University Of Engineering and Technology, Lahore
Computer Engineering Department

Course Name: Database Systems	**Course Code: CS363L**
Assignment Type: Lab	**Dated: 04-04-2022**
Semester: 6th	**Session: 2019**
Lab/Project/Assignment #: Lab 10	**CLOs to be covered: CLO1, CLO4**
Lab Title: Transactions, Query Optimization, and Indexes	**Teacher Name: Ms. Darakhshan**

Lab Evaluation:

CLO1	Construct DML queries to retrieve and store data in different relations					
Levels (Marks)	**Level1**	**Level2**	**Level3**	**Level4**	**Level5**	**Level6**
Cognitive (5)						
					Total	**/5**
CLO2	Construct DDL queries to manage relations, constraints, triggers and indexes					
	Level1	**Level2**	**Level3**	**Level4**	**Level5**	**Level6**
Cognitive (5)						
					Total	**/5**

Rubrics for Current Lab:

Scale	**Marks**	**Level**	**Rubric**
Excellent	**5**	L1	Solved 10 queries with same requirements as Rubric IV.
Very Good	**4**	L2	Wrote 7 queries with amalgam of different types of Joins, subqueries, and operators + Generated actual execution plans of these queries + Wrote answers on why DBMS made choices of X operators? + Rubric III requirements
Good	**3**	L3	Locally loaded bikestore database in SQL Server + Rubric 2 requirements
Basic	**2**	L4	Knows how to write SQL queries with Transaction statement.
Barely Acceptable	**1**	L5	Locally executed and understands all code shared in this lab using SQL Server.
Not Acceptable	**0**	L6	Did not attempt

LAB DETAILS:

Lab goals and objectives:

- To understand the basics of the transactions in databases
- Retrieving and restricting data using the SQL Transaction statement
- Adding large amount of data to tables.
- Evaluating query performances.
- Improve performance with indexes and views.
- Understanding Query Execution Plan

Theory/ Relevant Material:

Transaction:

The effects of all the SQL statements in a transaction can be either all **committed** (applied to the database) or all **rolled back** (undone from the database). A transaction is the propagation of one or more changes to the database. For example, if you are creating a record or updating a record or deleting a record from the table, then you are performing a transaction on that table. It is important to control these transactions to ensure the data integrity and to handle database errors.

Practically, you will club many SQL queries into a group and you will execute all of them together as a part of a transaction.

Properties of Transaction:

The following are the standard SQL transaction properties, abbreviated as **ACID**.

- Atomicity
- Consistency
- Isolation
- Durability

Atomicity:
Transactions in SQL are terminated at a point of error. In this way, It ensures the operations of work units were a success.

Consistency:
A successful commit transaction makes sure data is unchanged when a transaction begins and when it ends.

Isolation:

Transactions will operate independently and transparently.

Durability:

If the system fails, it ensures the result of committed Transaction continued across the application.

Commands of Transaction:

Below mentioned are the commands used to control the transactions.

COMMIT

This command will save the changes.

Syntax

```
CCOMMIT;
```

ROLLBACK

This command will rollback(undo or revert) the changes.

Syntax

```
RROLLBACK;
```

SAVEPOINT

This command creates save points for the group of transactions that has to be ROLLBACK.

Syntax

```
SAVEPOINT SAVEPOINT_NAME;
```

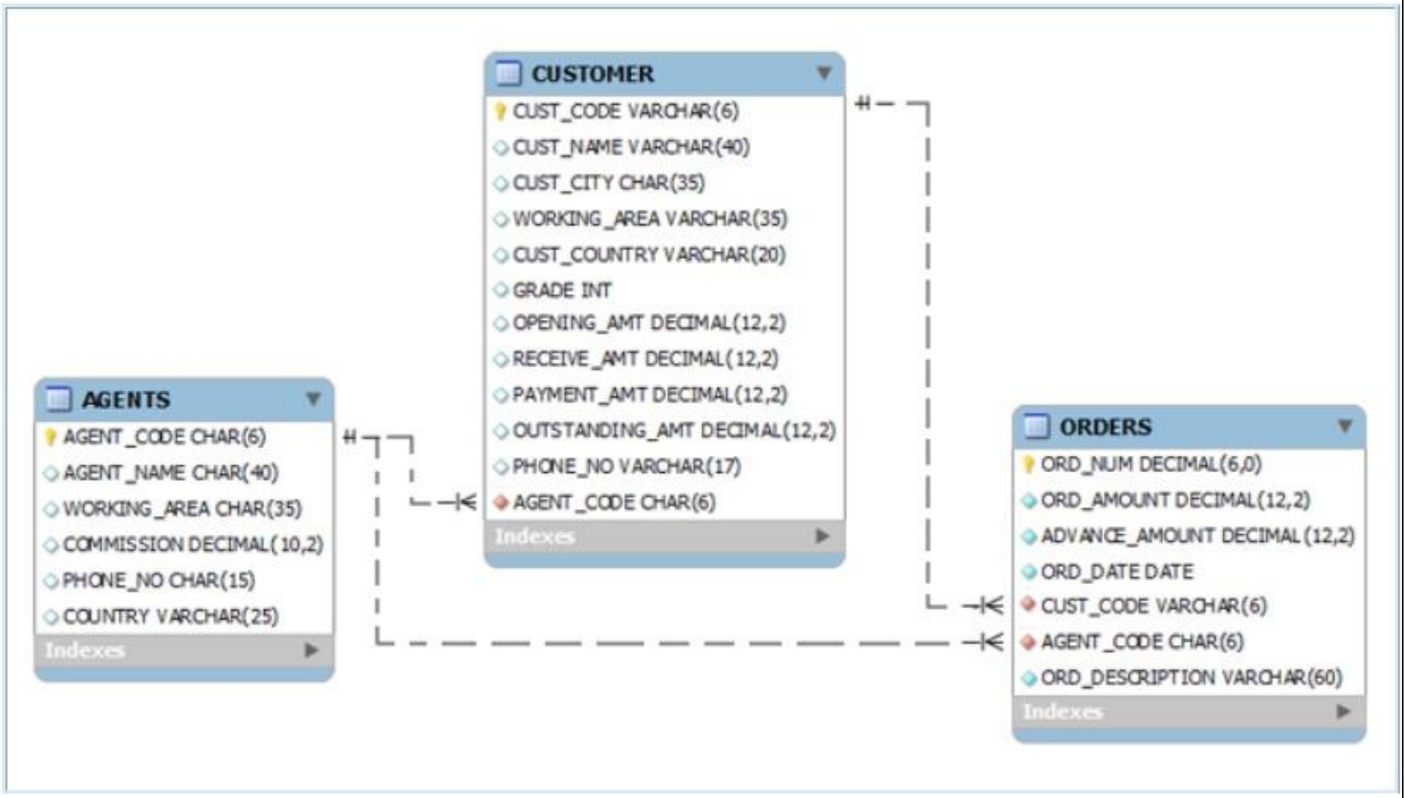
SET

Sets a name on a transaction.

Syntax:

```
SET TRANSACTION [READ ONLY| READ WRITE];
```

SCHEMA:



Lab tasks:

1. Place an order with:

Order number=202217
Order Amount=2000
Advance Paid=800
Order date=3/17/2022
Customer code=C202217
Agent Code=A202217
Order Description=COD

2. An order with Order ID=200120 is cancelled on the last minute but it is already added in the database.

Remove that specific entry as it is of no use

3. The Description for Order ID=200103 has changed from SOD to COD. Make changes in the database accordingly.
4. Customer table has the column named CUST_NAME. Break that column and make two separate columns from it

CUST_FIRST_NAME
CUST_LAST_NAME

5. COMMISSION Column of AGENTS is of no use for us. Remove that column.
6. Mark a transaction where agent code=A001.
7. Delete those records from the table which have order amount greater than 1000 and then ROLLBACK the changes in the database by keeping savepoints.

8. Write a named transaction to improve the grade of customers who have placed advance order amount.
9. Write a transaction to change the order description to SOD if the order amount is less than 1000 or not. If it is true, ROLLBACK the transaction; otherwise, Commit it in Select statement.
10. Make a named transaction to update the agent working area to Bangalore if the agent name is Mukesh and ROLLBACK the changes in the database by keeping savepoints.

Query Execution Plan:

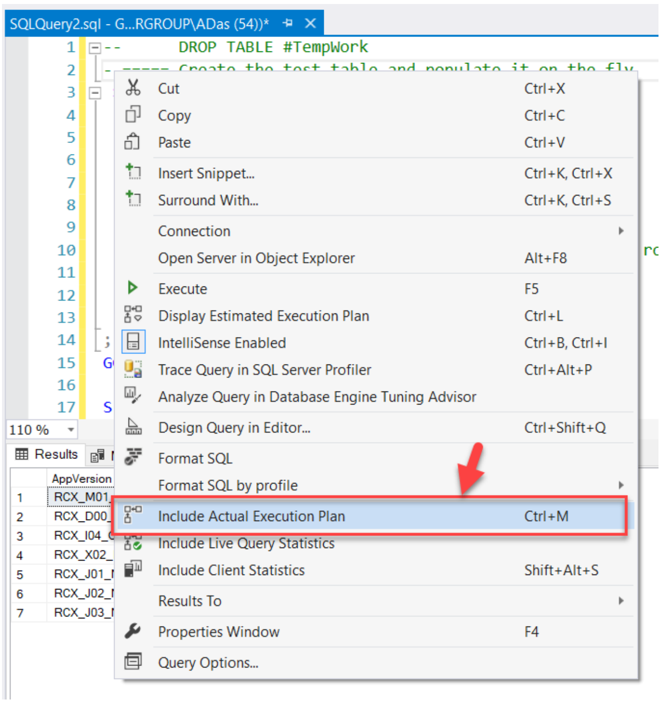
An execution plan in SQL Server Management Studio is a graphical representation of the various steps that are involved in fetching results from the database tables. Once a query is executed, the query processing engine quickly generates multiple execution plans and selects the one which returns the results with the best performance. There are two types of execution plans to be specific.

Estimated Execution Plan – As the name suggests, this type of execution plan is just a guess by the query processor about how the specific steps that are to be involved while returning the results. It is often generated before the query has been executed.

Actual Execution Plan – The Actual Execution Plan is generated after the query has been executed. It shows the actual operations and steps involved while executing the query. This may or may not differ from the Estimated Execution Plan

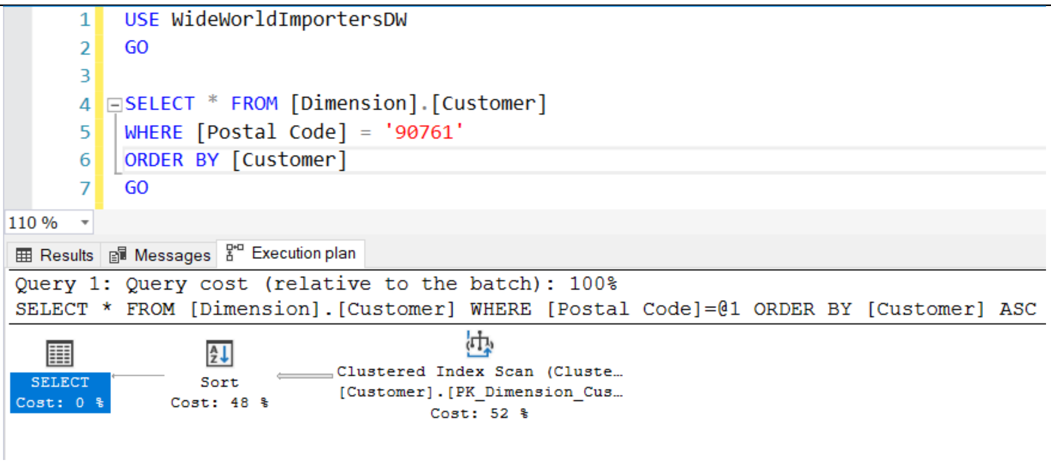
To generate an actual execution plan, you can follow following steps.

1. Hit "Ctrl + M" and it will generate the actual execution plan after the query has been executed successfully.
2. Right-click on the query window and select "Display Actual Execution Plan" from the context menu.


SQLQuery2.sql - G...RGROUP\ADas (54))*
110 %
Results
AppVersion
RCX_M01
RCX_D00
RCX_I04
RCX_X02
RCX_J01
RCX_J02
RCX_J03
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Insert Snippet... Ctrl+K, Ctrl+X
Surround With... Ctrl+K, Ctrl+S
Connection
Open Server in Object Explorer Alt+F8
Execute F5
Display Estimated Execution Plan Ctrl+L
IntelliSense Enabled Ctrl+B, Ctrl+I
Trace Query in SQL Server Profiler Ctrl+Alt+P
Analyze Query in Database Engine Tuning Advisor
Design Query in Editor... Ctrl+Shift+Q
Format SQL
Format SQL by profile
Include Actual Execution Plan Ctrl+M
Include Live Query Statistics
Include Client Statistics Shift+Alt+S
Results To
Properties Window F4
Query Options...

```
--        DROP TABLE #TempWork
----- Create the test table and populate it on the fly
```

Once you generate the execution plans as mentioned in the steps above, you'll see something like the diagram below as in Figure. For estimated plans, it will generate as soon as you perform the step whereas for the actual execution plan it will be displayed only after the query has been executed.

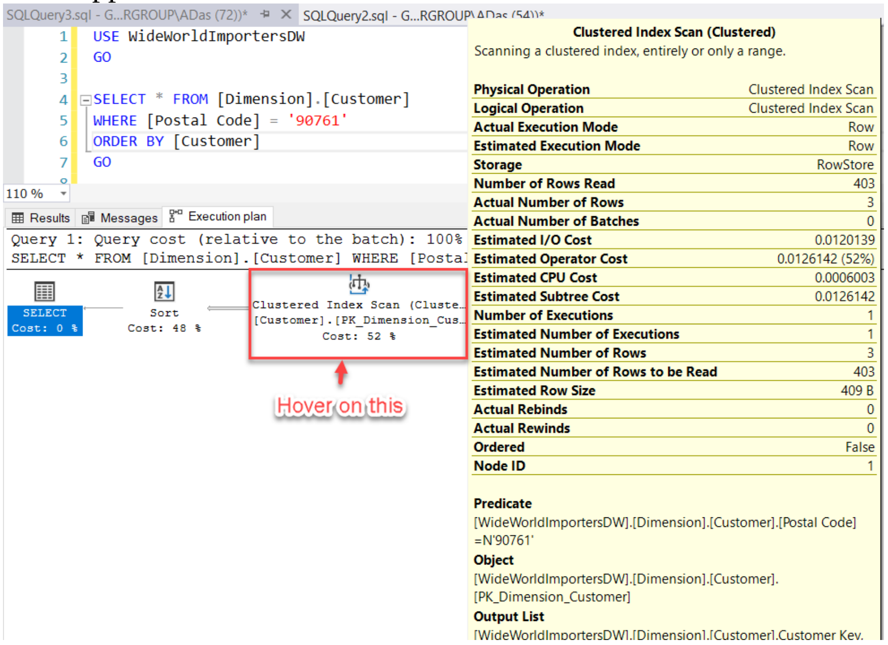


```
USE WideWorldImportersDW
GO

SELECT * FROM [Dimension].[Customer]
WHERE [Postal Code] = '90761'
ORDER BY [Customer]
GO
```

```
Query 1: Query cost (relative to the batch): 100%
SELECT * FROM [Dimension].[Customer] WHERE [Postal Code]=@1 ORDER BY [Customer] ASC
```

In the execution plan depicted in the above Figure, if you hover the cursor over the components, you can view the detailed stats for each of the operations and components being displayed in the execution plan. The plan is interpreted from right-to-left and top-to-bottom. Since our plan consists of only one single row, there is no need for the top-to-bottom approach.



SQLQuery3.sql - G...RGROUP\ADas (72))*
SQLQuery2.sql - G...RGROUP\ADas (54))*
110 %
Results
Messages
Execution plan
SELECT
Cost: 0 %
Sort
Cost: 48 %
Clustered Index Scan (Cluste…
[Customer].[PK_Dimension_Cus…
Cost: 52 %
Hover on this

```
USE WideWorldImportersDW
GO

SELECT * FROM [Dimension].[Customer]
WHERE [Postal Code] = '90761'
ORDER BY [Customer]
GO
```

```
Query 1: Query cost (relative to the batch): 100%
SELECT * FROM [Dimension].[Customer] WHERE [Posta
```

Clustered Index Scan (Clustered)

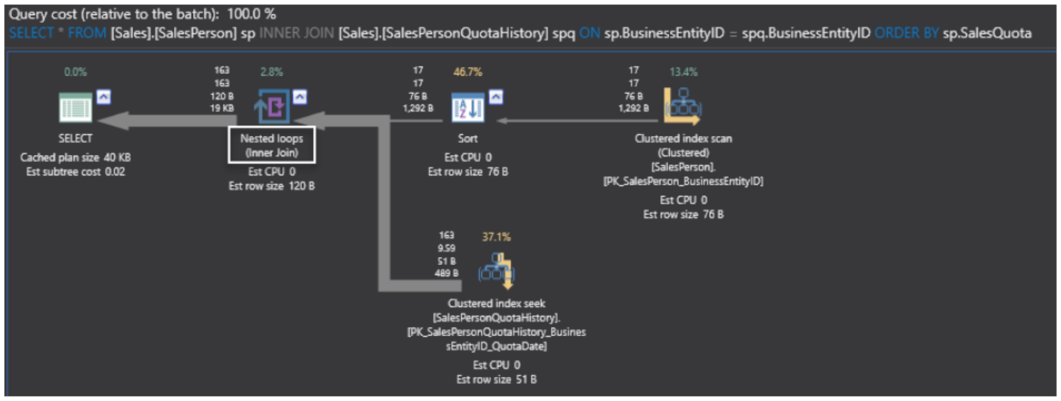
Scanning a clustered index, entirely or only a range.

Physical Operation	Clustered Index Scan
Logical Operation	Clustered Index Scan
Actual Execution Mode	Row
Estimated Execution Mode	Row
Storage	RowStore
Number of Rows Read	403
Actual Number of Rows	3
Actual Number of Batches	0
Estimated I/O Cost	0.0120139
Estimated Operator Cost	0.0126142 (52%)
Estimated CPU Cost	0.0006003
Estimated Subtree Cost	0.0126142
Number of Executions	1
Estimated Number of Executions	1
Estimated Number of Rows	3
Estimated Number of Rows to be Read	403
Estimated Row Size	409 B
Actual Rebinds	0
Actual Rewinds	0
Ordered	False
Node ID	1

Predicate
[WideWorldImportersDW].[Dimension].[Customer].[Postal Code] =N'90761'
Object
[WideWorldImportersDW].[Dimension].[Customer].[PK_Dimension_Customer]
Output List
[WideWorldImportersDW].[Dimension].[Customer].Customer Key,

You can get information about the parameters listed in yellow box using following link.
Another example for an actual execution plan is shown below.

```sql
USE AdventureWorks2014
GO
SELECT *
FROM [Sales].[SalesPerson] sp
INNER JOIN [Sales].[SalesPersonQuotaHistory] spq
  ON sp.BusinessEntityID = spq.BusinessEntityID
ORDER BY sp.SalesQuota
GO
```



Query cost (relative to the batch): 100.0 %
SELECT * FROM [Sales].[SalesPerson] sp INNER JOIN [Sales].[SalesPersonQuotaHistory] spq ON sp.BusinessEntityID = spq.BusinessEntityID ORDER BY sp.SalesQuota

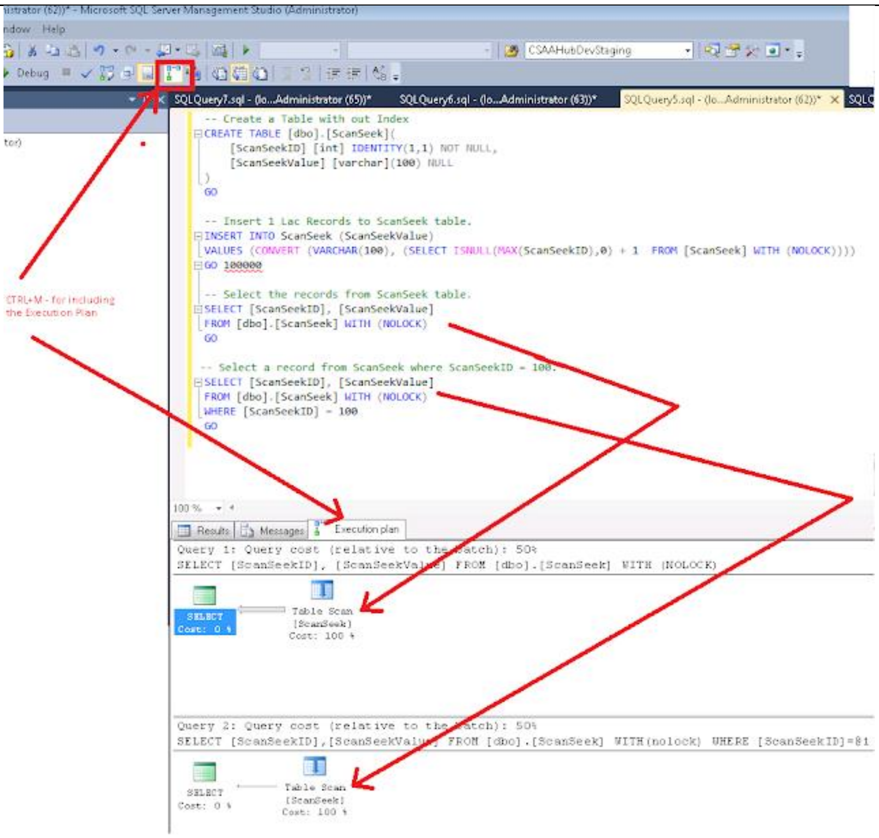
Indexes:

Reference link:

Try following set of queries and with Ctrl + L and Ctrl + M commands find out how queries are being executed with and without indexes. These commands will show you execution plan of your query.

SQL scripts:

```sql
CREATE TABLE [dbo].[ScanSeek](
	[ScanSeekID] [int] IDENTITY(1,1) NOT NULL,
	[ScanSeekValue] [varchar](100) NULL
)GO
-- Insert 1 Lac Records to ScanSeek table.
INSERT INTO ScanSeek (ScanSeekValue)
VALUES (CONVERT (VARCHAR(100), (SELECT ISNULL(MAX(ScanSeekID),0) + 1  FROM [ScanSeek]
WITH (NOLOCK))))
GO 100000
-- Select all the records from ScanSeek table.
SELECT [ScanSeekID], [ScanSeekValue]
FROM [dbo].[ScanSeek] WITH (NOLOCK)
GO
-- Select a record from ScanSeek table where ScanSeekID = 100.
SELECT [ScanSeekID], [ScanSeekValue]
FROM [dbo].[ScanSeek] WITH (NOLOCK)
WHERE [ScanSeekID] = 100
GO
```


istrator (62))* - Microsoft SQL Server Management Studio (Administrator)
ndow Help
CSAAHubDevStaging
Debug
SQLQuery7.sql - (lo...Administrator (65))*
SQLQuery6.sql - (lo...Administrator (63))*
SQLQuery5.sql - (lo...Administrator (62))*
CTRL+M - for including the Execution Plan
100 %
Results
Messages
Execution plan
SELECT Cost: 0 %
Table Scan [ScanSeek] Cost: 100 %
SELECT Cost: 0 %
Table Scan [ScanSeek] Cost: 100 %

```
-- Create a Table with out Index
CREATE TABLE [dbo].[ScanSeek](
    [ScanSeekID] [int] IDENTITY(1,1) NOT NULL,
    [ScanSeekValue] [varchar](100) NULL
)
GO

-- Insert 1 Lac Records to ScanSeek table.
INSERT INTO ScanSeek (ScanSeekValue)
VALUES (CONVERT (VARCHAR(100), (SELECT ISNULL(MAX(ScanSeekID),0) + 1  FROM [ScanSeek] WITH (NOLOCK))))
GO 100000

-- Select the records from ScanSeek table.
SELECT [ScanSeekID], [ScanSeekValue]
FROM [dbo].[ScanSeek] WITH (NOLOCK)
GO

-- Select a record from ScanSeek where ScanSeekID = 100.
SELECT [ScanSeekID], [ScanSeekValue]
FROM [dbo].[ScanSeek] WITH (NOLOCK)
WHERE [ScanSeekID] = 100
GO
```

```
Query 1: Query cost (relative to the batch): 50%
SELECT [ScanSeekID], [ScanSeekValue] FROM [dbo].[ScanSeek] WITH (NOLOCK)

Query 2: Query cost (relative to the batch): 50%
SELECT [ScanSeekID],[ScanSeekValue] FROM [dbo].[ScanSeek] WITH(nolock) WHERE [ScanSeekID]=@1
```

From the above two select queries, either to fetch all the records from table OR to fetch only one the matched record from table where ScanID=100, while seeing the execution plan the Cost is 100%, so it is obviouly bottleneck for the performance,
From the second SELECT query, the query optimizer used the table scan which means it reads/scans 100000 (1Lac) records sequentially row by row of the table for fetching the matched row.
How to overcome this issue OR to fetch only the matched records from table?
Adding/creating appropriate indices on table that helps for retriving the data fast.
Index Scan:
An index scan is, scan the data or index pages to find the appropriate records, an index scan retrieves all the rows from the table, it means that all the leaf-level of the index was searched to find the information for the query,
There are two types of Index Scan.

1. Index Scan on Clustered Index
2. Index Scan On Non-Clustred Index

1. Index Scan on Clustered Index

In a table, when the index is a clustered index and search key column is not indexed, in this case, SQL server navigates from root level to leaf-level (data pages) until the data match, which means that scanning the data page by using the clustering key value as a pointer/reference.

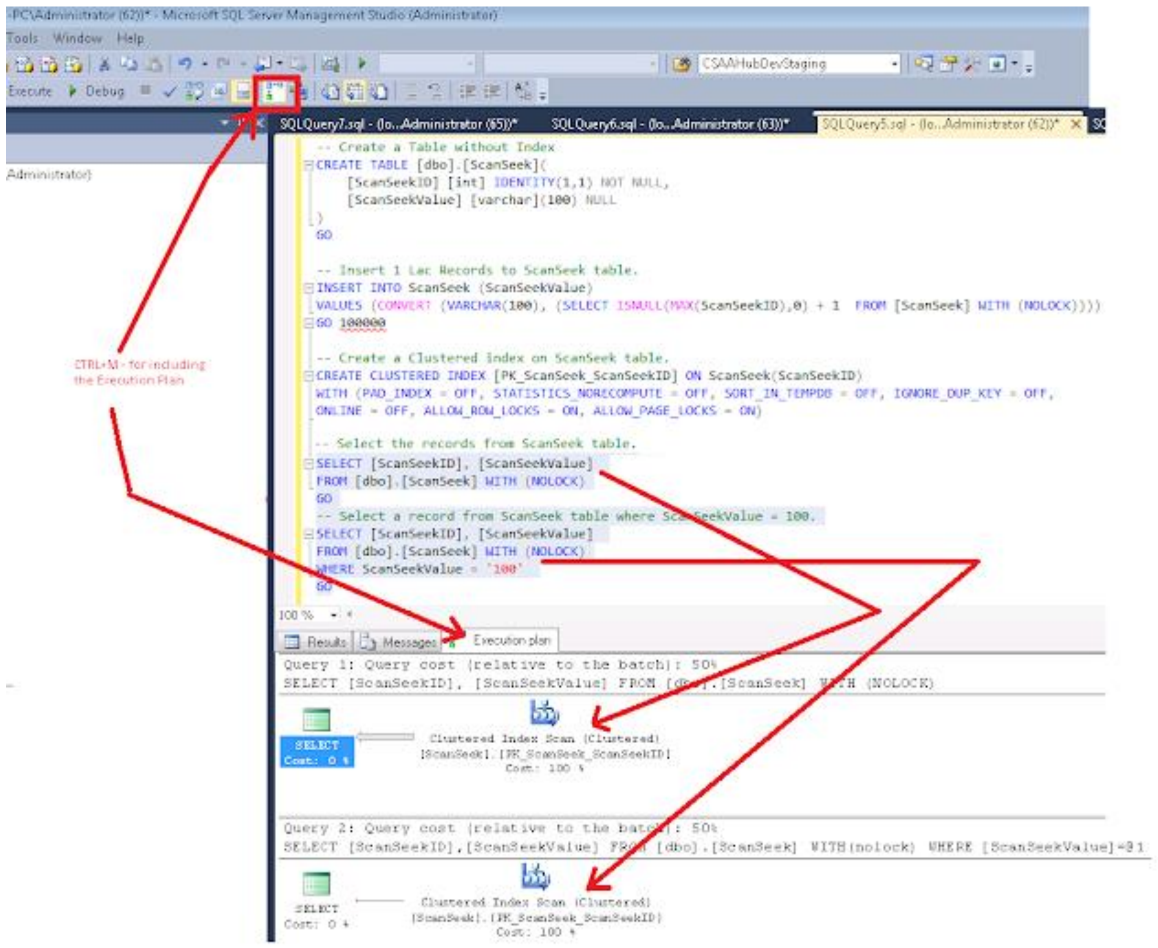
SQL scripts
- Create a Table without Index CREATE TABLE [dbo].[ScanSeek]([ScanSeekID] [int] IDENTITY(1,1) NOT NULL,   [ScanSeekValue] [varchar](100) NULL) GO -- Insert 1 Lac Records to ScanSeek table.

```
INSERT INTO ScanSeek (ScanSeekValue)
VALUES (CONVERT (VARCHAR(100), (SELECT ISNULL(MAX(ScanSeekID),0) + 1
FROM [ScanSeek] WITH (NOLOCK))))
GO 100000

-- Create a Clustered index on ScanSeek table.
CREATE CLUSTERED INDEX [PK_ScanSeek_ScanSeekID] ON ScanSeek(ScanSeekID)
WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB
 = OFF, IGNORE_DUP_KEY = OFF,
ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON)

-- Select all the records from ScanSeek table.
SELECT [ScanSeekID], [ScanSeekValue]
FROM [dbo].[ScanSeek] WITH (NOLOCK)
GO

-- Select a record from ScanSeek table where ScanSeekValue = 100.
SELECT [ScanSeekID], [ScanSeekValue]
FROM [dbo].[ScanSeek] WITH (NOLOCK)
WHERE ScanSeekValue = '100'
GO
```


-PC\Administrator (62))* - Microsoft SQL Server Management Studio (Administrator)
Tools Window Help
CSAAHubDevStaging
Execute Debug
SQLQuery7.sql - (lo...Administrator (65))*
SQLQuery6.sql - (lo...Administrator (63))*
SQLQuery5.sql - (lo...Administrator (62))*
Administrator)
CTRL+M - for including the Execution Plan
100 %
Results
Messages
Execution plan

```
-- Create a Table without Index
CREATE TABLE [dbo].[ScanSeek](
    [ScanSeekID] [int] IDENTITY(1,1) NOT NULL,
    [ScanSeekValue] [varchar](100) NULL
)
GO

-- Insert 1 Lac Records to ScanSeek table.
INSERT INTO ScanSeek (ScanSeekValue)
VALUES (CONVERT (VARCHAR(100), (SELECT ISNULL(MAX(ScanSeekID),0) + 1  FROM [ScanSeek] WITH (NOLOCK))))
GO 100000

-- Create a Clustered index on ScanSeek table.
CREATE CLUSTERED INDEX [PK_ScanSeek_ScanSeekID] ON ScanSeek(ScanSeekID)
WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF, IGNORE_DUP_KEY = OFF,
ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON)

-- Select the records from ScanSeek table.
SELECT [ScanSeekID], [ScanSeekValue]
FROM [dbo].[ScanSeek] WITH (NOLOCK)
GO
-- Select a record from ScanSeek table where ScanSeekValue = 100.
SELECT [ScanSeekID], [ScanSeekValue]
FROM [dbo].[ScanSeek] WITH (NOLOCK)
WHERE ScanSeekValue = '100'
GO
```

```
Query 1: Query cost (relative to the batch): 50%
SELECT [ScanSeekID], [ScanSeekValue] FROM [dbo].[ScanSeek] WITH (NOLOCK)

SELECT          Clustered Index Scan (Clustered)
Cost: 0 %       [ScanSeek].[PK_ScanSeek_ScanSeekID]
                Cost: 100 %

Query 2: Query cost (relative to the batch): 50%
SELECT [ScanSeekID],[ScanSeekValue] FROM [dbo].[ScanSeek] WITH(nolock) WHERE [ScanSeekValue]=@1

SELECT          Clustered Index Scan (Clustered)
Cost: 0 %       [ScanSeek].[PK_ScanSeek_ScanSeekID]
                Cost: 100 %
```

For both types of select queries, DBMS is using index file for searching the objects.

2. **Index Scan on Non-Clustered Index**

A Non-Clustered Index Scan occurs when columns are part of the non-clustered index and the query goes about accessing a large amount of data across this particular index

SQL Scripts:

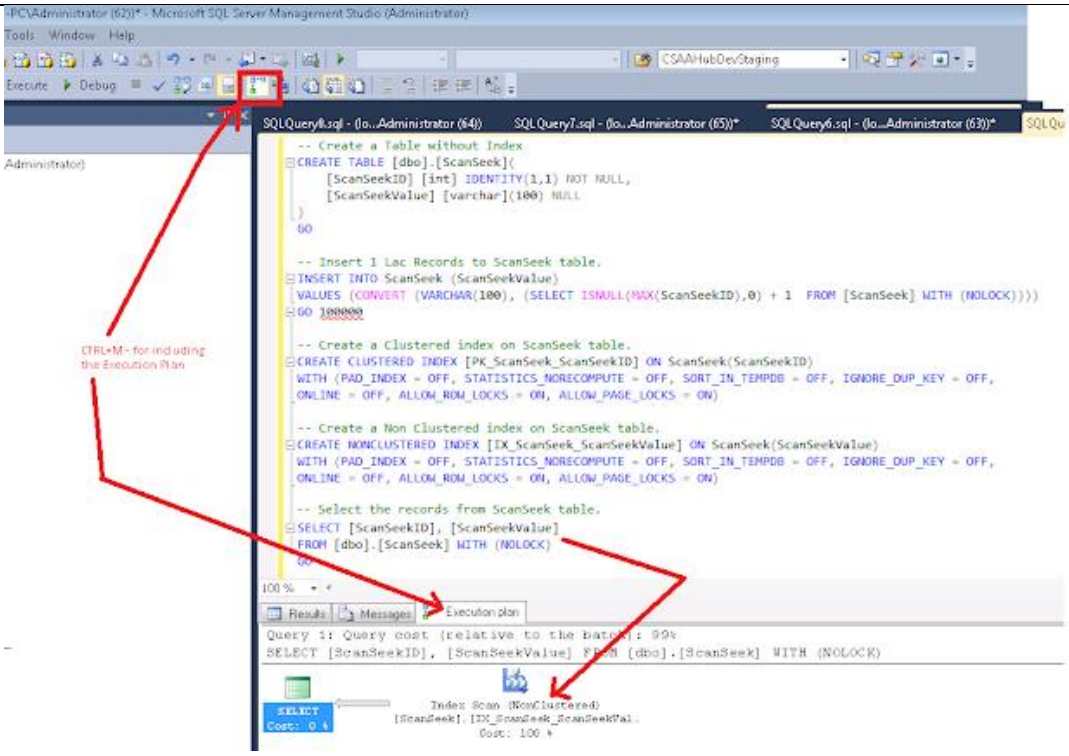
```
-- Create a Table without Index
CREATE TABLE [dbo].[ScanSeek](
   [ScanSeekID] [int] IDENTITY(1,1) NOT NULL,
   [ScanSeekValue] [varchar](100) NULL
)
GO

-- Insert 1 Lac Records to ScanSeek table.
INSERT INTO ScanSeek (ScanSeekValue)
VALUES (CONVERT (VARCHAR(100), (SELECT ISNULL(MAX(ScanSeekID),0) + 1  FR
OM [ScanSeek] WITH (NOLOCK))))
GO 100000

-- Create a Clustered index on ScanSeek table.
CREATE CLUSTERED INDEX [PK_ScanSeek_ScanSeekID] ON ScanSeek(ScanSeekID)
WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB
 = OFF, IGNORE_DUP_KEY = OFF,
ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON)

-- Create a Non Clustered index on ScanSeek table.
CREATE NONCLUSTERED INDEX [IX_ScanSeek_ScanSeekValue] ON ScanSeek(ScanSe
ekValue)
WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB
 = OFF, IGNORE_DUP_KEY = OFF,
ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON)

-- Select all the records from ScanSeek table.
SELECT [ScanSeekID], [ScanSeekValue]
FROM [dbo].[ScanSeek] WITH (NOLOCK)
GO
```


-PC\Administrator (62))* - Microsoft SQL Server Management Studio (Administrator)
Tools Window Help
CSAAHubDevStaging
Execute Debug
SQLQuery8.sql - (lo...Administrator (64))
SQLQuery7.sql - (lo...Administrator (65))*
SQLQuery6.sql - (lo...Administrator (63))*
Administrator)
CTRL+M - for including the Execution Plan
100 %
Results
Messages
Execution plan
SELECT
Cost: 0 %
Index Scan (NonClustered)
[ScanSeek].[IX_ScanSeek_ScanSeekVal...
Cost: 100 %

```
-- Create a Table without Index
CREATE TABLE [dbo].[ScanSeek](
    [ScanSeekID] [int] IDENTITY(1,1) NOT NULL,
    [ScanSeekValue] [varchar](100) NULL
)
GO

-- Insert 1 Lac Records to ScanSeek table.
INSERT INTO ScanSeek (ScanSeekValue)
VALUES (CONVERT (VARCHAR(100), (SELECT ISNULL(MAX(ScanSeekID),0) + 1  FROM [ScanSeek] WITH (NOLOCK))))
GO 100000

-- Create a Clustered index on ScanSeek table.
CREATE CLUSTERED INDEX [PK_ScanSeek_ScanSeekID] ON ScanSeek(ScanSeekID)
WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF, IGNORE_DUP_KEY = OFF,
ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON)

-- Create a Non Clustered index on ScanSeek table.
CREATE NONCLUSTERED INDEX [IX_ScanSeek_ScanSeekValue] ON ScanSeek(ScanSeekValue)
WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF, IGNORE_DUP_KEY = OFF,
ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON)

-- Select the records from ScanSeek table.
SELECT [ScanSeekID], [ScanSeekValue]
FROM [dbo].[ScanSeek] WITH (NOLOCK)
GO
```

```
Query 1: Query cost (relative to the batch): 99%
SELECT [ScanSeekID], [ScanSeekValue] FROM [dbo].[ScanSeek] WITH (NOLOCK)
```

As you can see in this example, that non-clustered index has been used to create the query plan.

Expected Outcome:

- We will use bikestore database for this lab. (https://www.sqlservertutorial.net/load-sample-database/)
- We will add 20,000 dummy rows in our tables. You can use this link for details: https://www.mssqltips.com/sqlservertip/5148/populate-large-tables-with-random-data-for-sql-server-performance-testing/
- After adding dummy rows, try executing several sets of queries, create views and indexes and observe the time along-with other details. Use EXPLAIN for this purpose (https://www.sitepoint.com/using-explain-to-write-better-mysql-queries/)
- Also, generate query execution plans for your plans and understand why DBMS has made certain choices in the selection of options.
- You need to create at least 10 such plans and write your understanding as description.

Submission Instructions:

Make a document name DBLab10_2019_CE_X.docx, add supporting SQL scripts of your lab work and submit on google classroom by Sunday, 10th April, 2022 9 P.M.