gender, City)
           VALUES (@Counter, @EmployeeName, @Salary, @Gender, @City);
   ```

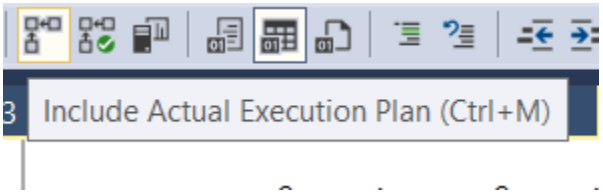
```
        SET @Counter = @Counter + 1;
    END
END;
```

Execute the stored procedure to insert data to the table.

```
EXEC sp_InsertEmployees
```

4. **Performance Statistics**

   One way to understand Query Cost is by enabling the Include Actual Execution Plan in the toolbar.



   We can check performance statistics by enabling the `STATISTICS` for IO and Time before executing the respective query. Let us set Time and IO statistics to check performance of the query for a random value of the City column.

```
SET STATISTICS IO ON; SET STATISTICS TIME ON;
```

5. **Run frequent query**
   Once the statistics are 'ON', Execute the Query to select rows where City = <random_value> and note the performance in Result Set. For example

```
SELECT * FROM Employee WHERE City = 'New York'
```

6. **Create Non-Clustered Index**
   Now we will create an index on the column which is used in the above query.
   And check the performance to see any improvement. What do you observe?

Assignment

Create an Orders table and build a stored procedure to insert 1,00,000 rows to this table.

OrderID: INT
CustomerID: INT
OrderDate: Date

TotalAmount: Decimal (10,2)
Status: VARCHAR(10) Random Values: {"Pending," "Shipped," "Delivered," }

A. Build a stored procedure sp_Insert_Orders_<Your_First_Name> to insert 1,00,000 rows in this table.

B. Execute the below query and check the performance statistics for Time and IO. Make sure to note down the results. Observe and note the type of Operation in Execution Plan.

```
SELECT * FROM Orders WHERE OrderID = 1
```

C. Create a Clustered Index on the OrderID.

D. Execute the query in part (B) and check the performance. Now observe and note the type of Operation in Execution Plan. Compare the performance results prior and post to creating index and discuss if there is any improvement.