Lab: Query Rewriting & Index Opimization

Scenario:

Consider a large e-commerce database with a Orders table containing millions of records. A frequently executed query is:

```sql
SELECT
    CustomerID,
    SUM(TotalAmount) AS TotalSales
FROM
    Orders
WHERE
    OrderDate BETWEEN '2023-01-01' AND '2023-12-31'
GROUP BY
    CustomerID;
```

This query calculates total sales for each customer within a specific year.

Performance Analysis:

Without appropriate indexes, this query might perform slowly due to a full table scan. To improve performance, we can analyze the execution plan and identify potential bottlenecks.

Optimization Steps:

```
Create an Index:
    Create a nonclustered index on the OrderDate column:
```

```sql
CREATE NONCLUSTERED INDEX IX_Orders_OrderDate ON Orders (OrderDate);
```

This index will help the query optimizer efficiently locate rows within the specified date range.

Review Execution Plan:

- Re-run the query and examine the execution plan.
- Look for any full table scans or inefficient join operations.

- If the index is being used effectively, you should see an Index Seek operation on the IX_Orders_OrderDate index.

Consider Data Distribution:

- If the OrderDate column is highly skewed (e.g., many orders in a few specific months), partitioning the table might further improve performance.

Revised Query

```
SELECT
    CustomerID,
    SUM(TotalAmount) AS TotalSales
FROM
    Orders
WHERE
    OrderDate BETWEEN '2023-01-01' AND '2023-12-31'
GROUP BY
    CustomerID;
```

Database Query Performance Analysis & Tuning

Objective

To enhance understanding of query performance and optimization techniques in a database environment.

Tasks

Task 1: Create an empty database

```
CREATE DATABASE TestDB;
GO
```

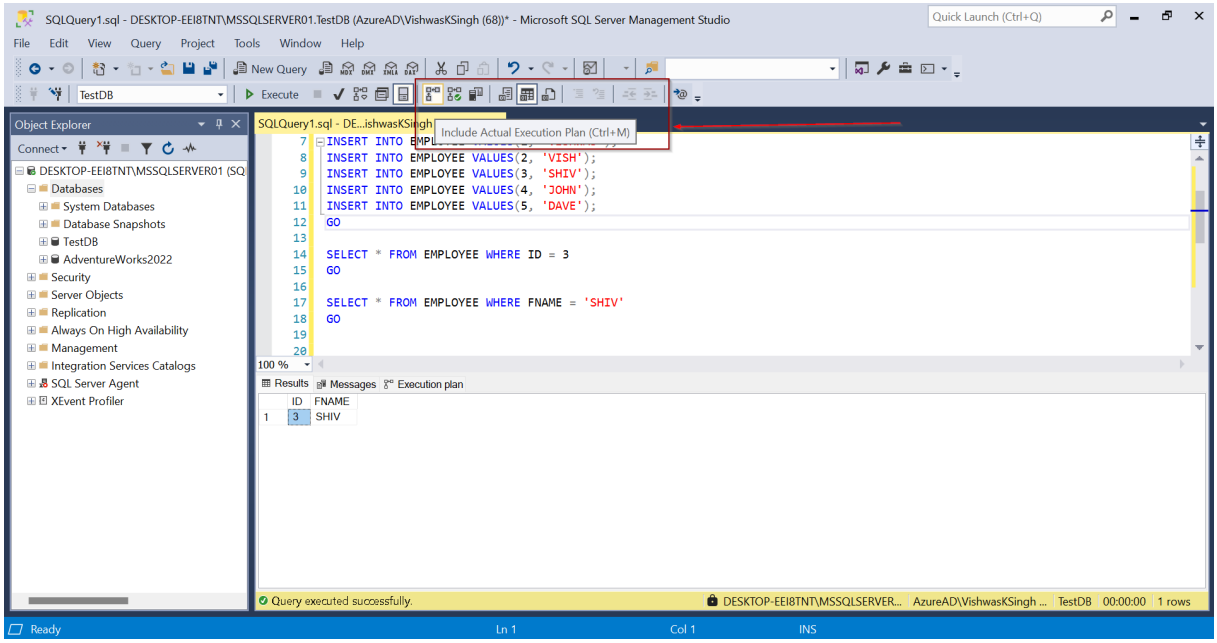
Task 2: Create a table and insert data into it

```
USE TestDB
GO

CREATE TABLE EMPLOYEE(ID INT PRIMARY KEY, FNAME VARCHAR(30))
GO

INSERT INTO EMPLOYEE VALUES(1, 'VISHWAS');
INSERT INTO EMPLOYEE VALUES(2, 'VISH');
INSERT INTO EMPLOYEE VALUES(3, 'SHIV');
INSERT INTO EMPLOYEE VALUES(4, 'JOHN');
INSERT INTO EMPLOYEE VALUES(5, 'DAVE');
GO
```

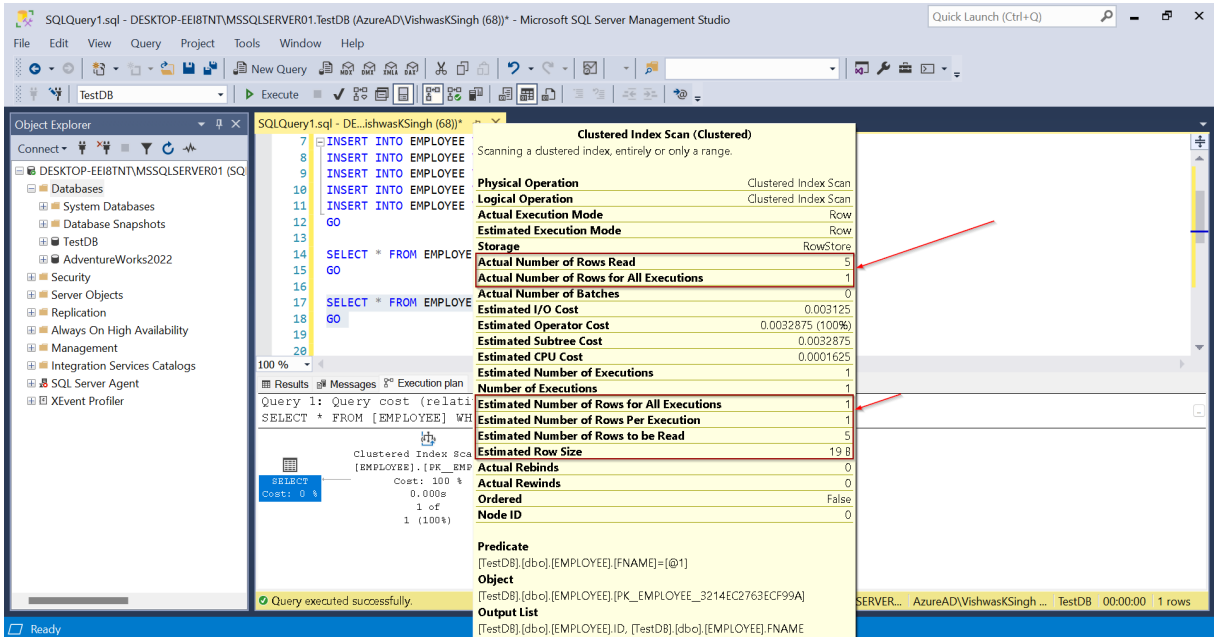
Task 3: Enable Include Actual Execution Plan



Task 4: Execute the Below Query

```
SELECT * FROM EMPLOYEE WHERE FNAME = 'SHIV'
GO
```

Observe the execution Plan



The query searches a record using a non index value or a non primary value. Hence the query searches the whole table. When scaled this is a performance bottleneck.

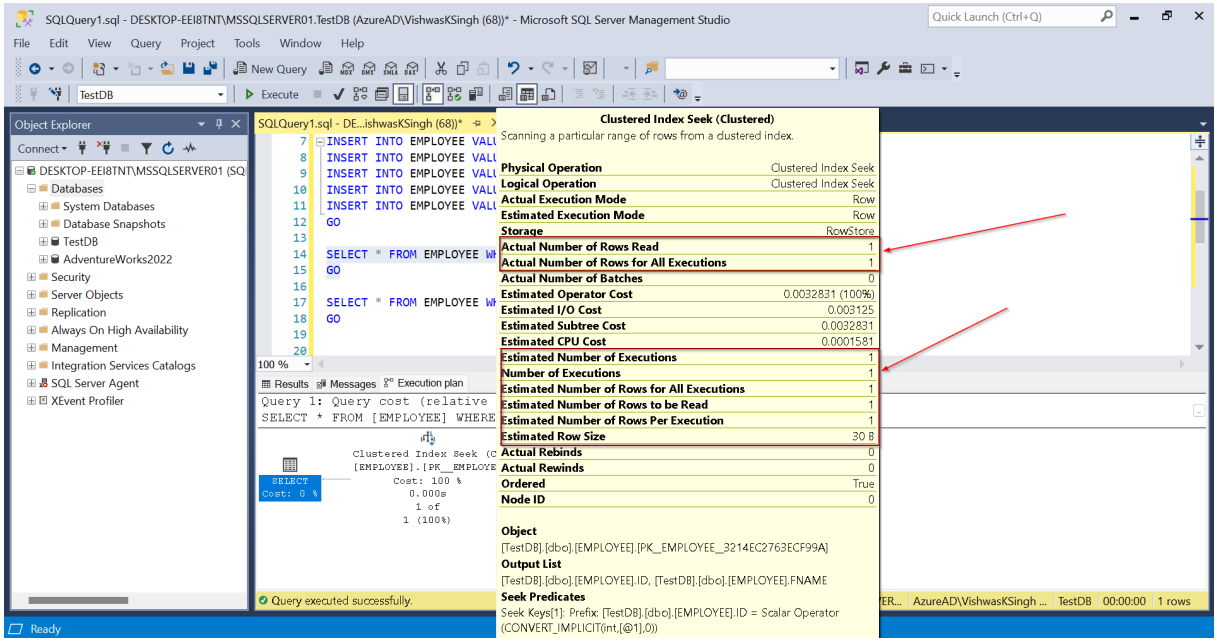
This can be improved by

1. Creating an index over the column
2. using an existing indexed column(Primary Key)

Task 5: Creating an optimized query

```
SELECT * FROM EMPLOYEE WHERE ID = 3
GO
```

Execute the query and observe the execution plan



Hence, Query performance can be improved by rewriting the query using indexed columns

Similarly, We can apply the above steps on Northwind/AdventureWorks Database

We can also observe the query performance using activity monitor