

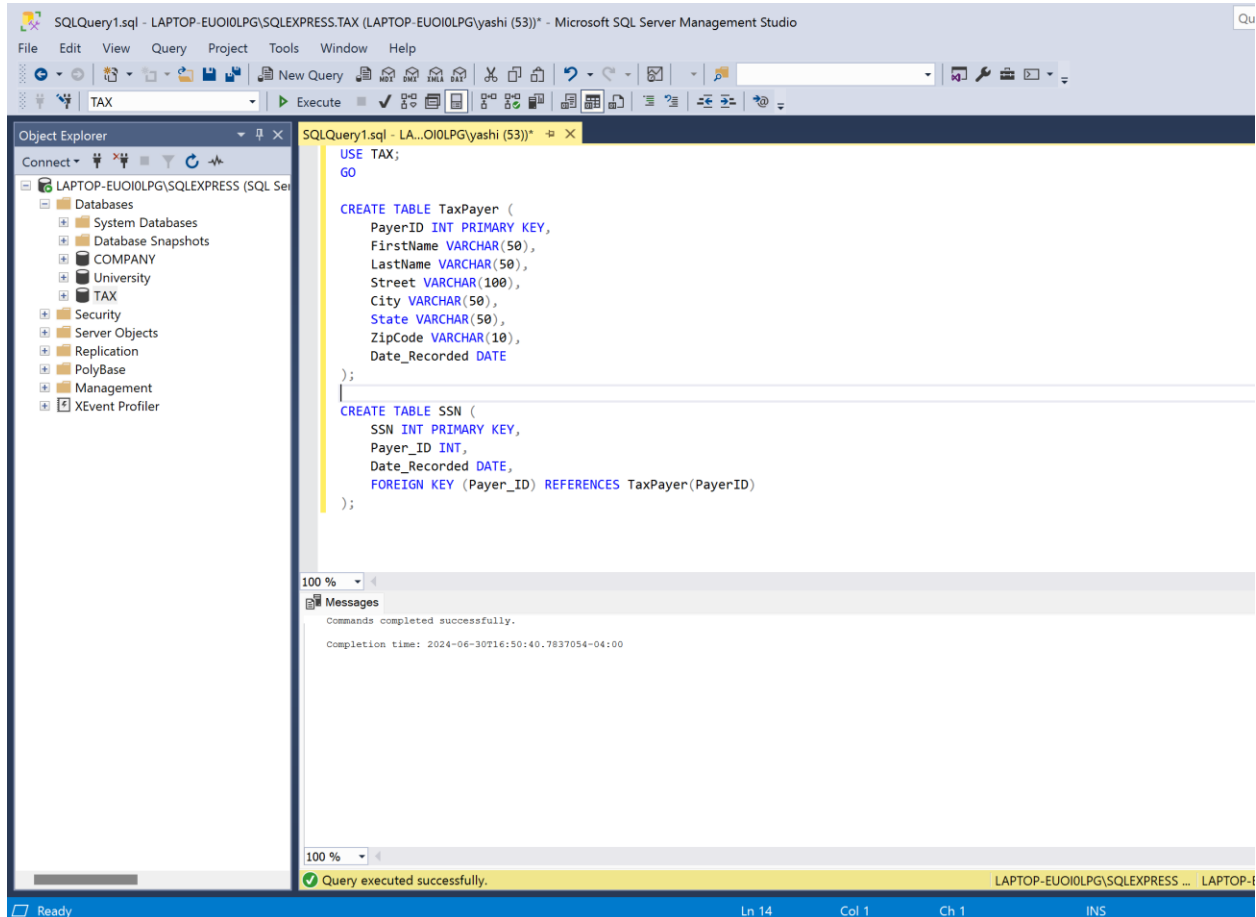
# CIS 552: DATABASE DESIGN

Submitted by – Yashika Patil

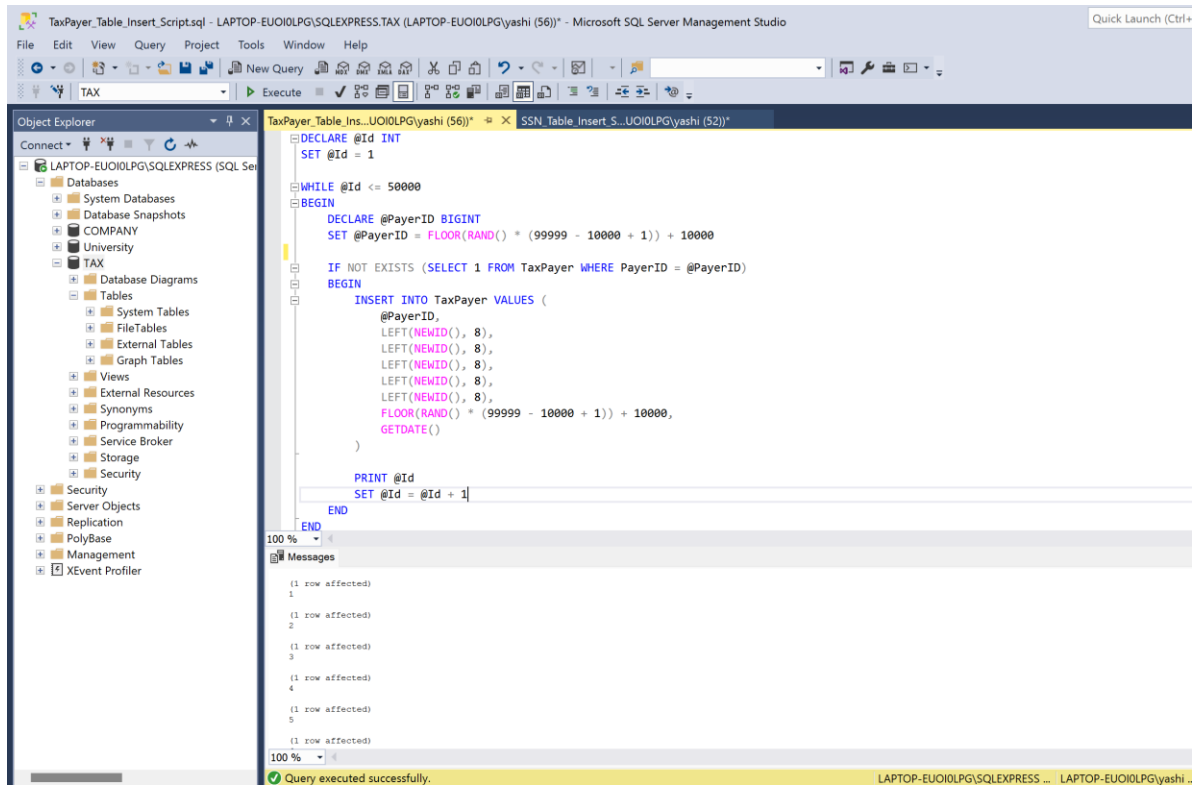
## Lab - 4

### Task 1: Performance Tuning of Queries using Indexes

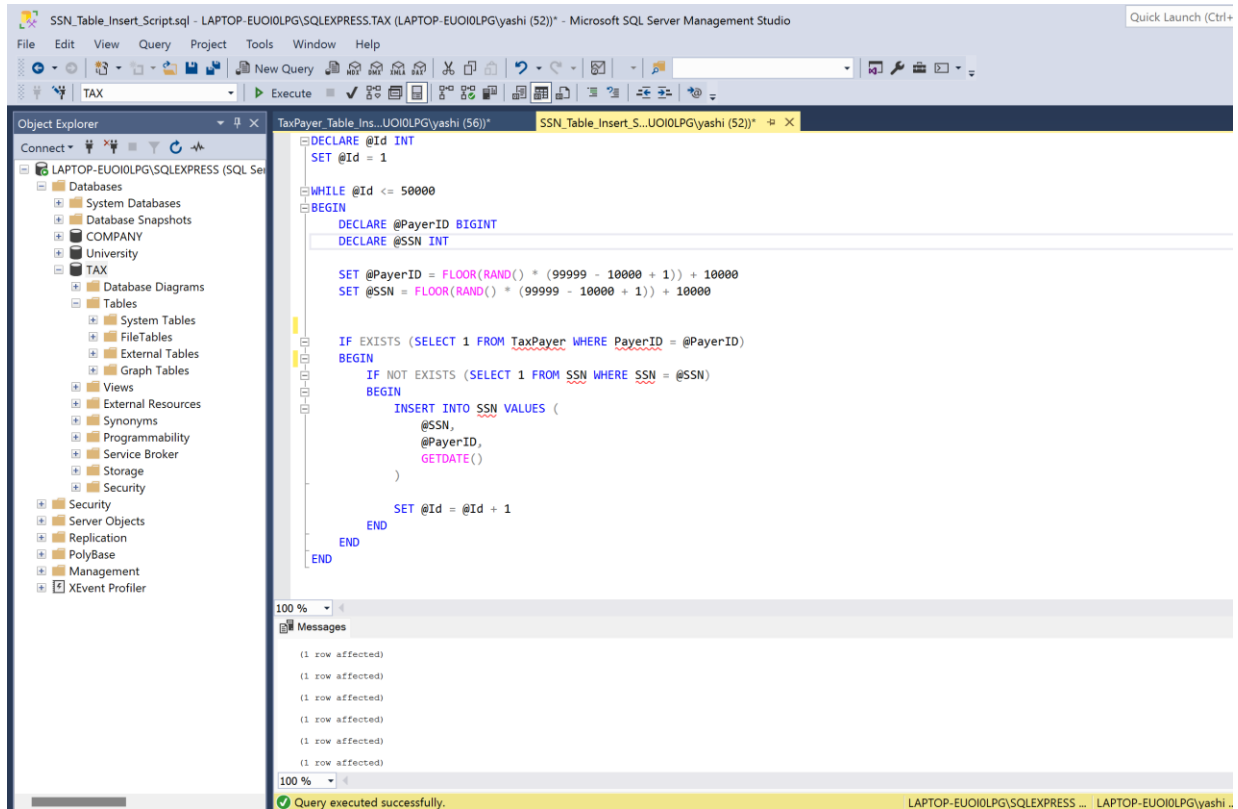
1. **Database and Table Generation:** I generated a database called TAX and created tables according to the provided schema, ensuring all columns and constraints were included.



2. **Data Population:** I populated the tables with 50,000 rows each using the provided SQL queries from the Lab 4 manual, TaxPayer\_Table\_Insert\_Script.sql and SSN\_Table\_Insert\_Script.sql,
  - a. Execution of both the queries: 'TaxPayer\_Table\_Insert\_Script.sql'

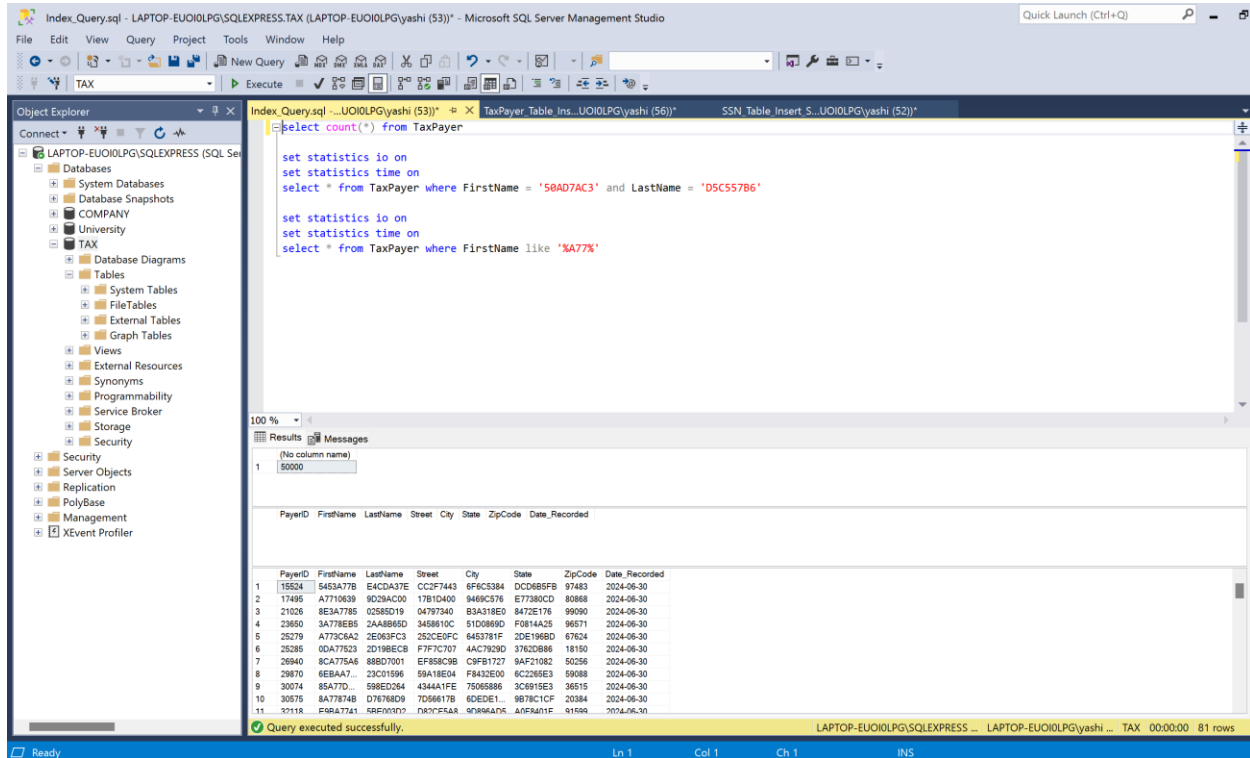


b. Execution of both the queries: 'SSN\_Table\_Insert\_Script.sql'



### 3. Without non-clustered index performance analysis:

I executed the Index\_query.sql script to observe the initial query performance without any indexes.



The screenshot displays the Microsoft SQL Server Enterprise Manager interface. The left pane shows the Object Explorer with the 'TAX' database selected. The right pane shows the 'Index\_Query.sql' script with the following SQL code:

```
select count(*) from TaxPayer

set statistics io on
set statistics time on
select * from TaxPayer where FirstName = 'S0AD7AC3' and LastName = 'D5C557B6'

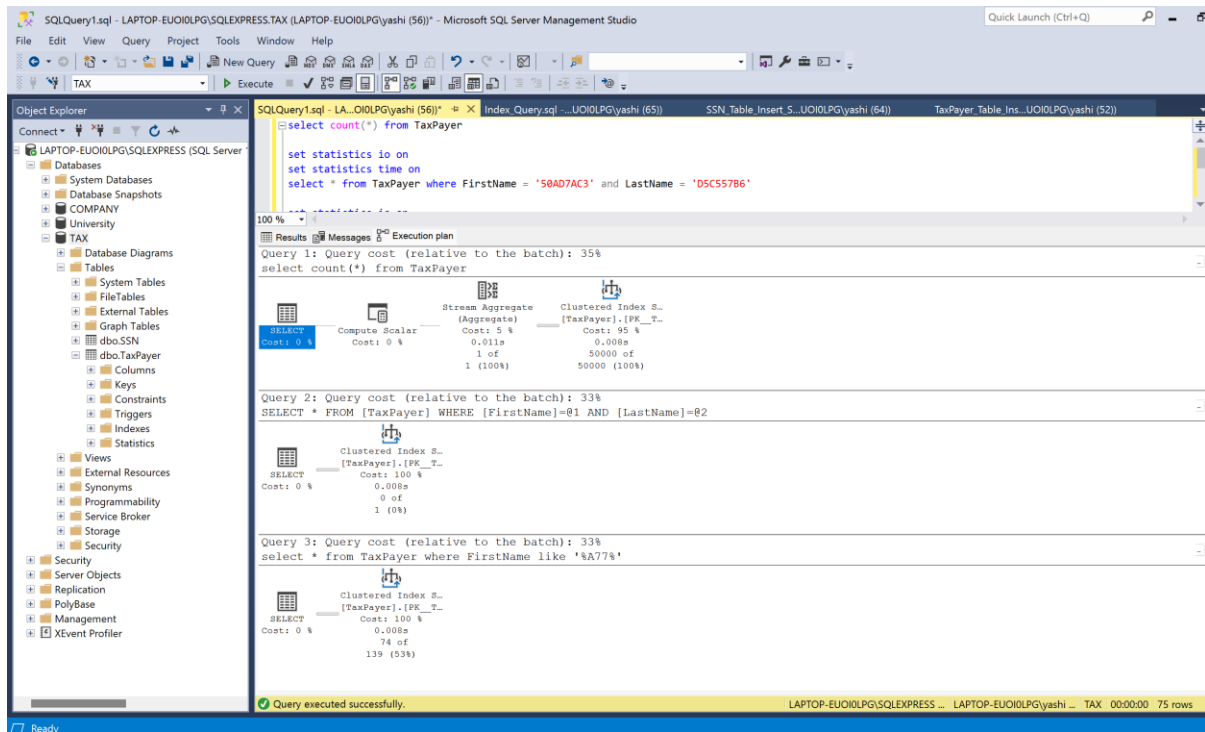
set statistics io on
set statistics time on
select * from TaxPayer where FirstName like 'XA77%'
```

The 'Results' pane shows the execution plan and the query results. The execution plan indicates a full table scan. The query results are as follows:

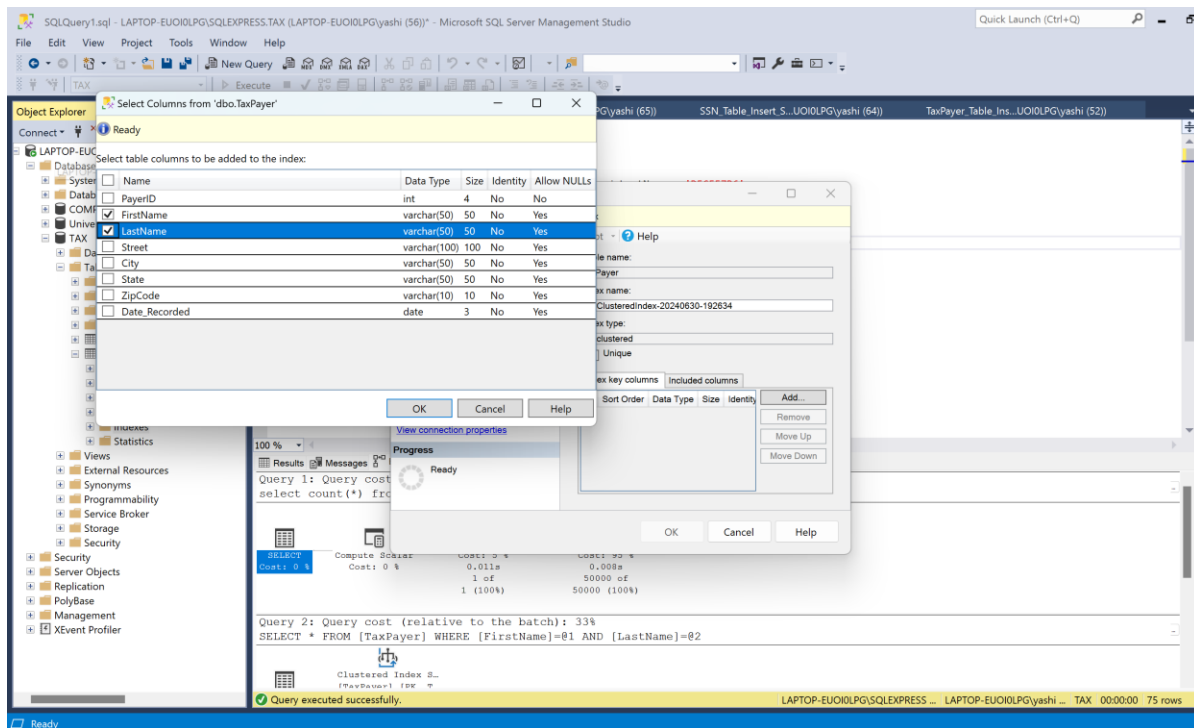
PayerID	FirstName	LastName	Street	City	State	ZipCode	Date_Recorded
1	15534	S4SA77B	E4CD437E	CC3F7443	6F9C3384	DCD685F9	97483
2	17495	A7710639	9029AC00	1781D400	9469C576	E77380CD	80868
3	21026	8E3A7785	02585D19	04797340	B3A318E0	8472E176	99090
4	23650	3A778EB5	2AA8B65D	3458610C	51D0869D	F0814A25	96571
5	25279	A773C6A2	2E063FC3	252CE0FC	6453781F	2DE1968D	67624
6	25285	00A77523	2D198EC8	F7F7C701	4AC7929D	37620B86	18150
7	26940	8CA775A6	886D7001	EF858C98	C9F81727	9AF21082	50256
8	29870	6EBA77D...	23C01596	59A18E04	F8432E00	6C2265E3	59088
9	30074	85A77D...	598ED264	4344A1FE	75065886	3C6915E3	36515
10	30575	8A7787AB	D76786D9	7D566178	6DEDE1...	9878C1CF	20384
11	31116	F56A7741	585D09D1	F8D175A8	6D986A76	A0F5A11F	61566

The status bar at the bottom indicates 'Query executed successfully' and '81 rows'.

**Execution plan without non-clustered:** The execution plan before adding any indexes showed longer execution times and higher resource consumption.



- 4. Creating Non-Clustered Index:** I created a non-clustered index on the FirstName and LastName columns of the TaxPayer table by right-clicking on the 'Indexes' section, selecting 'New index', choosing 'Non-Clustered Index', and adding the specified columns.



After selecting non-clustered index, on refreshing I could see the column names in the indexes as:

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left displays the database structure, including the 'TaxPayer' table and its non-clustered index. The query window in the center contains the following SQL code:

```
select count(*) from TaxPayer
set statistics io on
select * from TaxPayer where FirstName = '50AD7AC3' and LastName = 'DSC557B6'
set statistics io on
set statistics time on
select * from TaxPayer where FirstName like '%A77%'
```

The Results pane at the bottom shows the execution plan for the query. The plan indicates that the non-clustered index is used for the query. The results table shows the following data:

PayerID	FirstName	LastName	Street	City	State	ZipCode	Date_Recorded
1	12960	65AA7709	29FC07F3	AC31BEC0	25F84D47	44EBE510	64584
2	14012	ECAT7320	470376B8	8F8A0A08	E92A1549	A287247C	24743
3	14418	D38B1A77	86473BA8	E5580E0B	CBED8E01	E0948657	95381
4	15513	SC6EA77A	300049EC	00B89E10	55136881	3C6B411C	17513
5	16277	A78B450	8522E734	16241239	C1E0CA7	40EDCB33	73092
6	16849	A7719904	D07382A5	8FPA737	394539FE	12A5AD7B	28119
7	17481	CD4A772	58164564	7ABD9B55	03C5C5DC	AC2F4630	47100
8	19140	D798A774	84B73403	ASDC4A45	F7801E10	C5B6E3DF	22629
9	19334	21A77C7F	41643514	AFBF434E	989242D3	10B79BDA	17458
10	19712	AA7711F7	825E5A84	221413D2	F9AAFA03	82FF7E3A	81175
11	2007X	488F7A77	82F6A4C3	3D1D797C	75676A87	C96C7908	70174

Executed the query again with execution plan.

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left displays the database structure, including the 'TaxPayer' table and its non-clustered index. The query window in the center contains the following SQL code:

```
select count(*) from TaxPayer
set statistics io on
select * from TaxPayer where FirstName = '50AD7AC3' and LastName = 'DSC557B6'
set statistics io on
set statistics time on
select * from TaxPayer where FirstName like '%A77%'
```

The Results pane at the bottom shows the execution plan for the query. The plan indicates that the non-clustered index is used for the query. The results table shows the following data:

PayerID	FirstName	LastName	Street	City	State	ZipCode	Date_Recorded
1	12960	65AA7709	29FC07F3	AC31BEC0	25F84D47	44EBE510	64584
2	14012	ECAT7320	470376B8	8F8A0A08	E92A1549	A287247C	24743
3	14418	D38B1A77	86473BA8	E5580E0B	CBED8E01	E0948657	95381
4	15513	SC6EA77A	300049EC	00B89E10	55136881	3C6B411C	17513
5	16277	A78B450	8522E734	16241239	C1E0CA7	40EDCB33	73092
6	16849	A7719904	D07382A5	8FPA737	394539FE	12A5AD7B	28119
7	17481	CD4A772	58164564	7ABD9B55	03C5C5DC	AC2F4630	47100
8	19140	D798A774	84B73403	ASDC4A45	F7801E10	C5B6E3DF	22629
9	19334	21A77C7F	41643514	AFBF434E	989242D3	10B79BDA	17458
10	19712	AA7711F7	825E5A84	221413D2	F9AAFA03	82FF7E3A	81175
11	2007X	488F7A77	82F6A4C3	3D1D797C	75676A87	C96C7908	70174

This execution plan showed significant improvements in query performance, with reduced logical reads and less execution times.

## Task 2 - Query Optimization:

**Inefficient Query:** By executing the inefficient query from ‘Efficient\_inefficient\_Join\_Example.sql’:

The screenshot displays the Microsoft SQL Server Management Studio interface. The query editor shows the following SQL code:

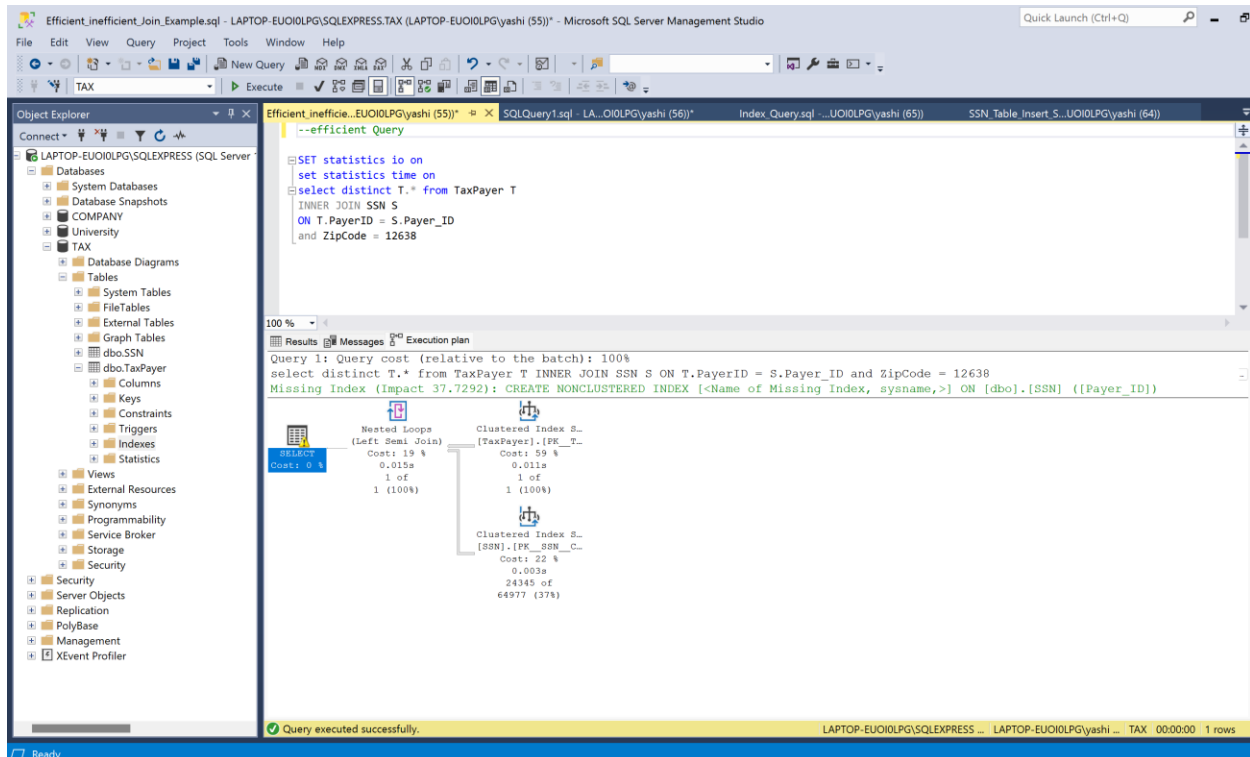
```
--Inefficient Query
SET statistics io on
set statistics time on
SELECT * FROM TaxPayer WHERE PayerID in (select Payer_ID from SSN)
and ZipCode = 12638
```

The execution plan for this query is shown below the code. It indicates a 'Missing Index' with an impact of 37.7292, suggesting the creation of a nonclustered index on the SSN table. The plan also shows a 'Nested Loops (Left Semi Join)' operation with a cost of 19% and 0.012s, and a 'Clustered Index Scan' on the TaxPayer table with a cost of 59% and 0.007s. The SSN table scan has a cost of 22% and 0.003s, reading 24345 of 64977 rows (37%).

Query 1: Query cost (relative to the batch): 100%  
SELECT \* FROM TaxPayer WHERE PayerID in (select Payer\_ID from SSN) and ZipCode = 12638  
Missing Index (Impact 37.7292): CREATE NONCLUSTERED INDEX [<Name of Missing Index, sysname,>] ON [dbo].[SSN] ((Payer\_ID))

The inefficient queries showed higher execution times and more resource usage in all steps.

**Efficient Query:** The efficient queries showed lower execution times and more optimal resource usage.



## Conclusion:

- Understood the importance of database performance tuning and query optimization.
  - Initial queries without indexes showed high resource use and long execution times.
  - Execution plans before indexing indicated high logical reads and extended durations.
- Observed how generating indexes and optimizing queries can improve the process of data retrieval.
  - Creating a non-clustered index on the FirstName and LastName columns significantly boosted query performance.
  - Post-index execution plans showed reduced logical reads and faster execution times.
- Understood the concept of indexes, both non-clustered and clustered, in improving query performance by reducing logical reads and execution time.
- Learned to write optimized SQL queries for efficient database systems.
- The comparison between efficient and inefficient queries signifies the importance of writing optimized SQL queries for better performance. This has deepened my understanding of database management and of how strategic indexing and query writing can lead to more resource-efficient database systems.
- Learned to interpret execution plans for diagnosing and improving query performance. Recognized the value of execution plans in guiding performance tuning decisions.

**References:**

1. Lab Manual – 4 uploaded on MyCourses.
2. <https://youtu.be/YuRO9-rOgv4?si=Aj8euQKCy7OfmXtL>