

Use case 1: NOT IN

The aim is to find the userId of users who haven't posted anything on their profile yet.

Note that postUserId is a foreign key in the POSTS relation that refers to the primary key userId of the USERS relation. The number of tuples in POSTS is much lesser than the number of tuples in USERS, hence this query is a valid and logical one to make.

At this point, the indexes are:

- On USERS: index on the userId key which is the primary key of this relation.
PK_USERS_CB9A1CFF004D37AB is clustered, unique, primary key located on FG1
- On POSTS: index on postId key which is the primary key of this relation.
PK_POSTS_DD0C739A88B15966 is clustered, unique, primary key located on FG1

Version 1: `select userId from USERS where USERS.userId not in (select postUserId from POSTS)`

This query is written using not in. It is perhaps the simplest form of writing the query to get the desired output, however, it takes more time for the query compiler to process this query as compared to the next two versions.

Notice for the USERS table, scan count is 1, number of logical reads is 2775 and number of physical reads is 3. And for the POSTS table, scan count is 1, number of logical reads is 371 and number of physical reads is 1. Also see that the read-aheads in this and each of the following versions is non-zero, because I have cleared the cache buffers before every statement and so the pages will not be fetched from the cache.

```

SQLQuery2.sql - DESKTOP-8AIVGG4.InstagramDB (DESKTOP-8AIVGG4(KR (55)) - Microsoft SQL Server Management Studio
File Edit View Query Project Tools Window Help
New Query Execute Results Messages Execution plan CustId
Object Explorer
Connect DESKTOP-8AIVGG4 (SQL Server 15.0.200)
Databases Security Server Objects Replication PolyBase Always On High Availability Management Integration Services Catalogs SQL Server Agent (Agent XPs disabled) XEvent Profiler
SQLQuery2.sql - D...P-8AIVGG4(KR (55)) tablesAndTriggers...P-8AIVGG4(KR (56)) SQLQuery1.sql - D...P-8AIVGG4(KR (54)) queries.sql - DESK...P-8AIVGG4(KR (53))
select userId from USERS where USERS.userId not in (select postUserId from POSTS)
go

150 % | Results Messages Execution plan
SQL Server parse and compile time:
CPU time = 0 ms, elapsed time = 0 ms.

SQL Server Execution Times:
CPU time = 0 ms, elapsed time = 0 ms.

SQL Server parse and compile time:
CPU time = 0 ms, elapsed time = 12 ms.

(217962 rows affected)
Table 'Workfile'. Scan count 0, logical reads 0, physical reads 0, page server reads 0, read-ahead reads 0, page server read-ahead reads 0, lob logic reads 0, lob writes 0, lob read-ahead reads 0.
Table 'Worktable'. Scan count 0, logical reads 0, physical reads 0, page server reads 0, read-ahead reads 0, page server read-ahead reads 0, lob logic reads 0, lob writes 0, lob read-ahead reads 0.
Table 'USERS'. Scan count 1, logical reads 2775, physical reads 3, page server reads 0, read-ahead reads 2787, page server read-ahead reads 0, lob logic reads 0, lob writes 0, lob read-ahead reads 0.
Table 'POSTS'. Scan count 1, logical reads 371, physical reads 1, page server reads 0, read-ahead reads 363, page server read-ahead reads 0, lob logic reads 0, lob writes 0, lob read-ahead reads 0.

(1 row affected)

SQL Server Execution Times:
CPU time = 125 ms, elapsed time = 1295 ms.

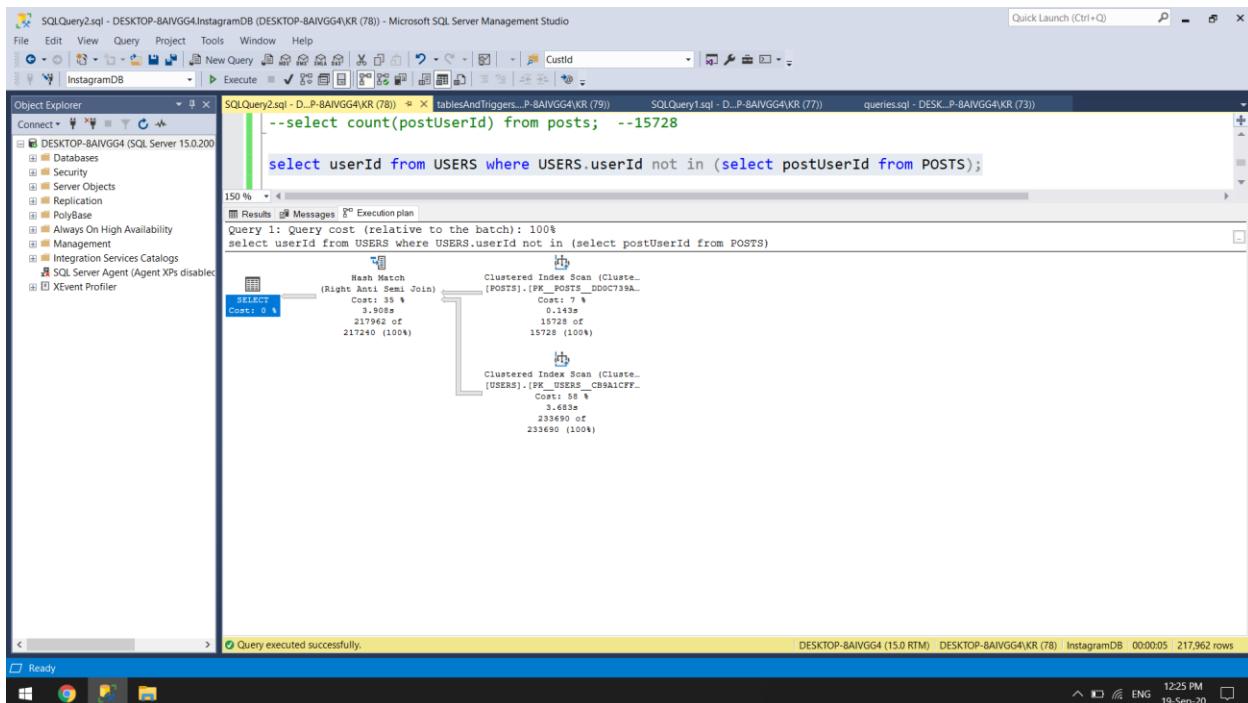
SQL Server parse and compile time:
CPU time = 0 ms, elapsed time = 0 ms.

SQL Server Execution Times:
CPU time = 0 ms, elapsed time = 0 ms.

Completion time: 2020-09-20T11:01:18.4456642+05:30
|
```

Query executed successfully.

The execution plan is as follows- The clustered index scan on POSTS costs 7% and the clustered index scan on USERS costs 58%.



The details of the clustered index scan on POSTS is shown. The output list contains postUserId because that is what is projected in the inner/sub-query.

```
--select count(postUserId) from posts; --15728
select userId from USERS where USERS.userId not in (select postUserId from POSTS);
```

Execution plan details:

- Hash Match (Right Anti Semi Join)**: Cost: 3.908, 217962 of 217240 (100%)
 - Physical Operation**: Logical Operation: Row, Estimated Execution Mode: RowStore
 - Estimated Operator Cost**: 0.293175 (7%)
 - Estimated I/O Cost**: 0.275718
 - Estimated CPU Cost**: 0.0174578
 - Estimated Subtree Cost**: 0.293175
 - Number of Executions**: 1
 - Estimated Number of Executions**: 1
 - Estimated Number of Rows to be Read**: 15728
 - Estimated Number of Rows Per Execution**: 15728
 - Estimated Row Size**: 11 B
 - Actual Rebinds**: 0
 - Actual Rewinds**: 0
 - Ordered**: False
 - Node ID**: 1
- Clustered Index Scan (Clustered)**: Cost: 1.5728
 - Physical Operation**: Logical Operation: Clustered Index Scan, Storage: RowStore
 - Estimated Operator Cost**: 0.293175 (7%)
 - Estimated I/O Cost**: 0.275718
 - Estimated CPU Cost**: 0.0174578
 - Estimated Subtree Cost**: 0.293175
 - Number of Executions**: 1
 - Estimated Number of Executions**: 1
 - Estimated Number of Rows for All Executions**: 15728
 - Actual Number of Batches**: 0
 - Estimated Operator Cost**: 0.293175 (7%)
 - Estimated I/O Cost**: 0.275718
 - Estimated CPU Cost**: 0.0174578
 - Estimated Subtree Cost**: 0.293175
 - Number of Executions**: 1
 - Estimated Number of Executions**: 1
 - Estimated Number of Rows to be Read**: 15728
 - Estimated Number of Rows Per Execution**: 15728
 - Estimated Row Size**: 11 B
 - Actual Rebinds**: 0
 - Actual Rewinds**: 0
 - Ordered**: False
 - Node ID**: 1

Object: [InstagramDB].[dbo].[POSTS].[PK_POSTS]

Output List: [InstagramDB].[dbo].[POSTS].postUserId

The details of the clustered index scan on USERS is shown. The output list contains userId because that is what is projected in the outer query.

```
--select count(postUserId) from posts; --15728
select userId from USERS where USERS.userId not in (select postUserId from POSTS);
```

Execution plan details:

- Hash Match (Right Anti Semi Join)**: Cost: 3.908, 217962 of 217240 (100%)
 - Physical Operation**: Logical Operation: Row, Estimated Execution Mode: RowStore
 - Estimated Operator Cost**: 0.293175 (58%)
 - Estimated I/O Cost**: 0.24683
 - Estimated CPU Cost**: 0.252716
 - Estimated Subtree Cost**: 0.293175
 - Number of Executions**: 1
 - Estimated Number of Executions**: 1
 - Estimated Number of Rows to be Read**: 233690
 - Estimated Number of Rows Per Execution**: 233690
 - Estimated Row Size**: 11 B
 - Actual Rebinds**: 0
 - Actual Rewinds**: 0
 - Ordered**: False
 - Node ID**: 2
- Clustered Index Scan (Clustered)**: Cost: 1.5728
 - Physical Operation**: Logical Operation: Clustered Index Scan, Storage: RowStore
 - Estimated Operator Cost**: 0.293175 (58%)
 - Estimated I/O Cost**: 0.275718
 - Estimated CPU Cost**: 0.0174578
 - Estimated Subtree Cost**: 0.293175
 - Number of Executions**: 1
 - Estimated Number of Executions**: 1
 - Estimated Number of Rows for All Executions**: 233690
 - Actual Number of Batches**: 0
 - Estimated Operator Cost**: 0.293175 (58%)
 - Estimated I/O Cost**: 0.275718
 - Estimated CPU Cost**: 0.0174578
 - Estimated Subtree Cost**: 0.293175
 - Number of Executions**: 1
 - Estimated Number of Executions**: 1
 - Estimated Number of Rows to be Read**: 233690
 - Estimated Number of Rows Per Execution**: 233690
 - Estimated Row Size**: 11 B
 - Actual Rebinds**: 0
 - Actual Rewinds**: 0
 - Ordered**: False
 - Node ID**: 2

Object: [InstagramDB].[dbo].[USERS].[PK_USERS]

Output List: [InstagramDB].[dbo].[USERS].userId

The hash match users each row from POSTS to build a hash table and each row from USERS to probe into the hash table, and then outputs the matching rows. Note that what is projected (output list) is userId from USERS. This makes up about 35% of the cost of the query.

The screenshot shows the Microsoft SQL Server Management Studio interface. In the Object Explorer, the 'InstagramDB' database is selected. In the center pane, a query window displays the following T-SQL code:

```
--select count(postUserId) from posts; --15728
select userId from USERS where USERS.userId not in (select postUserId from POSTS);
```

An execution plan is shown for the second query, indicating a Hash Match (Right Anti Semi Join) operation. The plan details the following statistics:

Physical Operation	Logical Operation
Hash Match	Right Anti Semi Join
(Right Anti Semi Join)	Clustered Index Scan (Clustered Index Scan on [POSTS].[PK_POSTS_DDC739A...])
Cost: 35 \$	Cost: 7 %
1.968	
217962 of 217240 (100%)	

A note in the plan states: "Use each row from the top input to build a hash table, and each row from the bottom input to probe into the hash table, outputting all matching rows."

Below the execution plan, the 'Statistics' section provides detailed performance metrics:

Actual Number of Rows for All Executions	Estimated Execution Mode
217962	Row
0	Estimated Execution Mode
217962	Actual Number of Batches
0	Estimated Operator Cost
1.406645 (35%)	Estimated I/O Cost
0	Estimated CPU Cost
1.406645	Estimated Subtree Cost
4.00386	Number of Executions
1	Estimated Number of Executions
1	Estimated Number of Rows Per Execution
217240	Estimated Row Size
11 B	Actual Rewinds
0	Node ID

The 'Output List' section shows the results of the query:

```
[InstagramDB].[dbo].[USERS].userId
[InstagramDB].[dbo].[USERS].userId
```

The status bar at the bottom right indicates: DESKTOP-8AIVGG4 (15.0 RTM) | DESKTOP-8AIVGG4(KR (78)) | InstagramDB | 00:00:05 | 217,962 rows | 12:26 PM | ENG | 19-Sep-20.

Version 2: `select userId from USERS left outer join POSTS on POSTS.postUserId=USERS.userId where POSTS.postUserId is NULL`

This query is written using left outer join.

For the USERS table, scan count is 1, number of logical reads is 2775 and number of physical reads is 3. And for the POSTS table, scan count is 1, number of logical reads is 371 and number of physical reads is 1. This is the same as in Version 1. Looking at this, we can say that the query compiler executes in the same way (most optimum way) to achieve the end result, irrespective of how the query is written.

The screenshot shows the Microsoft SQL Server Management Studio interface with two open queries:

- Query1.sql - D...P-8AIVGG4(KR (78))**:

```
select postUserId from POSTS

select userId from USERS left outer join POSTS on POSTS.postUserId=USERS.userId where POSTS.postUserId is NULL
```
- queries.sql - DESKTOP-8AIVGG4(KR (73))**:

```
150 %

Results Messages Execution plan
SQL Server parse and compile time:
CPU time = 0 ms, elapsed time = 0 ms.

SQL Server Execution Times:
CPU time = 0 ms, elapsed time = 0 ms.
SQL Server parse and compile time:
CPU time = 0 ms, elapsed time = 12 ms.

(217962 rows affected)
Table 'Workfile'. Scan count 0, logical reads 0, physical reads 0, page server reads 0, read-ahead reads 0, page server read-ahead reads 0, lob logic
Table 'Worktable'. Scan count 0, logical reads 0, physical reads 0, page server reads 0, read-ahead reads 0, page server read-ahead reads 0, lob logic
Table 'USERS'. Scan count 1, logical reads 2775, physical reads 3, page server reads 0, read-ahead reads 2787, page server read-ahead reads 0, lob logic
Table 'POSTS'. Scan count 1, logical reads 371, physical reads 1, page server reads 0, read-ahead reads 363, page server read-ahead reads 0, lob logic

(1 row affected)

SQL Server Execution Times:
CPU time = 219 ms, elapsed time = 1232 ms.
SQL Server parse and compile time:
CPU time = 0 ms, elapsed time = 0 ms.

SQL Server Execution Times:
CPU time = 0 ms, elapsed time = 0 ms.

Completion time: 2020-09-19T12:23:53.7486342+05:30
```

The status bar at the bottom indicates "Query executed successfully." and shows system information like "DESKTOP-8AIVGG4 (15.0 RTM) | DESKTOP-8AIVGG4(KR (78)) | InstagramDB | 00:00:01 | 217,962 rows".

This is the execution plan of the query. Here, the cost of clustered index scan on POSTS is the same as in Version 1.

The screenshot shows the SQL Server Management Studio interface with the following details:

- Object Explorer:** Shows the database structure for "DESKTOP-BAIVGG4 (SQL Server 15.0.200)".
- SQL Query1.sql - D...P-8AIVGG4(KR (78))**: Contains the query:


```
select postUserId from POSTS

select userId from USERS left outer join POSTS on POSTS.postUserId=USERS.userId where POSTS.postUserId is NULL
```
- Execution Plan:** A detailed diagram showing the execution flow. It starts with a "SELECT" node, followed by a "Filter" node (Cost: 3 %), then a "Hash Match (Right Outer Join)" node (Cost: 34 %). This leads to two "Clustered Index Scan" nodes: one for the [POSTS].[PK__POSTS__DD0C739A88B15966] clustered index on the [POSTS] table, and another for the [USERS].[PK__USERS__CBDA1CF] clustered index on the [USERS] table.
- Properties Window (right side):** Displays properties for the "Clustered Index Scan (Clustered) [POSTS].[PK__POSTS__DD0C739A88B15966]" operator, including:

Physical Operation	Storage
Clustered Index Scan	RowStore
Logical Operation	
Actual Execution Mode	Row
Estimated Execution Mode	Row
Storage	
Number of Rows Read	15728
Actual Number of Rows for All Executions	15728
Number of Batches	0
Estimated Operator Cost	0.293175 (7%)
Estimated I/O Cost	0.0275718
Estimated CPU Cost	0.0174578
Estimated Subtree Cost	0.293175
Number of Executions	1
Estimated Number of Executions	1
Estimated Number of Rows to be Read	15728
Estimated Number of Rows Per Execution	15728
Estimated Row Size	11 B
Actual Rebinds	0
Actual Rewinds	0
Ordered	False
Node ID	2
- Status Bar:** Shows "Query executed successfully." and "Ready".

The clustered index scan on USERS has come down to 56% from 58%, but there is an additional filter step after the hash match which wasn't there in Version 1.

The screenshot shows the Microsoft SQL Server Management Studio interface. In the Object Explorer, the database 'InstagramDB' is selected. In the main pane, a query window displays the following T-SQL code:

```

SELECT postUserId FROM POSTS
SELECT userId FROM USERS left outer join POSTS on POSTS.postUserId=USERS.userId where POSTS.postUserId is NULL

```

An execution plan is shown for the second query. The plan starts with a 'Filter' node (Cost: 3 %) followed by a 'Hash Match (Right Outer Join)' node. This is followed by two 'Clustered Index Scan (Clustered)' nodes: one for the 'POSTS' table and one for the 'USERS' table. The output list for the Hash Match node includes 'userId' from the USERS table and 'postUserId' from the POSTS table.

The execution plan details table shows the following statistics:

Physical Operation	Logical Operation
Clustered Index Scan	Clustered Index Scan
Actual Execution Mode	Row
Estimated Execution Mode	Row
Storage	RowScan
Number of Rows Read	233690
Actual Number of Rows for All Executions	233690
Actual Number of Batches	0
Estimated Operator Cost	2.30404 (56%)
Estimated I/O Cost	2.04683
Estimated CPU Cost	0.257216
Estimated Subtree Cost	2.30404
Number of Executions	1
Estimated Number of Executions	1
Estimated Number of Rows to be Read	233690
Estimated Number of Rows Per Execution	233690
Estimated Row Size	11 B
Actual Rebinds	0
Actual Rewinds	0
Ordered	False
Node ID	3

The 'Object' section of the tooltip for the Hash Match node shows the output list: '[InstagramDB].[dbo].[USERS].[userId], [InstagramDB].[dbo].[POSTS].[postUserId]'. The 'Output List' section also lists '[InstagramDB].[dbo].[USERS].[userId]'.

Unlike in Version 1 where the output list for hash match was only userId, here the output list has userId from USERS and postUserId from POSTS.

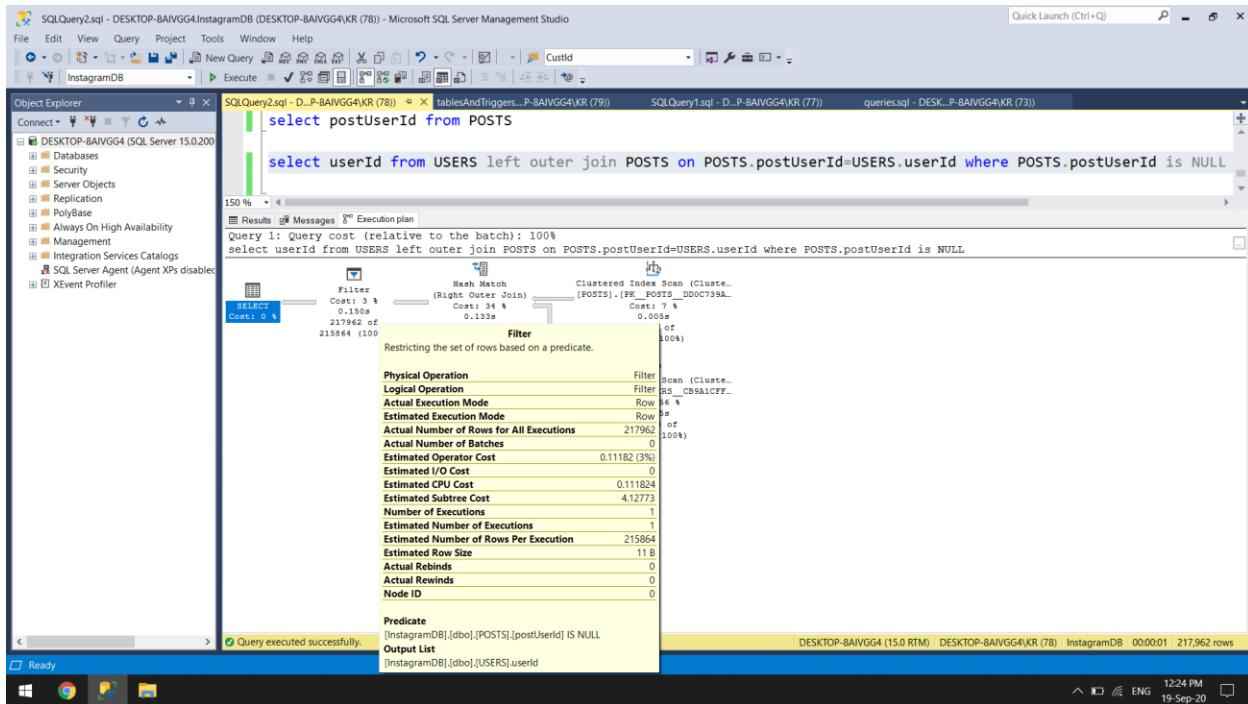
This screenshot is similar to the previous one, but the execution plan is focused on the Hash Match node. A callout box highlights the 'Hash Match' node, explaining its function: 'Use each row from the top input to build a hash table, and each row from the bottom input to probe into the hash table, outputting all matching rows.'

The execution plan details table for the Hash Match node shows the following statistics:

Physical Operation	Logical Operation
Hash Match	Right Outer Join
Actual Execution Mode	Row
Estimated Execution Mode	Row
Storage	RowScan
Number of Rows Read	233690
Actual Number of Rows for All Executions	233690
Actual Number of Batches	0
Estimated Operator Cost	1.418695 (34%)
Estimated I/O Cost	0
Estimated CPU Cost	1.418686
Estimated Subtree Cost	4.01591
Number of Executions	1
Estimated Number of Executions	1
Estimated Number of Rows Per Execution	232967
Estimated Row Size	15 B
Actual Rebinds	0
Actual Rewinds	0
Node ID	1

The 'Object' section of the tooltip for the Hash Match node shows the output list: '[InstagramDB].[dbo].[USERS].[userId], [InstagramDB].[dbo].[POSTS].[postUserId]'. The 'Output List' section also lists '[InstagramDB].[dbo].[USERS].[userId]'.

After this step is when the filtering happens for the where condition that we have specified in the query, i.e, in this step, only the values that do have postUserId as NULL as filtered and the userId for those rows make the output.



Version 3: `select userId from Users except select postUserId from POSTS`

The query is written using `except` which is just like `MINUS` for SQL Server.

For the `USERS` table, scan count is 1, number of logical reads is 2775 and number of physical reads is 3. And for the `POSTS` table, scan count is 1, number of logical reads is 371 and number of physical reads is 1. This is the same as in Versions 1 and 2. Looking at this, we can say that the query compiler executes in the same way (most optimum way) to achieve the end result, irrespective of how the query is written.

SQLQuery2.sql - DESKTOP-8AIVGG4.InstagramDB (DESKTOP-8AIVGG4(KR (55)) - Microsoft SQL Server Management Studio

File Edit View Query Project Tools Window Help

InstaDB

Object Explorer

Connect

SQLQuery2.sql - D...P-8AIVGG4(KR (55)) * x tablesAndTriggers...P-8AIVGG4(KR (56))

SQLQuery1.sql - D...P-8AIVGG4(KR (54))

queries.sql - DESK...P-8AIVGG4(KR (53))

select userId from Users
except
select postUserId from POSTS
go

150 %

Results SQL Server Execution Times:
CPU time = 0 ms, elapsed time = 0 ms.
SQL Server parse and compile time:
CPU time = 13 ms, elapsed time = 13 ms.

(217962 rows affected)

Table 'Workfile'. Scan count 0, logical reads 0, physical reads 0, page server reads 0, read-ahead reads 0, page server read-ahead reads 0, lob logic.
Table 'Worktable'. Scan count 0, logical reads 0, physical reads 0, page server reads 0, read-ahead reads 0, page server read-ahead reads 0, lob logic.
Table 'USERS'. Scan count 1, logical reads 2775, physical reads 3, page server reads 0, read-ahead reads 2787, page server read-ahead reads 0, lob logic.
Table 'POSTS'. Scan count 1, logical reads 371, physical reads 1, page server reads 0, read-ahead reads 363, page server read-ahead reads 0, lob logic.

(1 row affected)

SQL Server Execution Times:
CPU time = 141 ms, elapsed time = 1162 ms.
SQL Server parse and compile time:
CPU time = 0 ms, elapsed time = 0 ms.

SQL Server Execution Times:
CPU time = 0 ms, elapsed time = 0 ms.

Completion time: 2020-09-20T11:03:53.4139072+05:30

150 %

Query executed successfully.

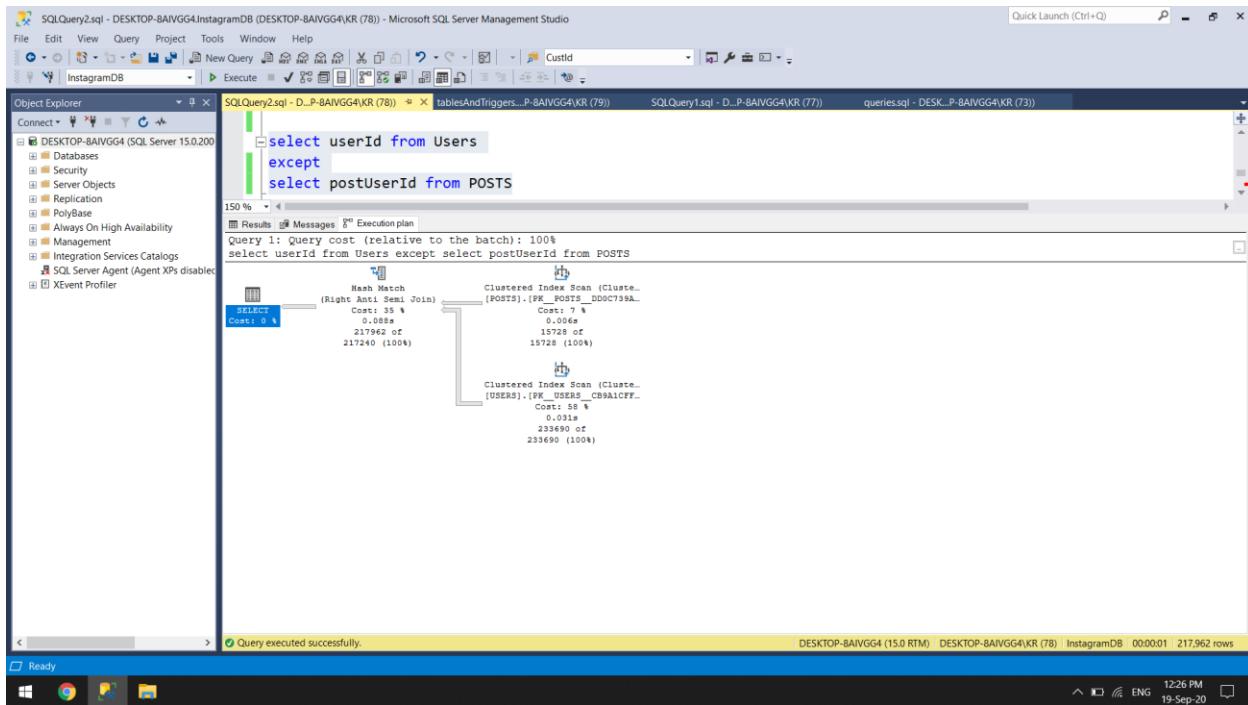
LN 26 Col 1 Ch 1 INS

DESKTOP-8AIVGG4 (15.0 RTM) DESKTOP-8AIVGG4(KR (55)) InstagramDB 00:00:01 217,962 rows

Ready

Windows 1104 AM ENG 20-Sep-20

This is the execution plan of the query. It looks exactly like the one in Version1.



SQLQuery2.sql - DESKTOP-8AIVGG4.InstagramDB (DESKTOP-8AIVGG4\KR (78)) - Microsoft SQL Server Management Studio

```

File Edit View Query Project Tools Window Help
New Query Execute
Object Explorer
Connect
DESKTOP-8AIVGG4 (SQL Server 15.0.2000)
  Databases
  Security
  Server Objects
  Replication
  PolyBase
  Always On High Availability
  Management
  Integration Services Catalogs
  SQL Server Agent (Agent XPs disabled)
  XEvent Profiler
  InstagramDB
  execute
  tablesAndTriggers_P-8AIVGG4(KR(79))
  SQLQuery1.sql - D_P-8AIVGG4(KR(77))
  queries.sql - DESK_P-8AIVGG4(KR(73))

SELECT
    userId
FROM
    Users
EXCEPT
    SELECT
        postUserId
FROM
    POSTS

```

Results Messages Execution plan

Query 1: Query cost (relative to the batch): 100%

```

select userid from users except select postUserId from POSTS

```

Physical Operation

- Logical Operation: Clustered Index Scan
- Actual Execution Mode: Row
- Estimated Execution Mode: Row

Storage

- Number of Rows Read: 15728
- Actual Number of Rows for All Executions: 15728
- Actual Number of Batches: 0
- Estimated Operator Cost: 0.293175 (7%)
- Estimated I/O Cost: 0.275718
- Estimated CPU Cost: 0.0174578
- Estimated Subtree Cost: 0.293175
- Number of Executions: 1
- Estimated Number of Executions: 1
- Estimated Number of Rows to be Read: 15728
- Estimated Number of Rows Per Execution: 15728
- Estimated Row Size: 11 B
- Actual Rebinds: 0
- Actual Rewinds: 0
- Ordered: False
- Node ID: 1

Object

- [InstagramDB].[dbo].[POSTS].[PK_POSTS_DD0C739A8B15966]

Output List

- [InstagramDB].[dbo].[POSTS].postUserId

Query executed successfully.

Ready

DESKTOP-8AIVGG4(KR(78)) InstagramDB 00:00:01 217,962 rows

12:26 PM ENG 19-Sep-20

SQLQuery2.sql - DESKTOP-8AIVGG4.InstagramDB (DESKTOP-8AIVGG4\KR (78)) - Microsoft SQL Server Management Studio

```

File Edit View Query Project Tools Window Help
New Query Execute
Object Explorer
Connect
DESKTOP-8AIVGG4 (SQL Server 15.0.2000)
  Databases
  Security
  Server Objects
  Replication
  PolyBase
  Always On High Availability
  Management
  Integration Services Catalogs
  SQL Server Agent (Agent XPs disabled)
  XEvent Profiler
  InstagramDB
  execute
  tablesAndTriggers_P-8AIVGG4(KR(79))
  SQLQuery1.sql - D_P-8AIVGG4(KR(77))
  queries.sql - DESK_P-8AIVGG4(KR(73))

SELECT
    userId
FROM
    Users
EXCEPT
    SELECT
        postUserId
FROM
    POSTS

```

Results Messages Execution plan

Query 1: Query cost (relative to the batch): 100%

```

select userid from users except select postUserId from POSTS

```

Physical Operation

- Logical Operation: Clustered Index Scan
- Actual Execution Mode: Row
- Estimated Execution Mode: Row

Storage

- Number of Rows Read: 15728
- Actual Number of Rows for All Executions: 15728
- Actual Number of Batches: 0
- Estimated Operator Cost: 2.30404 (58%)
- Estimated I/O Cost: 2.04683
- Estimated CPU Cost: 0.257216
- Estimated Subtree Cost: 2.30404
- Number of Executions: 1
- Estimated Number of Executions: 1
- Estimated Number of Rows to be Read: 233690
- Estimated Number of Rows Per Execution: 233690
- Estimated Row Size: 11 B
- Actual Rebinds: 0
- Actual Rewinds: 0
- Ordered: False
- Node ID: 2

Object

- [InstagramDB].[dbo].[USERS].[PK_USERS_CB9A1CF004D37AB]

Output List

- [InstagramDB].[dbo].[USERS].userId

Query executed successfully.

Ready

DESKTOP-8AIVGG4(KR(78)) InstagramDB 00:00:01 217,962 rows

12:26 PM ENG 19-Sep-20

The screenshot shows the Microsoft SQL Server Management Studio interface. In the Object Explorer, the database 'InstagramDB' is selected. In the center pane, a query window displays the following T-SQL code:

```

SQLQuery2.sql - DESKTOP-8AIVGG4.InstagramDB (DESKTOP-8AIVGG4\KR (78)) - Microsoft SQL Server Management Studio
File Edit View Query Project Tools Window Help
New Query Execute
SQLQuery2.sql - D...P-8AIVGG4\KR (78) tablesAndTriggers...P-8AIVGG4\KR (79) SQLQuery1.sql - D...P-8AIVGG4\KR (77) queries.sql - DESK...P-8AIVGG4\KR (73)
Quick Launch (Ctrl+Q) P X
Object Explorer
Connect ▾
DESKTOP-8AIVGG4 (SQL Server 15.0.200)
Databases Security Server Objects Replication PolyBase Always On High Availability Management Integration Services Catalogs SQL Server Agent (Agent XPs disabled) XEvent Profiler
SQLQuery2.sql - D...P-8AIVGG4\KR (78) tablesAndTriggers...P-8AIVGG4\KR (79) SQLQuery1.sql - D...P-8AIVGG4\KR (77) queries.sql - DESK...P-8AIVGG4\KR (73)
Results Messages Execution plan
Query 1: Query cost (relative to the batch): 100%
select userId from Users
except
select postUserId from POSTS

```

The execution plan details the following operations:

- Hash Match (Right Anti Semi Join)**: Cost: 35 %, Rows: 217962 of 217240 (100%). Description: Use each row from the top input to build a hash table, and each row from the bottom input to probe into the hash table, outputting all matching rows.
- Clustered Index Scan (Clustered Index Scan)**: Rows: 217962 where nonclustered

Physical Operation

Logical Operation	Physical Operation
Actual Execution Mode	Row
Estimated Execution Mode	Row
Actual Number of Rows for All Executions	217962
Actual Number of Batches	0
Estimated Operator Cost	1.406645 (35%)
Estimated I/O Cost	0
Estimated CPU Cost	1.40664
Estimated Subtree Cost	4.00386
Number of Executions	1
Estimated Number of Executions	1
Estimated Number of Rows Per Execution	217240
Estimated Row Size	11 B
Actual Rebinds	0
Actual Rewinds	0
Node ID	0

Output List

```

[InstagramDB].[dbo].[USERS].userId

```

Hash Keys Probe

```

[InstagramDB].[dbo].[USERS].userId

```

At the bottom, the status bar shows: DESKTOP-8AIVGG4 (15.0 RTM) | DESKTOP-8AIVGG4\KR (78) | InstagramDB | 00:00:01 | 217,962 rows | 12:26 PM | ENG | 19-Sep-20

For the queries of this use case, more than 50% of the cost has been for the clustered index scan on USERS. The clustered index is on the primary key userId, and even our queries are based on userId, so there is no better index that we can create to reduce this cost.

USE CASE 2: UNION

The aim is to get the privacy level, caption and number of likes of users whose userId lies between 5000 and 800000 and postId lies between 7000 and 10000 in the descending order of number of likes.

Version 1: `select U.privacyLevel, P.caption, P.likes`

```
from USERS U, POSTS P
where U.userId=P.postUserId and U.userId between 5000 and 800000 and P.postId between 7000 and
10000
group by U.privacyLevel, P.caption, P.likes
order by P.likes desc
```

The visualization of this query could be that it fetches the required attributes from both the tables simultaneously, based on the given constraints, all of which are specified together.

The output of the query is shown.

The screenshot shows the Microsoft SQL Server Management Studio interface. In the Object Explorer, the database 'InstagramDB' is selected. In the center pane, a SQL file named 'SQLQuery2.sql' contains the following code:

```
--union
--get the privacy level, caption and number of likes of users whose userId lies between 5000 and 800000 and pos
select U.privacyLevel, P.caption, P.likes
from USERS U, POSTS P
where U.userId=P.postUserId and U.userId between 5000 and 800000 and P.postId between 7000 and 10000
group by U.privacyLevel, P.caption, P.likes
order by P.likes desc
go

select U.privacyLevel, P.caption, P.likes
from USERS U, POSTS P
where U.userId=P.postUserId and U.userId between 5000 and 800000 and P.postId between 7000 and 8000
group by U.privacyLevel, P.caption, P.likes
UNION
select U.privacyLevel, P.caption, P.likes
from USERS U, POSTS P
```

The 'Results' tab shows the execution plan and the resulting data:

privacyLevel	caption	likes
1		100
2	Caption	0
3		0
4	I am a caption	0

At the bottom, a message indicates: 'Query executed successfully.'

Notice for the USERS table, scan count is 0, number of logical reads is 1987 and number of physical reads is 1. And for the POSTS table, scan count is 1, number of logical reads is 15 and number of physical reads is 2.

```

--get the privacy level, caption and number of likes of users whose userId lies between 5000 and 80000 and post
select U.privacyLevel, P.caption, P.likes
from USERS U, POSTS P
where U.userId=P.postUserId and U.userId between 5000 and 80000 and P.postId between 7000 and 10000
group by U.privacyLevel, P.caption, P.likes
order by P.likes desc

```

(4 rows affected)

Table 'USERS'. Scan count 0, logical reads 0, physical reads 0, page server reads 0, read-ahead reads 0, page server read-ahead reads 0, lob logical reads 0.

Table 'POSTS'. Scan count 1, logical reads 15, physical reads 2, page server reads 0, read-ahead reads 6, page server read-ahead reads 0, lob logical reads 0.

(1 row affected)

SQL Server Execution Times:

CPU time = 0 ms, elapsed time = 0 ms.

SQL Server parse and compile time:

CPU time = 9 ms, elapsed time = 9 ms.

SQL Server Execution Times:

CPU time = 31 ms, elapsed time = 111 ms.

SQL Server parse and compile time:

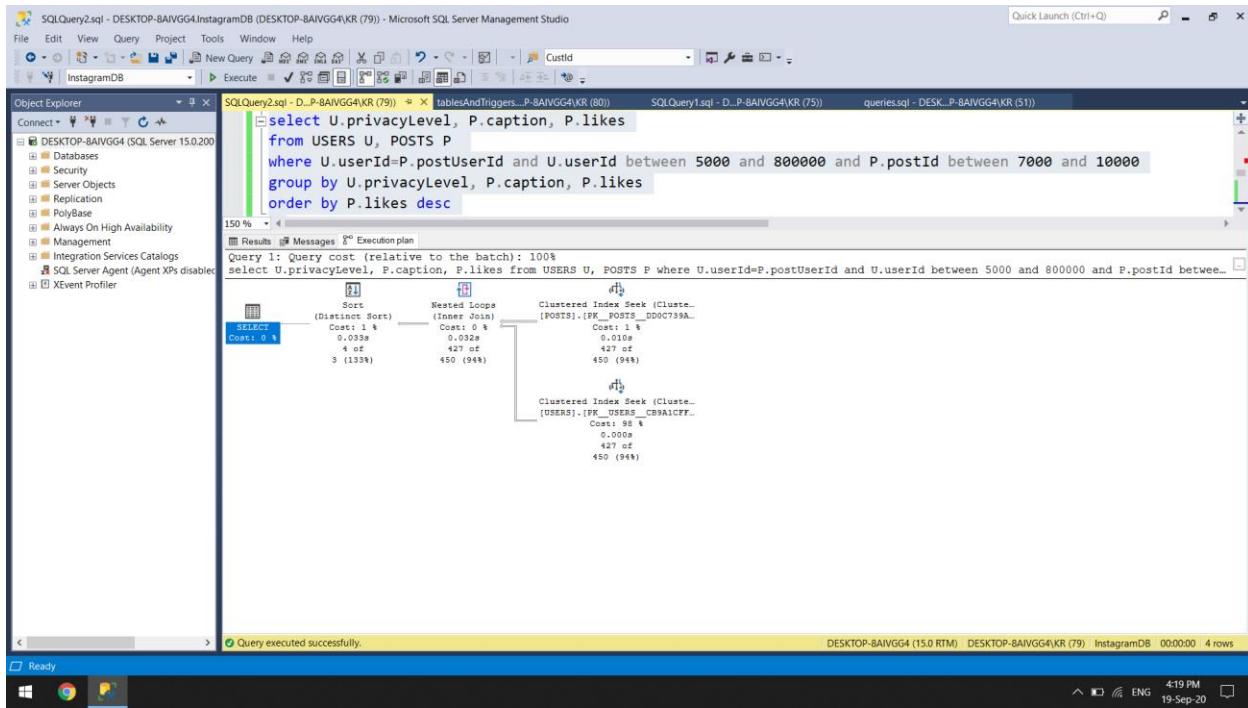
CPU time = 0 ms, elapsed time = 0 ms.

SQL Server Execution Times:

CPU time = 0 ms, elapsed time = 0 ms.

Completion time: 2020-09-20T12:43:12.0526681+05:30

This is the execution plan of the query.



A clustered index scan is performed on POSTS to seek the tuples where the primary key postId is >7000 and <10000, as specified in the query. So only those tuples that satisfy these conditions are projected in the first step.

```

SELECT U.privacyLevel, P.caption, P.likes
FROM USERS U, POSTS P
WHERE U.userId=P.postUserId AND U.userId between 5000 and 800000
      AND P.postId between 7000 and 10000
GROUP BY U.privacyLevel, P.caption, P.likes
ORDER BY P.likes desc

```

A clustered index scan is performed on USERS to seek the tuples that are currently clustered by the primary key userId. This can be seen in the seek predicates. After this seeking is done to get the rows based on the index (primary key index PK__USERS__CB9A1CFF004D37AB), filtering is done based on the condition for userId as we can see under Predicate.

```

SELECT U.privacyLevel, P.caption, P.likes
FROM USERS U, POSTS P
WHERE U.userId=P.postUserId AND U.userId between 5000 and 800000
      AND P.postId between 7000 and 10000
GROUP BY U.privacyLevel, P.caption, P.likes
ORDER BY P.likes desc

```

After the inner join, sorting is done to order the rows based on the specified attributed in the query and the output list has all the attributes we mentioned in the selList of the query.

SQLQuery2.sql - DESKTOP-8AIVGG4.InstagramDB (DESKTOP-8AIVGG4\KR (79)) - Microsoft SQL Server Management Studio

```

select U.privacyLevel, P.caption, P.likes
from USERS U, POSTS P
where U.userId=P.postUserId and U.userId between 5000 and 800000 and P.postId between 7000 and 10000
group by U.privacyLevel, P.caption, P.likes
order by P.likes desc

```

Execution plan details:

- Physical Operation:** Sort
- Logical Operation:** Distinct Sort
- Actual Execution Mode:** Row
- Estimated Execution Mode:** Row
- Actual Number of Rows for All Executions:** 4
- Actual Number of Batches:** 1
- Estimated Operator Cost:** 0.01755 (1%)
- Estimated I/O Cost:** 0.0112613
- Estimated CPU Cost:** 0.0062889
- Estimated Subtree Cost:** 1.37277
- Number of Executions:** 1
- Estimated Number of Executions:** 1
- Estimated Number of Rows Per Execution:** 2.70423
- Estimated Row Size:** 23 B
- Actual Rebinds:** 1
- Actual Rewinds:** 0
- Node ID:** 0

Output List:

- [InstagramDB].[dbo].[USERS].privacyLevel, [InstagramDB].[dbo].[POSTS].likes Descending,
- [dbo].[POSTS].caption, [InstagramDB].[dbo].[POSTS].likes Order By
- [InstagramDB].[dbo].[POSTS].likes Descending,
- [InstagramDB].[dbo].[USERS].privacyLevel Ascending,
- [InstagramDB].[dbo].[POSTS].caption Ascending

Ready

420 PM ENG 19-Sep-20

Version 2:

```

select U.privacyLevel, P.caption, P.likes
from USERS U, POSTS P
where U.userId=P.postUserId and U.userId between 5000 and 800000 and P.postId between 7000 and
8000
group by U.privacyLevel, P.caption, P.likes
UNION
select U.privacyLevel, P.caption, P.likes
from USERS U, POSTS P
where U.userId=P.postUserId and U.userId between 5000 and 800000 and P.postId between 8000 and
10000
group by U.privacyLevel, P.caption, P.likes
order by P.likes desc

```

This query uses UNION. Here, we have split the range condition for postUserId into two ranges instead of one and joined the results using the UNION operator.

The output of the query is shown. As expected, it is the same as that of Version 1.

The screenshot shows the Microsoft SQL Server Management Studio interface. In the center, there is a results grid titled 'Results' with columns 'privacyLevel', 'caption', and 'likes'. The data is as follows:

privacyLevel	caption	likes
1		100
2	Caption	0
3	Caption	0
4	I am a caption	0

At the bottom of the results pane, it says 'Query executed successfully.' Below the results pane, the status bar shows 'Ln 57 Col 3 Ch 3 INS' and the system tray shows the date and time as '19-Sep-20 42:1 PM'.

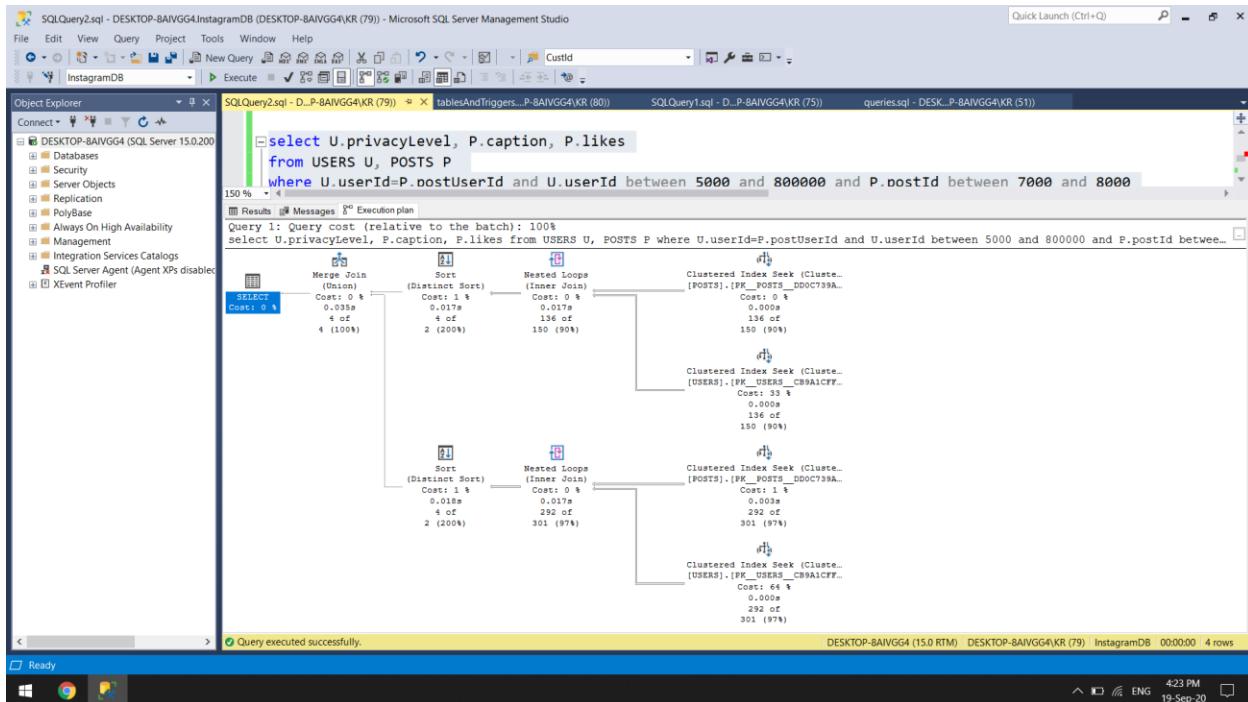
For the USERS table, scan count is 0, number of logical reads is 1985 and number of physical reads is 1. And for the POSTS table, scan count is 2, number of logical reads is 18 and number of physical reads is 2.

The screenshot shows the Microsoft SQL Server Management Studio interface with detailed execution statistics. The output includes:

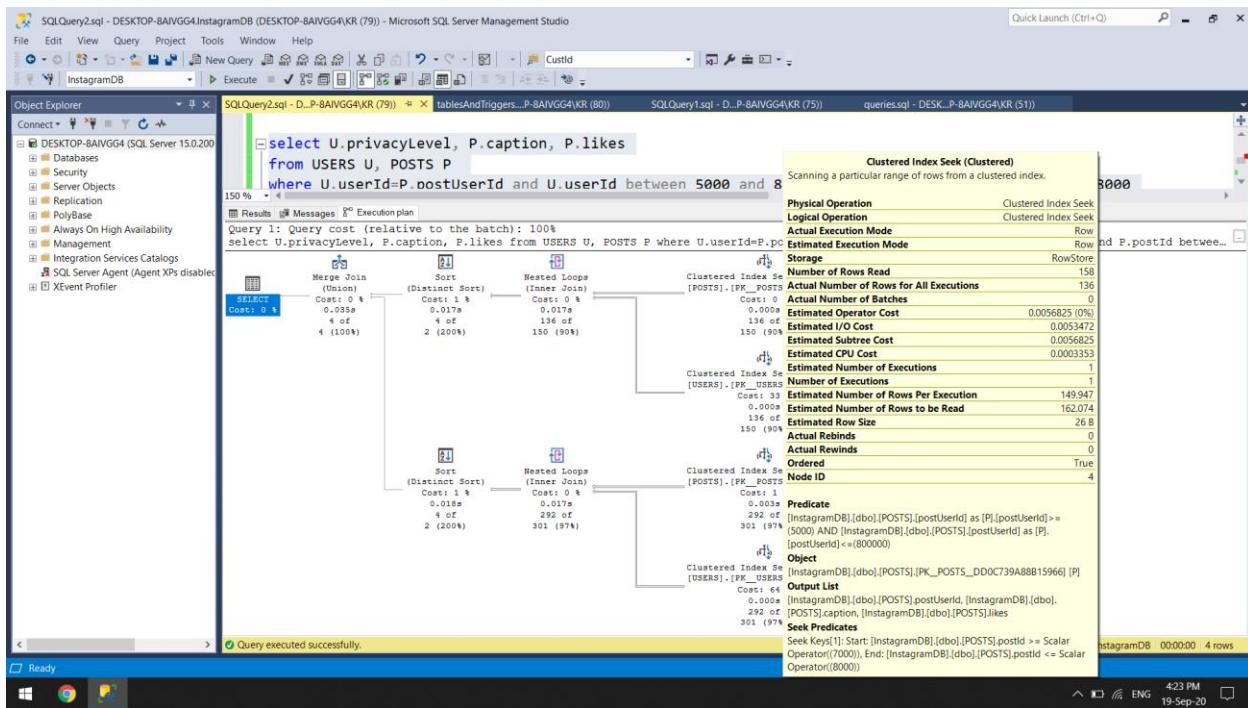
- SQL Server parse and compile time:** CPU time = 0 ms, elapsed time = 0 ms.
- SQL Server Execution Times:**
 - Table 'USERS'. Scan count 0, logical reads 1985, physical reads 1, page server reads 0, read-ahead reads 0, page server read-ahead reads 0, lob log: Table 'USERS'. Scan count 0, logical reads 1985, physical reads 1, page server reads 0, read-ahead reads 1896, page server read-ahead reads 0, lob log: Table 'POSTS'. Scan count 2, logical reads 18, physical reads 2, page server reads 0, read-ahead reads 0, page server read-ahead reads 0, lob logical
- (4 rows affected)**
- Completion time:** 2020-09-19T16:21:16.5511016+05:30

At the bottom of the results pane, it says 'Query executed successfully.' Below the results pane, the status bar shows 'Ln 57 Col 3 Ch 3 INS' and the system tray shows the date and time as '19-Sep-20 42:1 PM'.

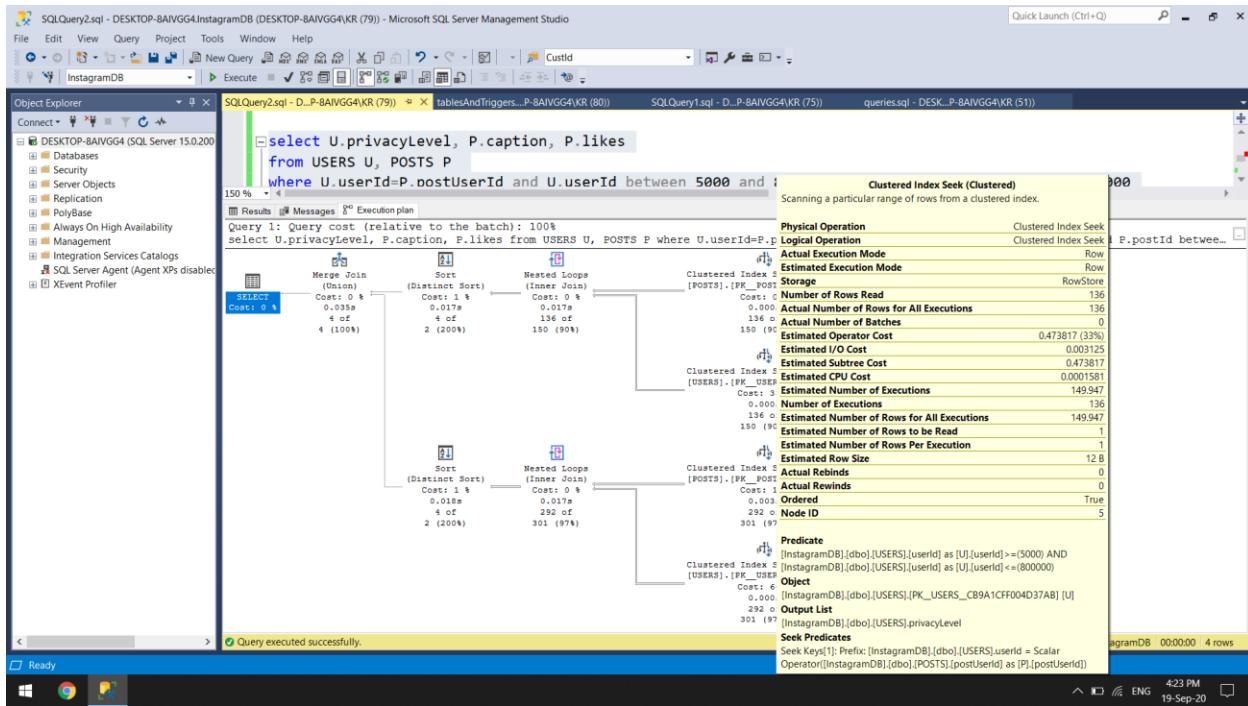
The execution plan is shown below. The same clustered index seek is performed on both the table, for both the ranges separately. The results from both the tables are joined and sorted individually and then finally merged together. The detailed specifications can be seen in the images below.



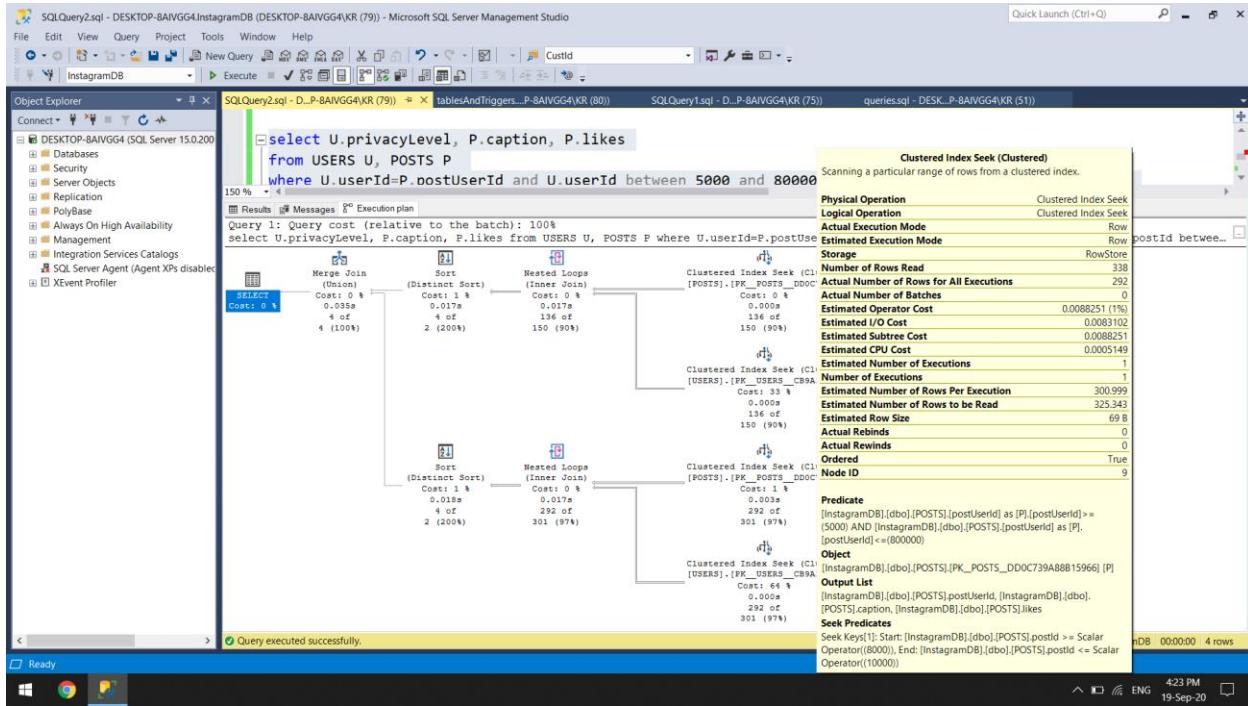
For POSTS table in the first range-



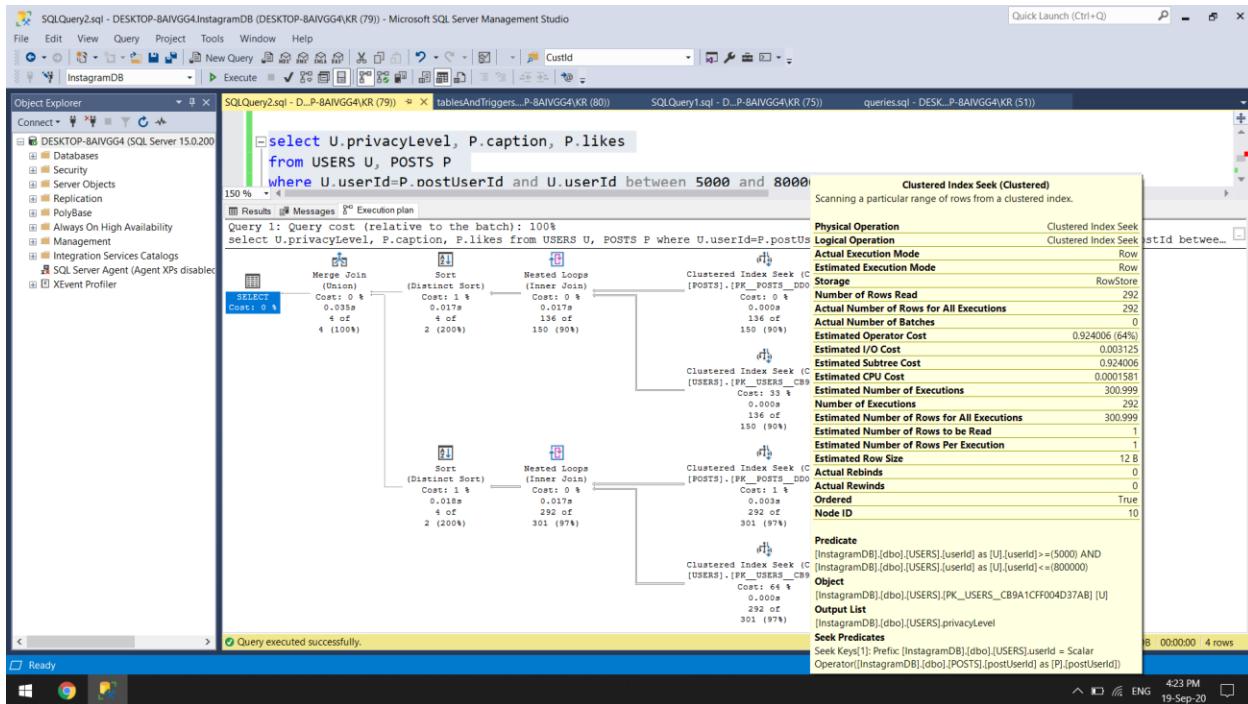
For USERS table in the first range-



For POSTS table in the second range-



For USERS table in the second range-



```

Version 3: select T.privacyLevel, T.caption, T.likes
from
(
    select U.userId, U.privacyLevel, P.caption, P.likes
    from USERS U, POSTS P
    where U.userId=P.postUserId and P.postId between 7000 and 8000
    group by U.userId, U.privacyLevel, P.caption, P.likes
    UNION
    select U.userId, U.privacyLevel, P.caption, P.likes
    from USERS U, POSTS P
    where U.userId=P.postUserId and P.postId between 8000 and 10000
    group by U.userId, U.privacyLevel, P.caption, P.likes
) T
where T.userId between 5000 and 800000
group by T.privacyLevel, T.caption, T.likes
order by T.likes desc

```

In this query, we have selected the required attributes from both the tables by giving a range condition for postId. Instead of giving the entire range at once, we have split the range into two parts- 7000 to 8000 and 8000 to 10000 and joined both the results using the UNION operator. On that result, we have specified the condition for userId and ordered the final result. This is kind of a mix of Version 1 and Version 2 to produce a very elaborate way of writing the query. It seems like the worst case scenario, however, we see that the query compiler optimizes the query to execute it in the same way as Version 2, as depicted in the images below.

The output message of the query is shown below. It is about the same as the previous two versions.

SQLQuery2.sql - DESKTOP-8AIVGG4.InstagramDB (DESKTOP-8AIVGG4(KR (55)) - Microsoft SQL Server Management Studio

File Edit View Query Project Tools Window Help

Object Explorer

Connect ▾

SQLQuery2.sql - D...P-8AIVGG4(KR (55)) X tablesAndTriggers_P-8AIVGG4(KR (56)) SQLQuery1.sql - D...P-8AIVGG4(KR (54)) queries.sql - DESK...P-8AIVGG4(KR (53))

select U.userId, U.privacyLevel, P.caption, P.likes
from USERS U, POSTS P
where U.userId=P.postUserId and P.postId between 7000 and 8000
group by U.userId, U.privacyLevel, P.caption, P.likes
UNION
select U.userId, U.privacyLevel, P.caption, P.likes

150 %

Results Execution plan
SQL Server parse and compile time:
CPU time = 0 ms, elapsed time = 0 ms.

SQL Server Execution Times:
CPU time = 0 ms, elapsed time = 0 ms.
SQL Server parse and compile time:
CPU time = 31 ms, elapsed time = 32 ms.

(4 rows affected)

Table 'Worktable'. Scan count 0, logical reads 0, physical reads 0, page server reads 0, read-ahead reads 0, page server read-ahead reads 0, lob log.
Table 'USERS'. Scan count 0, logical reads 1979, physical reads 1, page server reads 0, read-ahead reads 1896, page server read-ahead reads 0, lob lo.
Table 'POSTS'. Scan count 2, logical reads 18, physical reads 2, page server reads 0, read-ahead reads 6, page server read-ahead reads 0, lob logical

(1 row affected)

SQL Server Execution Times:
CPU time = 31 ms, elapsed time = 165 ms.
SQL Server parse and compile time:
CPU time = 0 ms, elapsed time = 0 ms.

SQL Server Execution Times:
CPU time = 0 ms, elapsed time = 0 ms.

Completion time: 2020-09-20T13:10:14.2723505+05:30

150 %

Query executed successfully.

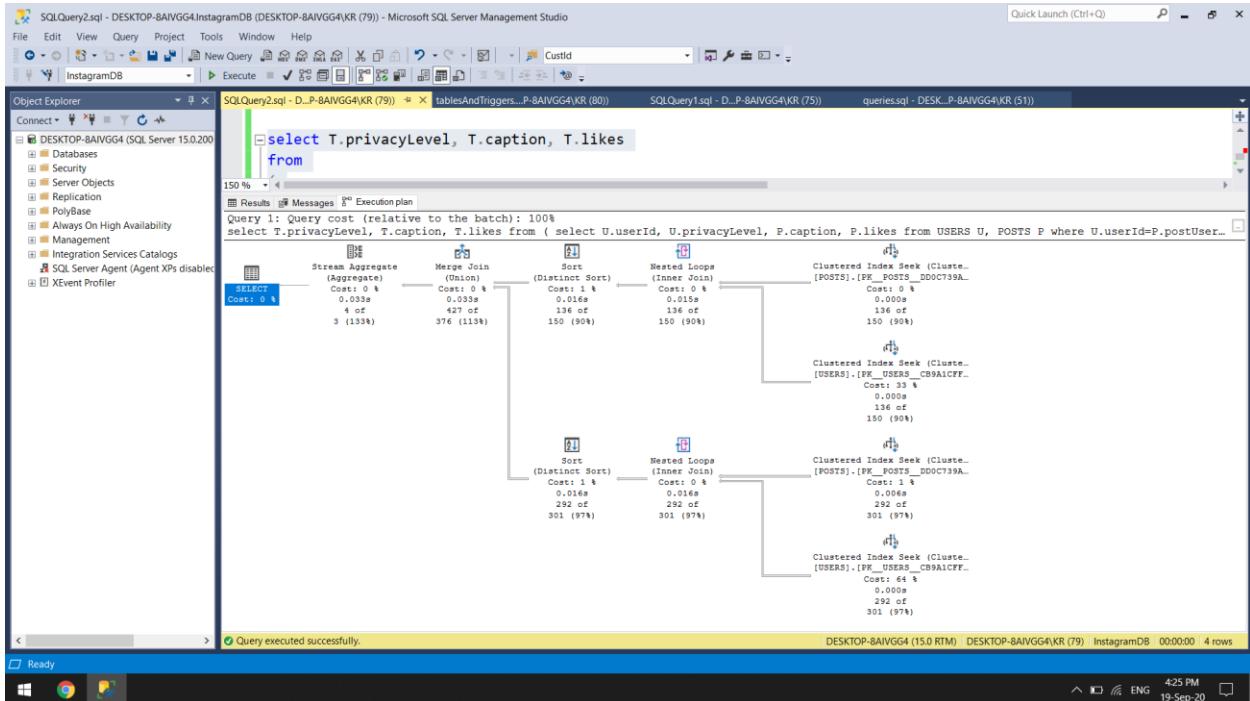
DESKTOP-8AIVGG4 (15.0 RTM) DESKTOP-8AIVGG4(KR (55)) InstagramDB 00:00:00 4 rows

Item(s) Saved

In 25 Col 1 Ch 1 INS

1:10 PM ENG 20-Sep-20

This is the execution plan of the query. It is the same as that of Version 2, just that we have a Steam Aggregate step before the final output because along with the group by clauses on table U and table P, we have specified the group by clause on the table T which is the union of tables U and P, whereas in Version 2, the group by clause was specified only on U and P. In effect, the execution plan has not changed from that of Version 2.



Use case 3: Multi-table join (4 tables)

Until now, we looked only at two tables- USERS and POSTS. For the join operation, we create two more tables- TAGGED and POSTS_PICTURE.

The screenshot shows the Microsoft SQL Server Management Studio interface. In the Object Explorer, the database 'InstagramDB' is selected. In the center pane, two CREATE TABLE statements are being run:

```
CREATE TABLE TAGGED (
    postId      INT          NOT NULL,
    taggedUserId INT          NOT NULL,
    PRIMARY KEY(taggedUserId, postId),
    FOREIGN KEY(postId) REFERENCES POSTS(postId),
    FOREIGN KEY(taggedUserId) REFERENCES USERS(userId)
    ON DELETE CASCADE ON UPDATE CASCADE
) on FG1;

CREATE TABLE POSTS_PICTURE (
    postId      INT          NOT NULL,
    pictureId  INT          NOT NULL,
    PRIMARY KEY(postId, pictureId),
```

Below the code, the Messages pane displays the execution results:

```
SQL Server parse and compile time:
CPU time = 0 ms, elapsed time = 0 ms.

SQL Server Execution Times:
CPU time = 0 ms, elapsed time = 0 ms.
SQL Server parse and compile time:
CPU time = 0 ms, elapsed time = 0 ms.

SQL Server Execution Times:
CPU time = 0 ms, elapsed time = 3 ms.
SQL Server parse and compile time:
CPU time = 0 ms, elapsed time = 0 ms.
```

The status bar at the bottom right indicates the session is running on 'DESKTOP-BAIVGG4 (15.0 RTM)' at '4:36 PM' on '19-Sep-20'.

This screenshot shows the Microsoft SQL Server Management Studio interface with the same setup as the previous one. The Object Explorer shows the 'InstagramDB' database. The central pane contains the same two CREATE TABLE statements:

```
CREATE TABLE TAGGED (
    postId      INT          NOT NULL,
    taggedUserId INT          NOT NULL,
    PRIMARY KEY(taggedUserId, postId),
    FOREIGN KEY(taggedUserId) REFERENCES USERS(userId)
    ON DELETE CASCADE ON UPDATE CASCADE
) on FG1;

CREATE TABLE POSTS_PICTURE (
    postId      INT          NOT NULL,
    pictureId  INT          NOT NULL,
    PRIMARY KEY(postId, pictureId),
    FOREIGN KEY(postId) REFERENCES POSTS(postId)
    ON DELETE CASCADE ON UPDATE CASCADE
) on FG1;
```

The Messages pane shows the execution results, which differ from the first screenshot due to a longer execution time for the second query:

```
SQL Server parse and compile time:
CPU time = 0 ms, elapsed time = 0 ms.

SQL Server Execution Times:
CPU time = 0 ms, elapsed time = 0 ms.
SQL Server parse and compile time:
CPU time = 0 ms, elapsed time = 0 ms.

SQL Server Execution Times:
CPU time = 15 ms, elapsed time = 4 ms.
SQL Server parse and compile time:
CPU time = 0 ms, elapsed time = 0 ms.
```

The status bar at the bottom right indicates the session is running on 'DESKTOP-BAIVGG4 (15.0 RTM)' at '4:37 PM' on '19-Sep-20'.

We populate these two tables with a couple of thousands of rows with random values.

```

SQLQuery2.sql - DESKTOP-8AIVGG4.InstagramDB (DESKTOP-8AIVGG4(KR (65)) - Microsoft SQL Server Management Studio
File Edit View Query Project Tools Window Help
New Query Execute
SQLQuery2.sql - D...P-8AIVGG4(KR (65)) tablesAndTriggers_P-8AIVGG4(KR (66)) SQLQuery1.sql - D...P-8AIVGG4(KR (64)) queries.sql - DESKTOP-8AIVGG4(KR (60))
Object Explorer
Connect ▾
DESKTOP-8AIVGG4 (SQL Server 15.0.200)
Databases Security Server Objects Replication PolyBase Always On High Availability Management Integration Services Catalogs SQL Server Agent (Agent XPs disabled) XEvent Profiler
SQLQuery2.sql - D...P-8AIVGG4(KR (65)) tablesAndTriggers_P-8AIVGG4(KR (66)) SQLQuery1.sql - D...P-8AIVGG4(KR (64)) queries.sql - DESKTOP-8AIVGG4(KR (60))
set @i = 1
while @i <= 1000000
begin
    select @i = RAND()*100000
    set @pi = (select top 1 postId from Posts order by newid())
    insert into posts_picture (postId, pictureId)
    select @pi, @i
    set @i = @i + 1
end
go

```

(1 row affected)
(1 row affected)

150 % ▾

Ready Ln 145 Col 1 Ch 1 INS 5:31 PM ENG 19-Sep-20

We successfully insert 16016 rows into the TAGGED table and 147932 rows into POSTS_PICTURE.

```

SQLQuery2.sql - DESKTOP-8AIVGG4.InstagramDB (DESKTOP-8AIVGG4(KR (65)) - Microsoft SQL Server Management Studio
File Edit View Query Project Tools Window Help
New Query Execute
SQLQuery2.sql - D...P-8AIVGG4(KR (65)) tablesAndTriggers_P-8AIVGG4(KR (66)) SQLQuery1.sql - D...P-8AIVGG4(KR (64)) queries.sql - DESKTOP-8AIVGG4(KR (60))
Object Explorer
Connect ▾
DESKTOP-8AIVGG4 (SQL Server 15.0.200)
Databases Security Server Objects Replication PolyBase Always On High Availability Management Integration Services Catalogs SQL Server Agent (Agent XPs disabled) XEvent Profiler
SQLQuery2.sql - D...P-8AIVGG4(KR (65)) tablesAndTriggers_P-8AIVGG4(KR (66)) SQLQuery1.sql - D...P-8AIVGG4(KR (64)) queries.sql - DESKTOP-8AIVGG4(KR (60))
set @tui = (select top 1 userId from Users order by newid())
set @pi = (select top 1 postId from Posts order by newid())
insert into TAGGED(postId, taggedUserId)
select @pi, @tui
set @i = @i + 1
end
go
select * from tagged
commit transaction T1

```

postId	taggedUserId
1	2899 2
2	5017 25
3	65546 50
4	84729 51
5	38319 93
6	8274 95
7	60598 96
8	23066 99
9	88880 120
10	26625 125
11	74386 133
12	86724 142
13	100000 178
14	45943 184
15	71988 185
16	16916 199
17	68371 216

Results Messages

Ready Ln 118 Col 22 Ch 22 INS 5:18 PM ENG 19-Sep-20

The screenshot shows the Microsoft SQL Server Management Studio interface. In the center, there is a query window titled 'SQLQuery2.sql - DESKTOP-8AIVGG4.InstagramDB (DESKTOP-8AIVGG4\KR (65)) - Microsoft SQL Server Management Studio'. The window contains the following T-SQL code:

```

select @i = RAND()*100000
set @pi = (select top 1 postId from Posts order by newid())
insert into posts_picture (postId, pictureId)
select @pi, @i

set @i = @i + 1
end
go

select * from POSTS_PICTURE;

```

Below the code, the results of the execution are displayed in a table:

postId	pictureId
1	4536
2	6992
3	8053
4	14908
5	17318
6	30479
7	30592
8	31953
9	40056
10	41228
11	41591
12	4453
13	70390
14	71463
15	77436
16	98113
17	99123

At the bottom of the window, a message says 'Query executed successfully.' and shows the status bar with 'DESKTOP-8AIVGG4 (15.0 RTM) | DESKTOP-8AIVGG4\KR (65) | InstagramDB | 00:00:00 | 147,932 rows'.

The aim of the join query is to join the 4 tables based on certain conditions. We try specifying conditions on attributes from different tables and see how the execution plan changes. The point to observe in every execution plan is that the tables to join are selected such that the number of rows is the least possible at that particular step.

Version 1: `select U.username, P.likes, PP.pictureId, T.postId, T.taggedUserId
from USERS U, POSTS P, POSTS_PICTURE PP, TAGGED T
where U.userId between 5000 and 800000 and
U.userId = P.postUserId and
P.postId = PP.postId and
P.postId = T.postId
order by 2 desc`

The below query specifies the condition on the primary key, userId, of the USERS table.

```

set nocount off

--join
select U.username, P.likes, PP.pictureId, T.postId, T.taggedUserId
from USERS U, POSTS P, POSTS_PICTURE PP, TAGGED T
where U.userId between 5000 and 80000 and
    U.userId = P.postUserId and
    P.postId = PP.postId and
    P.postId = T.postId
order by 2 desc
go

```

Results Messages

username	likes	pictureId	postId	taggedUserId
45839ABC	100	2241	20369	335988
46053ABC	100	4885	16996	792929
47451ABC	100	2768	7075	280490
45836ABC	100	1738	8846	21736
41889ABC	100	10779	16699	448315
46467ABC	100	488	28958	77928
40377ABC	100	26198	32247	297034
47998ABC	100	7348	6295	93177
48439ABC	100	2241	20369	287718
46053ABC	100	5068	16996	792929
47451ABC	100	15987	7075	280490
45836ABC	100	10779	16699	448315
41889ABC	100	10779	16699	448315
46467ABC	100	488	28958	77928
40377ABC	100	38584	32247	297034
47998ABC	100	10929	6295	93177
48439ABC	100	2241	20369	376833

Query executed successfully.

The following is the message we get for this query.

```

set nocount off

--join
select U.username, P.likes, PP.pictureId, T.postId, T.taggedUserId

```

Results Messages Execution plan

SQL Server parse and compile time:
CPU time = 0 ms, elapsed time = 0 ms.

SQL Server Execution Times:
CPU time = 0 ms, elapsed time = 0 ms.
SQL Server parse and compile time:
CPU time = 15 ms, elapsed time = 22 ms.

(129295 rows affected)

Table 'POSTS'. Scan count 9, logical reads 1108, physical reads 1, page server reads 0, read-ahead reads 363, page server read-ahead reads 0, lob log
Table 'USERS'. Scan count 9, logical reads 2521, physical reads 4, page server reads 0, read-ahead reads 2444, page server read-ahead reads 0, lob log
Table 'Workfile'. Scan count 0, logical reads 0, physical reads 0, page server reads 0, read-ahead reads 0, page server read-ahead reads 0, lob log
Table 'Worktable'. Scan count 0, logical reads 0, physical reads 0, page server reads 0, read-ahead reads 0, page server read-ahead reads 0, lob log
Table 'TAGGED'. Scan count 9, logical reads 154, physical reads 1, page server reads 0, read-ahead reads 44, page server read-ahead reads 0, lob log
Table 'POSTS_PICTURE'. Scan count 9, logical reads 1980, physical reads 2, page server reads 0, read-ahead reads 481, page server read-ahead reads 0,

(1 row affected)

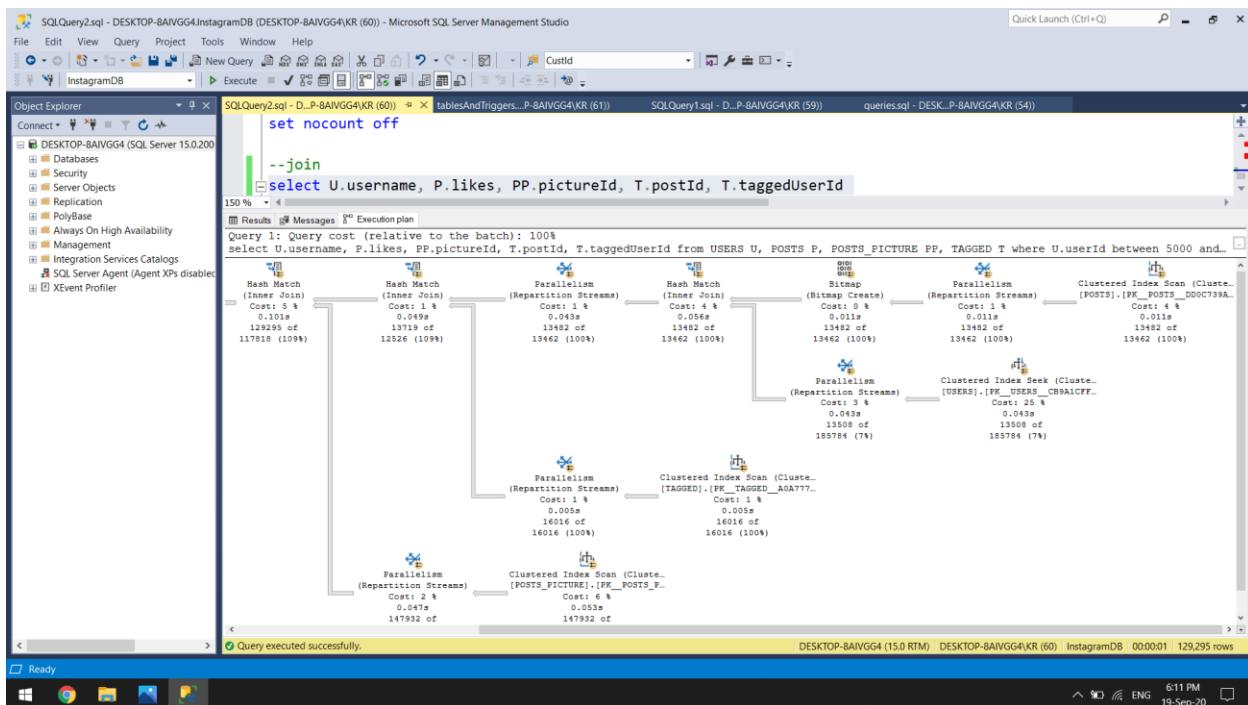
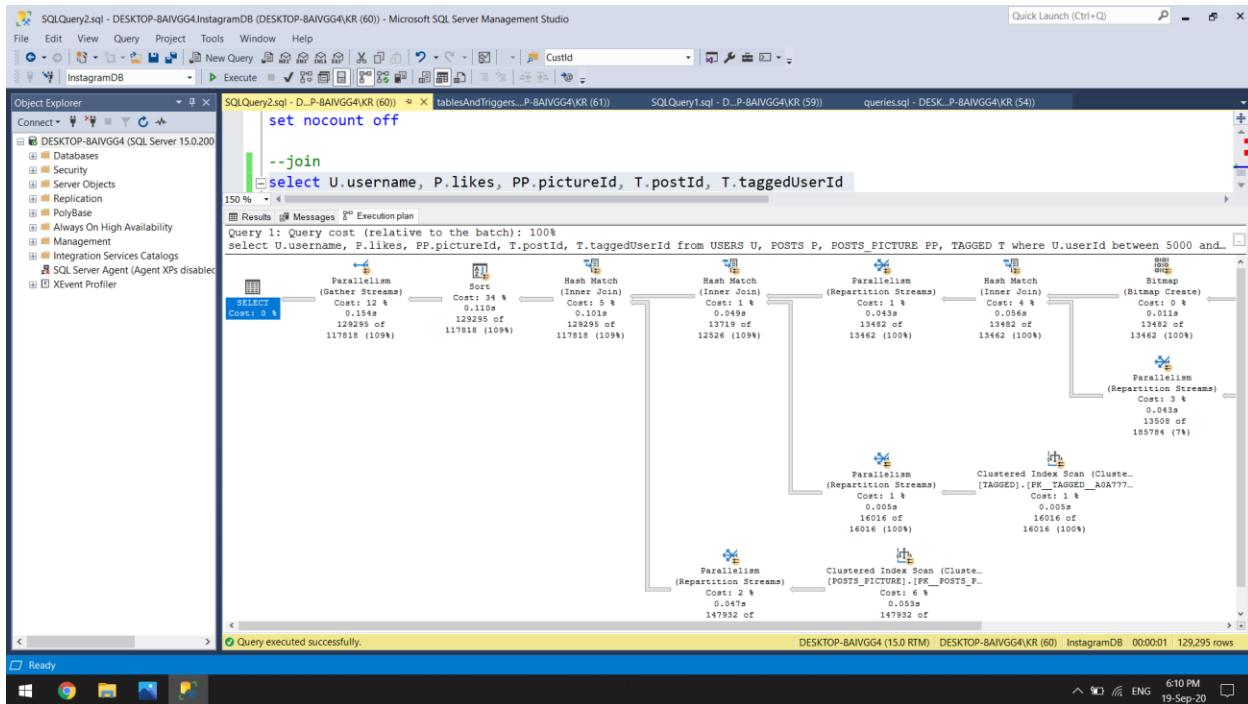
SQL Server Execution Times:
CPU time = 360 ms, elapsed time = 1037 ms.
SQL Server parse and compile time:
CPU time = 0 ms, elapsed time = 0 ms.

SQL Server Execution Times:
CPU time = 0 ms, elapsed time = 0 ms.

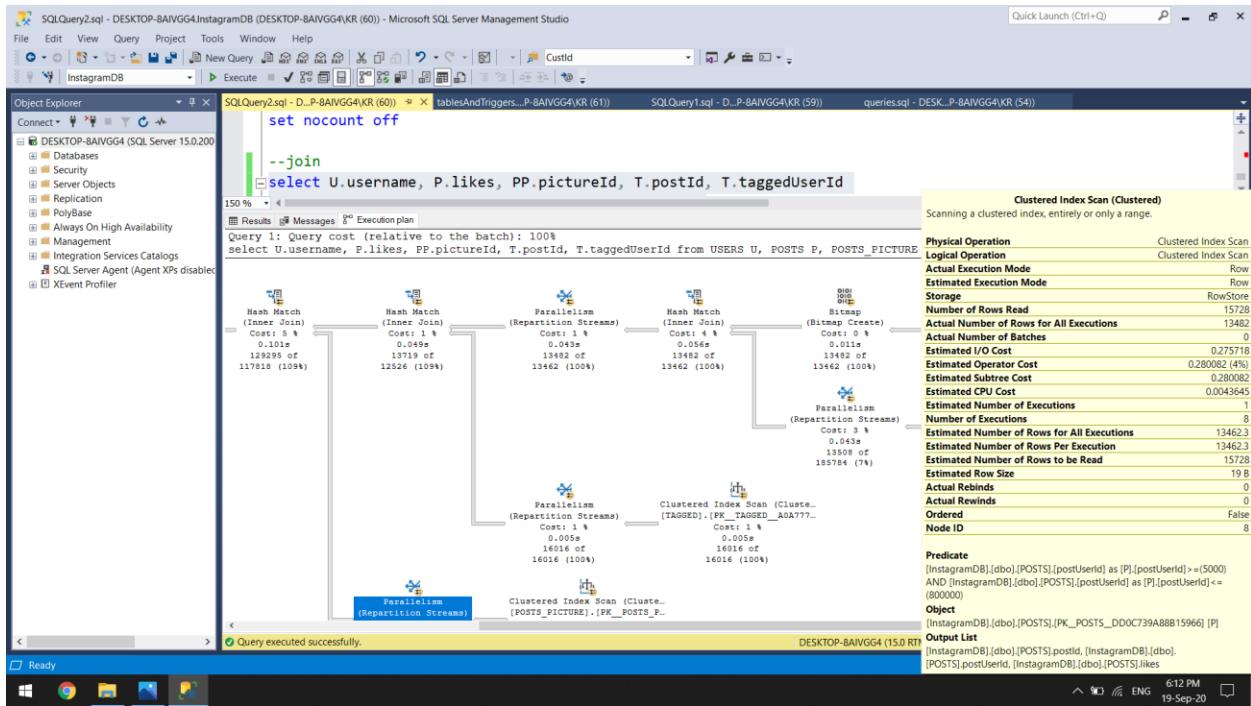
Completion time: 2020-09-19T18:10:27.9311230+05:30

Query executed successfully.

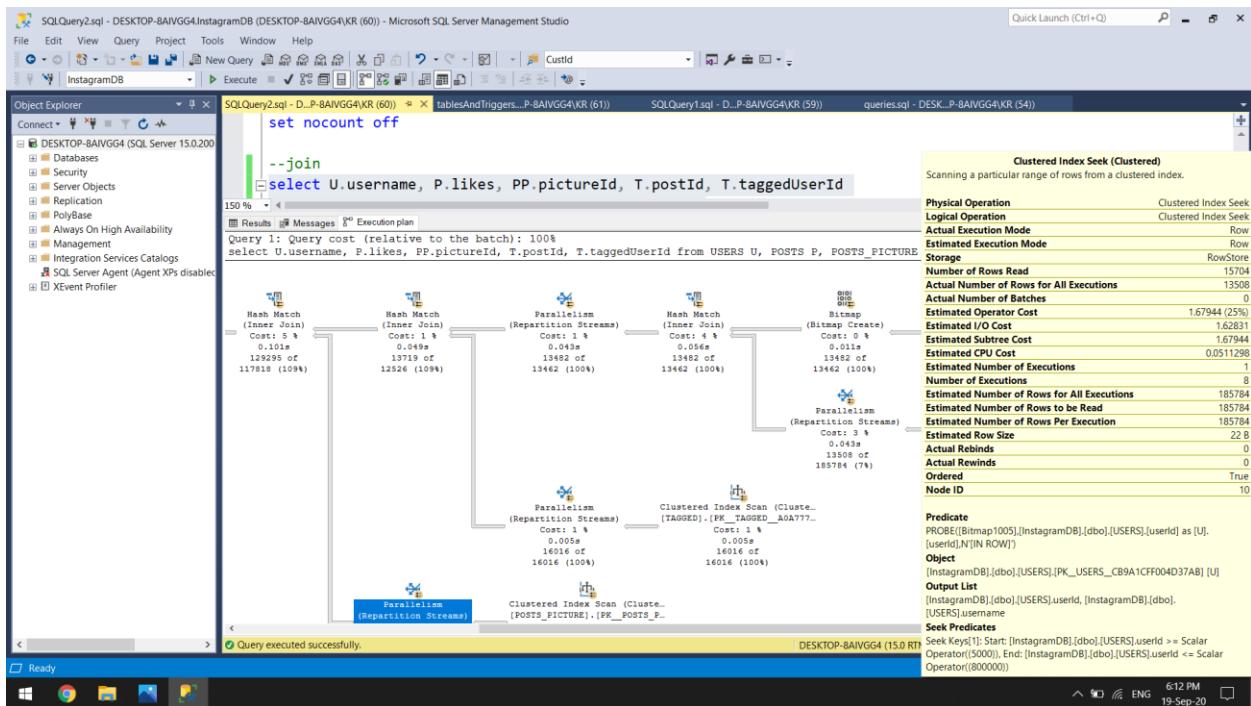
This is the execution plan for the query.



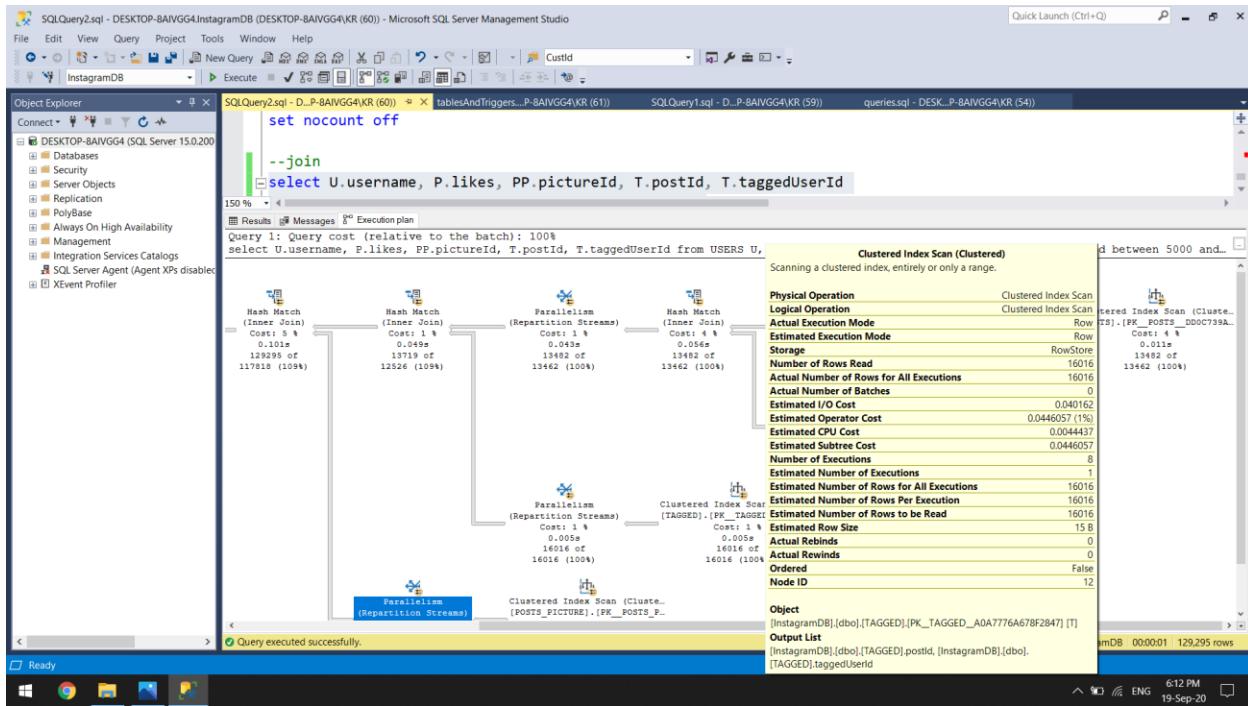
First, 13482 rows are fetched from POSTS.



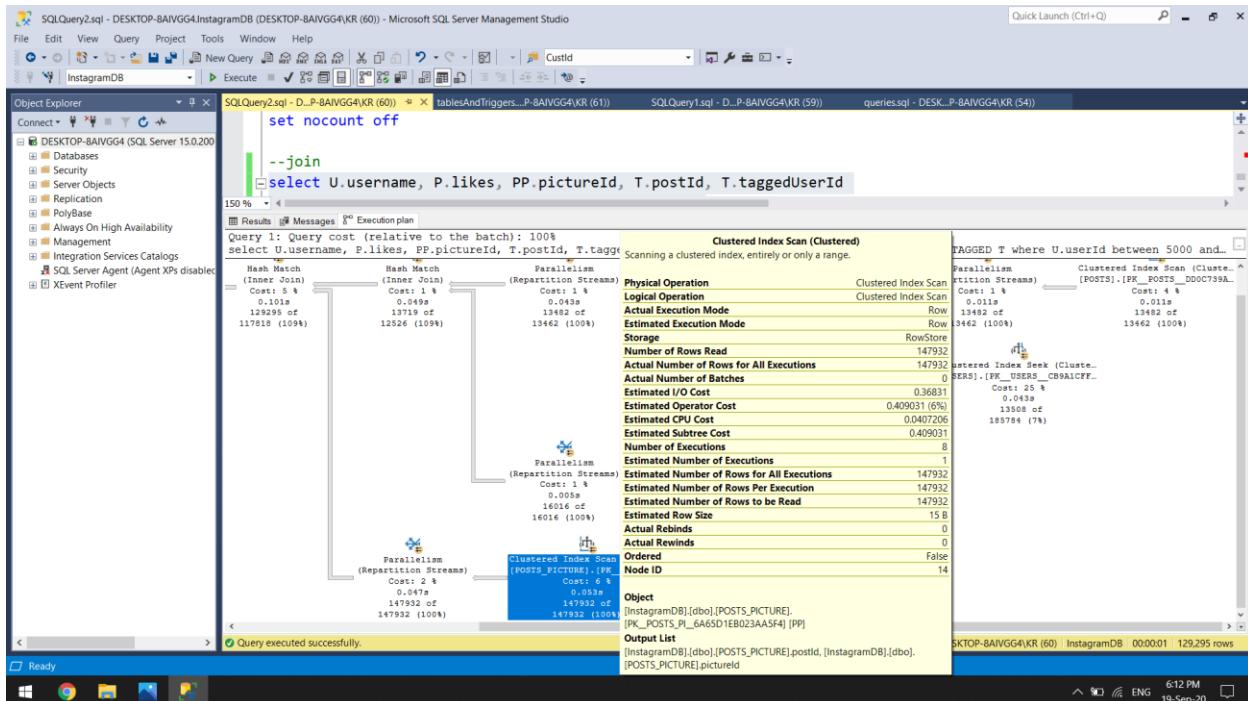
The second table chosen is USERS and 13508 rows (more than that of the previous table) are fetched from it.



The third table chosen is TAGGED and 16016 rows (more than that of the previous table) are fetched from it.



The fourth and final table for the join is POSTS_PICTURE. 147932 rows (more than that of the previous table) are fetched here.



Version 2: `select U.username, P.likes, PP.pictureId, T.postId, T.taggedUserId`
from USERS U, POSTS P, POSTS_PICTURE PP, TAGGED T
where

```

U.userId = P.postUserId and
P.postId = PP.postId and
PP.postId > 20000 and
P.postId = T.postId
order by 2 desc

```

Here, the condition is on postId of POSTS_PICTURE (we have removed the previous condition on userId of USERS).

```

go
select U.username, P.likes, PP.pictureId, T.postId, T.taggedUserId
from USERS U, POSTS P, POSTS_PICTURE PP, TAGGED T
where
    U.userId = P.postUserId and
    P.postId = PP.postId and
    PP.postId > 20000 and
    P.postId = T.postId
order by 2 desc

```

username	likes	pictureId	postId	taggedUserId
1895ABC	200	27950	29412	89256
1717ABC	200	3715	24597	285436
1409ABC	200	40473	57315	93980
1454ABC	200	2688	23035	78737
1607ABC	200	45666	42099	478550
1502ABC	200	2162	4158	974555
1045ABC	200	253	24070	477752
1105ABC	200	27950	28472	4600
9185ABC	200	27950	289725	
1717ABC	200	8644	24597	285436
1409ABC	200	40473	57315	78306
1454ABC	200	2688	23035	22343
1607ABC	200	57264	42099	478550
1502ABC	200	2162	4158	921280
1045ABC	200	6857	24070	477752
1105ABC	200	16870	28472	326443
1895ABC	200	27950	29412	922565
1717ABC	200	14948	24597	285436
1409ABC	200	51409	57315	93980
1454ABC	200	5166	23035	78737

The following is the message output for the query.

SQLQuery2.sql - DESKTOP-8AIVGG4.InstagramDB (DESKTOP-8AIVGG4(KR (60)) - Microsoft SQL Server Management Studio

File Edit View Query Project Tools Window Help

New Query Save All Close Connect Object Explorer

InstaGramDB

Execute

CustId

SQLQuery2.sql - D:\P-8AIVGG4(KR (60)) tablesAndTriggers...P-8AIVGG4(KR (61)) SQLQuery1.sql - D:\P-8AIVGG4(KR (59)) queries.sql - DESKTOP-8AIVGG4(KR (54))

go

```
select U.username, P.likes, PP.pictureId, T.postId, T.taggedUserId
from USERS U, POSTS P, POSTS_PICTURE PP, TAGGED T
```

Results Messages Execution plan

SQL Server parse and compile time:
CPU time = 0 ms, elapsed time = 0 ms.

SQL Server Execution Times:
CPU time = 0 ms, elapsed time = 0 ms.

SQL Server parse and compile time:
CPU time = 16 ms, elapsed time = 23 ms.

(120908 rows affected)

Table 'POSTS'. Scan count 9, logical reads 892, physical reads 1, page server reads 0, read-ahead reads 297, page server read-ahead reads 0, lob log: 0, lob read: 0, lob update: 0, lob delete: 0
Table 'USERS'. Scan count 9, logical reads 2921, physical reads 3, page server reads 0, read-ahead reads 2787, page server read-ahead reads 0, lob log: 0, lob read: 0, lob update: 0, lob delete: 0
Table 'Workfile'. Scan count 0, logical reads 0, physical reads 0, page server reads 0, read-ahead reads 0, page server read-ahead reads 0, lob log: 0, lob read: 0, lob update: 0, lob delete: 0
Table 'Worktable'. Scan count 0, logical reads 0, physical reads 0, page server reads 0, read-ahead reads 0, page server read-ahead reads 0, lob log: 0, lob read: 0, lob update: 0, lob delete: 0
Table 'TAGGED'. Scan count 9, logical reads 154, physical reads 1, page server reads 0, read-ahead reads 44, page server read-ahead reads 0, lob log: 0, lob read: 0, lob update: 0, lob delete: 0
Table 'POSTS_PICTURE'. Scan count 9, logical reads 1572, physical reads 2, page server reads 0, read-ahead reads 451, page server read-ahead reads 0, lob log: 0, lob read: 0, lob update: 0, lob delete: 0

(1 row affected)

SQL Server Execution Times:
CPU time = 295 ms, elapsed time = 957 ms.

SQL Server parse and compile time:
CPU time = 0 ms, elapsed time = 0 ms.

SQL Server Execution Times:
CPU time = 0 ms, elapsed time = 0 ms.

Completion time: 2020-09-19T18:15:41.7552564+05:30

150 %

Query executed successfully.

LN 171 Col 3 Ch 3 INS

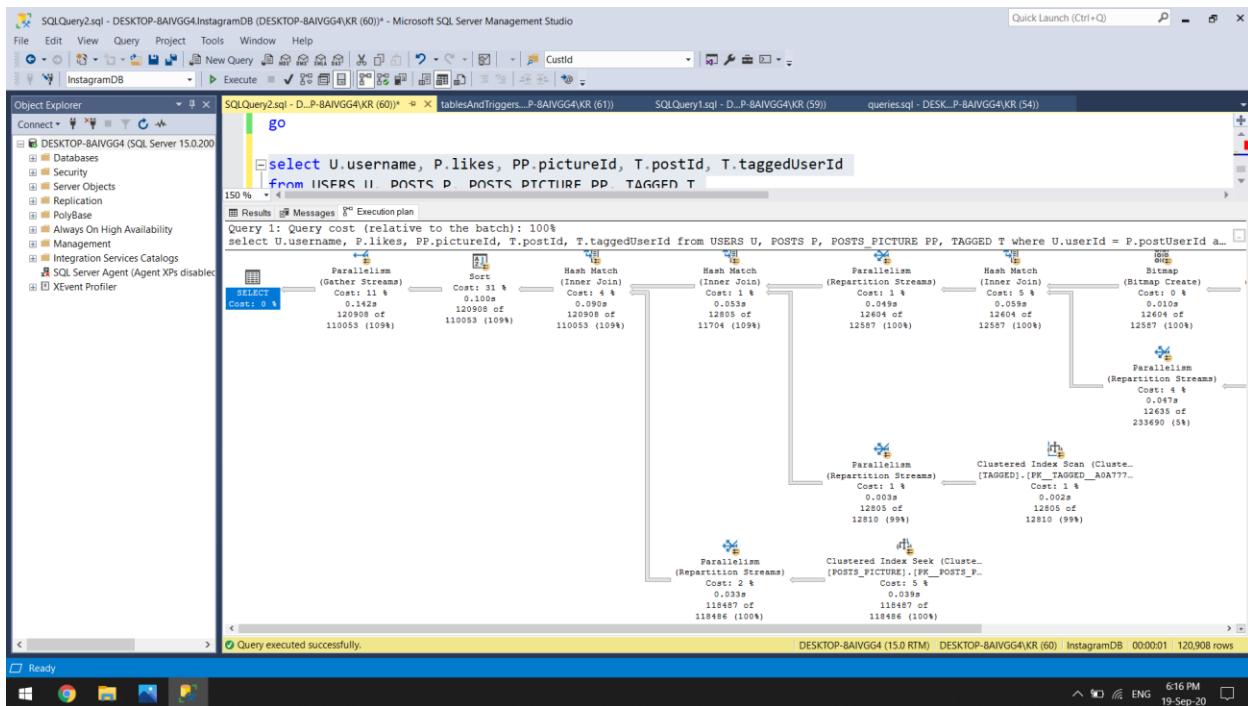
DESKTOP-8AIVGG4 (15.0 RTM) DESKTOP-8AIVGG4(KR (60)) InstagramDB 00:00:01 120,908 rows

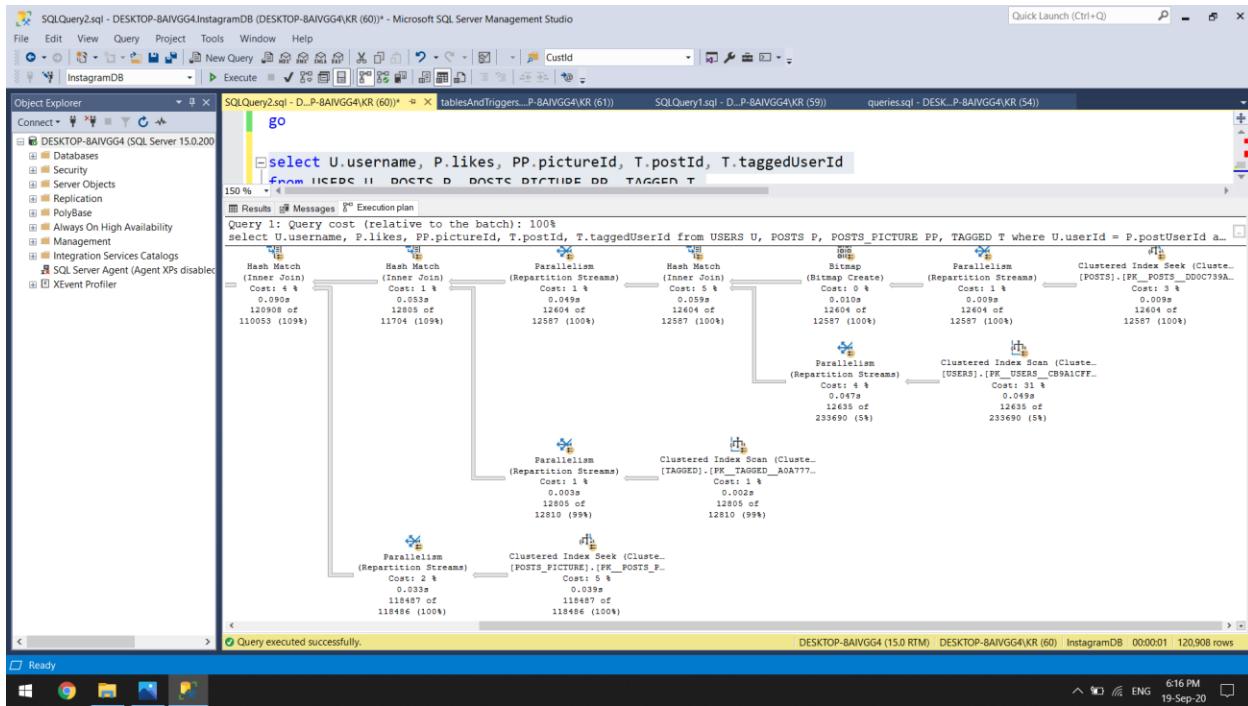
Ready

Quick Launch (Ctrl+Q)

616 PM ENG 19-Sep-20

The execution plan is shown below. Based on the current condition on postId of POSTS_PICTURE, the first table chosen is POSTS, followed by USERS, then TAGGED and finally POSTS_PICTURE.





Version 3: select U.username, P.likes, PP.pictureId, T.postId, T.taggedUserId
from USERS U, POSTS P, POSTS_PICTURE PP, TAGGED T
where

U.userId = P.postUserId and
P.caption='I am a caption' and
P.postId = PP.postId and
P.postId = T.postId

order by 2 desc

The query here has a condition on caption from POSTS table.

```

SQLQuery2.sql - DESKTOP-BAIVGG4.InstagramDB (DESKTOP-BAIVGG4\KR (60)) - Microsoft SQL Server Management Studio
File Edit View Query Project Tools Window Help
New Query Execute
CustId
SQLQuery2.sql - D...P-BAIVGG4\KR (60)* tablesAndTriggers_P-BAIVGG4\KR (61) SQLQuery1.sql - D...P-BAIVGG4\KR (59) queries.sql - DESK...P-BAIVGG4\KR (54)
Object Explorer
Connect DESKTOP-BAIVGG4 (SQL Server 15.0.200)
Databases Security Server Objects Replication PolyBase Always On High Availability Management Integration Services Catalogs SQL Server Agent (Agent XPs disabled) XEvent Profiler
SQLQuery2.sql - D...P-BAIVGG4\KR (60)* tablesAndTriggers_P-BAIVGG4\KR (61)
P.postId = T.postId
order by 2 desc
go

select U.username, P.likes, PP.pictureId, T.postId, T.taggedUserId
from USERS U, POSTS P, POSTS_PICTURE PP, TAGGED T
where
    U.userId = P.postUserId and
    P.caption='I am a caption' and
    P.postId = PP.postId and
    P.postId = T.postId
order by 2 desc
go

```

Results Messages Execution plan

	username	likes	pictureId	postId	taggedUserId
1	49842ABC	100	83483	55	764258
2	49842ABC	100	5437	55	764258
3	49842ABC	100	6750	55	764258
4	49842ABC	100	72288	55	764258
5	49842ABC	100	79956	55	764258
6	19167ABC	0	7923	38319	93
7	19167ABC	0	96473	38319	93
8	19167ABC	0	93370	38319	93
9	19167ABC	0	3771	38319	93
10	19167ABC	0	11921	38319	93
11	19167ABC	0	18320	38319	93
12	19167ABC	0	28356	38319	93
13	19167ABC	0	30169	38319	93
14	19167ABC	0	71347	38319	93
15	19203ABC	0	14734	29150	287

Query executed successfully.

The message output-

```

SQLQuery2.sql - DESKTOP-BAIVGG4.InstagramDB (DESKTOP-BAIVGG4\KR (60)) - Microsoft SQL Server Management Studio
File Edit View Query Project Tools Window Help
New Query Execute
CustId
SQLQuery2.sql - D...P-BAIVGG4\KR (60)* tablesAndTriggers_P-BAIVGG4\KR (61) SQLQuery1.sql - D...P-BAIVGG4\KR (59) queries.sql - DESK...P-BAIVGG4\KR (54)
Object Explorer
Connect DESKTOP-BAIVGG4 (SQL Server 15.0.200)
Databases Security Server Objects Replication PolyBase Always On High Availability Management Integration Services Catalogs SQL Server Agent (Agent XPs disabled) XEvent Profiler
SQLQuery2.sql - D...P-BAIVGG4\KR (60)* tablesAndTriggers_P-BAIVGG4\KR (61)
select U.username, P.likes, PP.pictureId, T.postId, T.taggedUserId
from USERS U, POSTS P, POSTS_PICTURE PP, TAGGED T
where
    U.userId = P.postUserId and
    P.caption='I am a caption' and
    P.postId = PP.postId and
    P.postId = T.postId
order by 2 desc
go

```

Results Messages Execution plan

SQL Server parse and compile time:
CPU time = 0 ms, elapsed time = 0 ms.

SQL Server Execution Times:
CPU time = 0 ms, elapsed time = 0 ms.
SQL Server parse and compile time:
CPU time = 0 ms, elapsed time = 17 ms.

(7427 rows affected)

Table 'Worktable'. Scan count 0, logical reads 0, physical reads 0, page server reads 0, read-ahead reads 33, page server read-ahead reads 0, lob log.
Table 'Workfile'. Scan count 0, logical reads 0, physical reads 0, page server reads 0, read-ahead reads 0, page server read-ahead reads 0, lob log.
Table 'TAGGED'. Scan count 1, logical reads 53, physical reads 1, page server reads 0, read-ahead reads 44, page server read-ahead reads 0, lob log.
Table 'POSTS_PICTURE'. Scan count 786, logical reads 2385, physical reads 63, page server reads 0, read-ahead reads 0, page server read-ahead reads 0
Table 'USERS'. Scan count 0, logical reads 2358, physical reads 73, page server reads 0, read-ahead reads 0, page server read-ahead reads 0, lob log.
Table 'POSTS'. Scan count 1, logical reads 371, physical reads 1, page server reads 0, read-ahead reads 363, page server read-ahead reads 0, lob log.

1 row affected

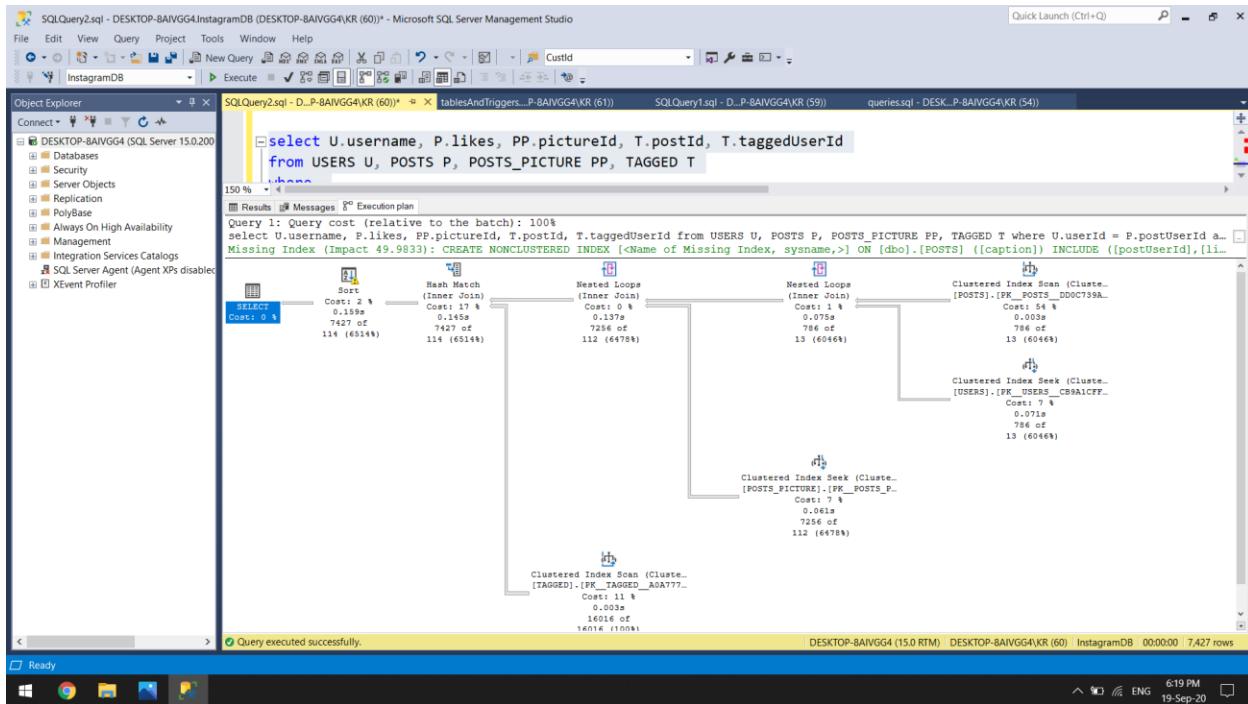
SQL Server Execution Times:
CPU time = 16 ms, elapsed time = 315 ms.
SQL Server parse and compile time:
CPU time = 0 ms, elapsed time = 0 ms.

SQL Server Execution Times:
CPU time = 0 ms, elapsed time = 0 ms.

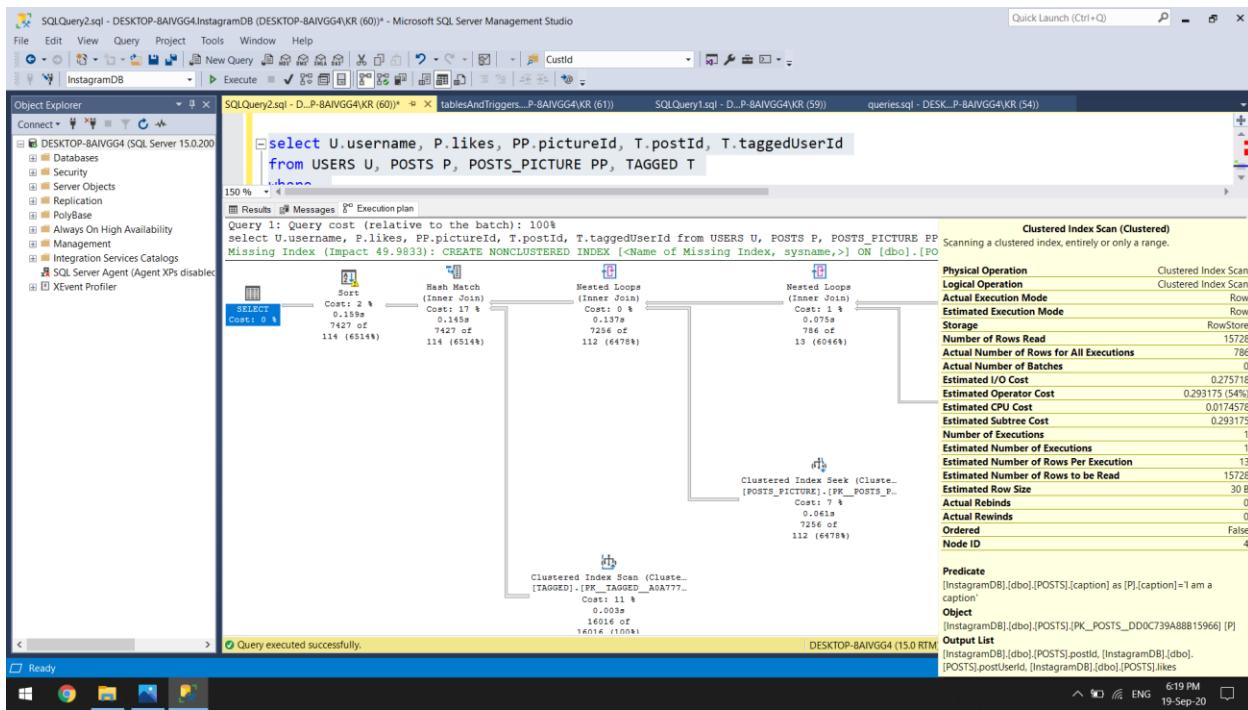
Completion time: 2020-09-19T18:18:43.2486646+05:30

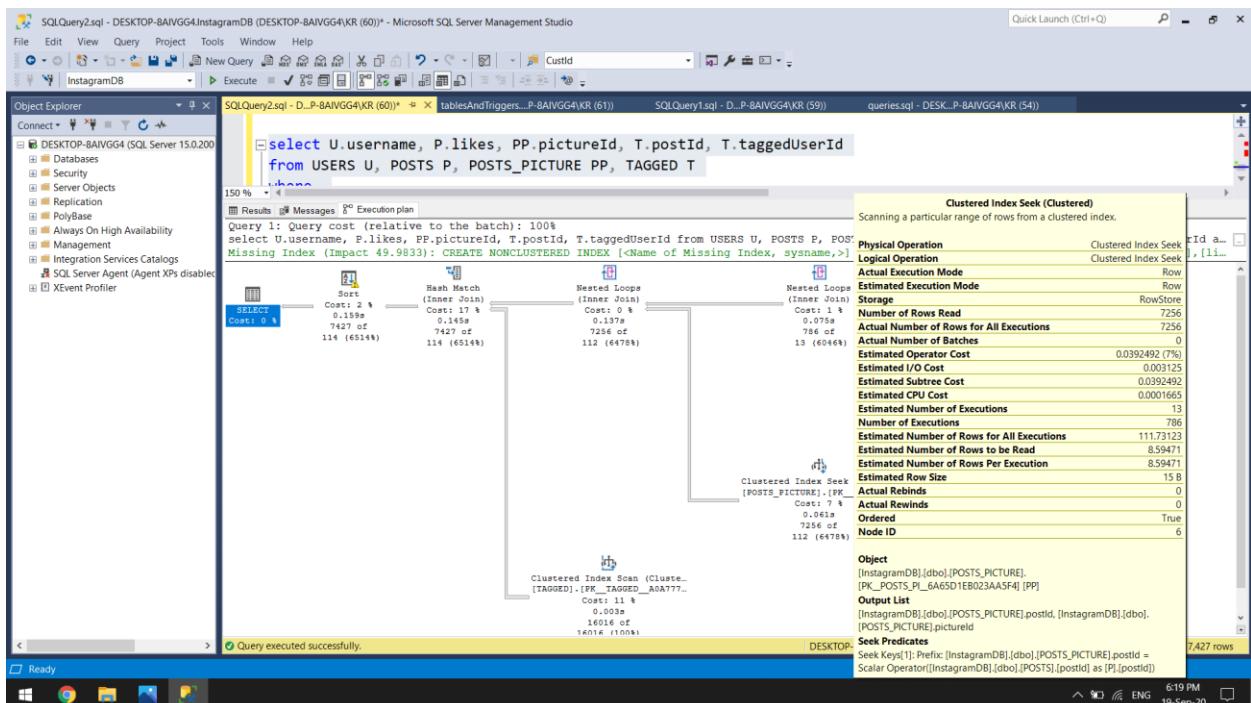
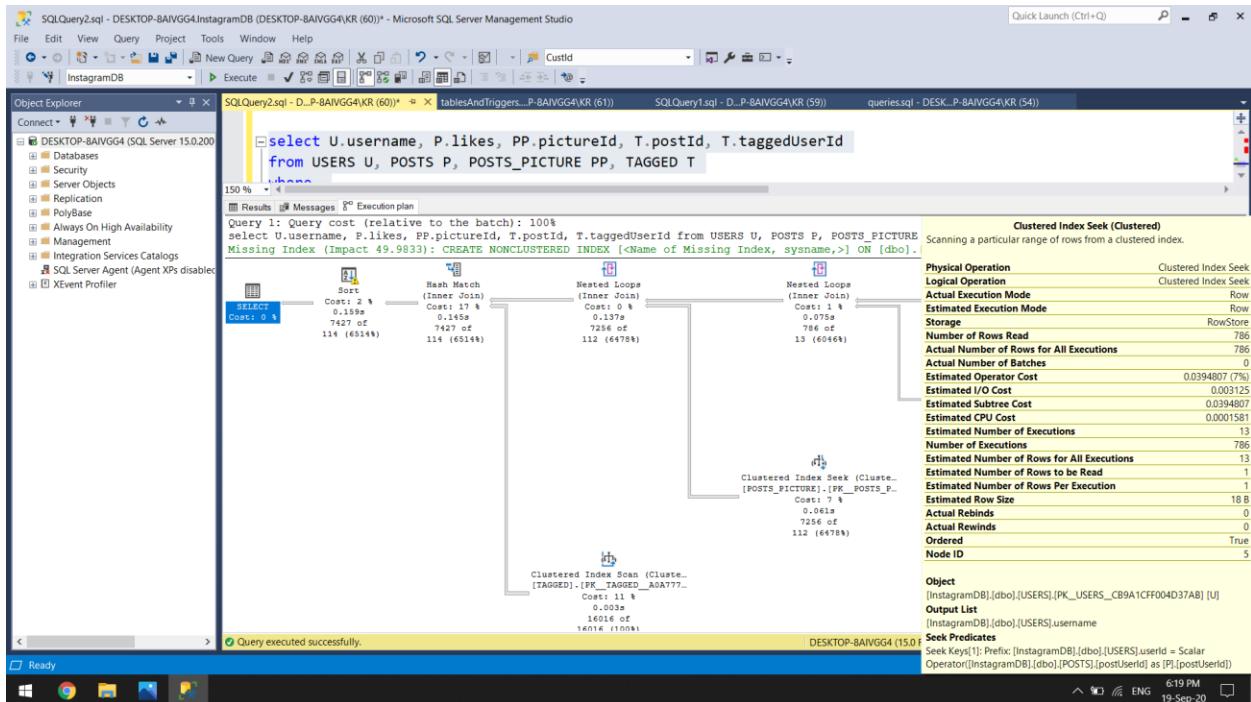
Query executed successfully.

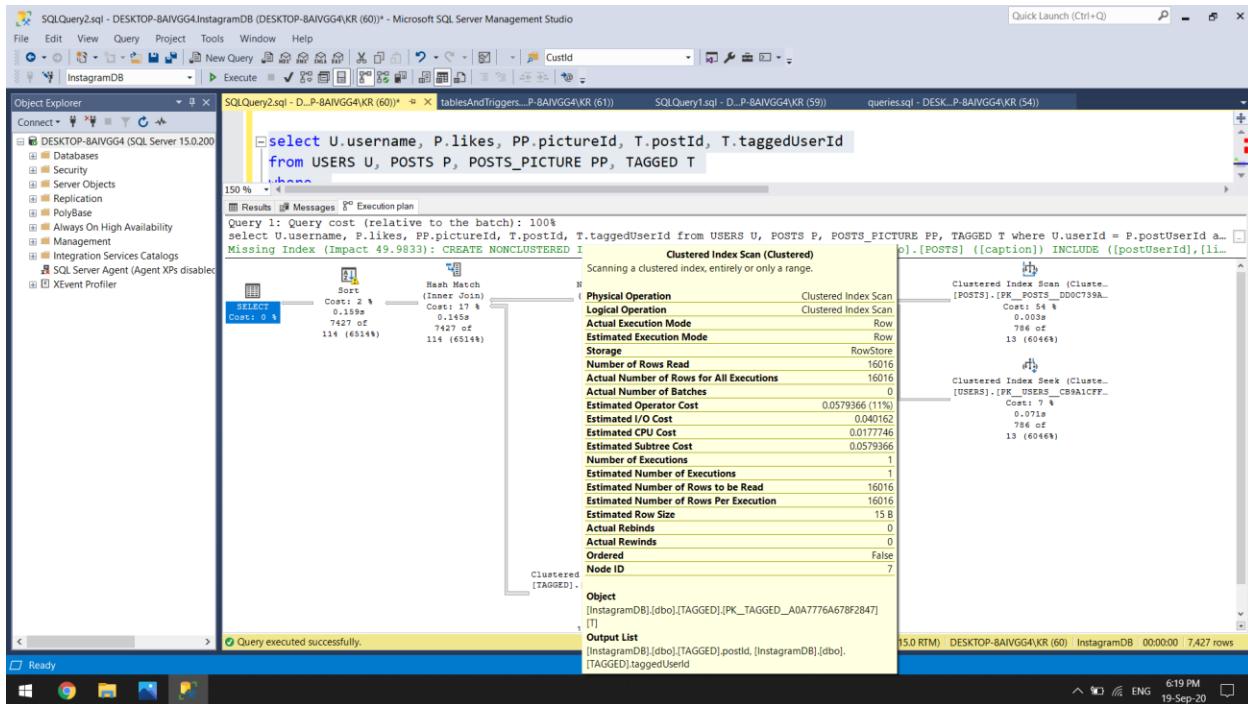
The execution plan shows that the tables are chosen in the order POSTS, USERS, POSTS_PICTURE, TAGGED. Notice the suggestion to create a non-clustered index on caption from POSTS table.



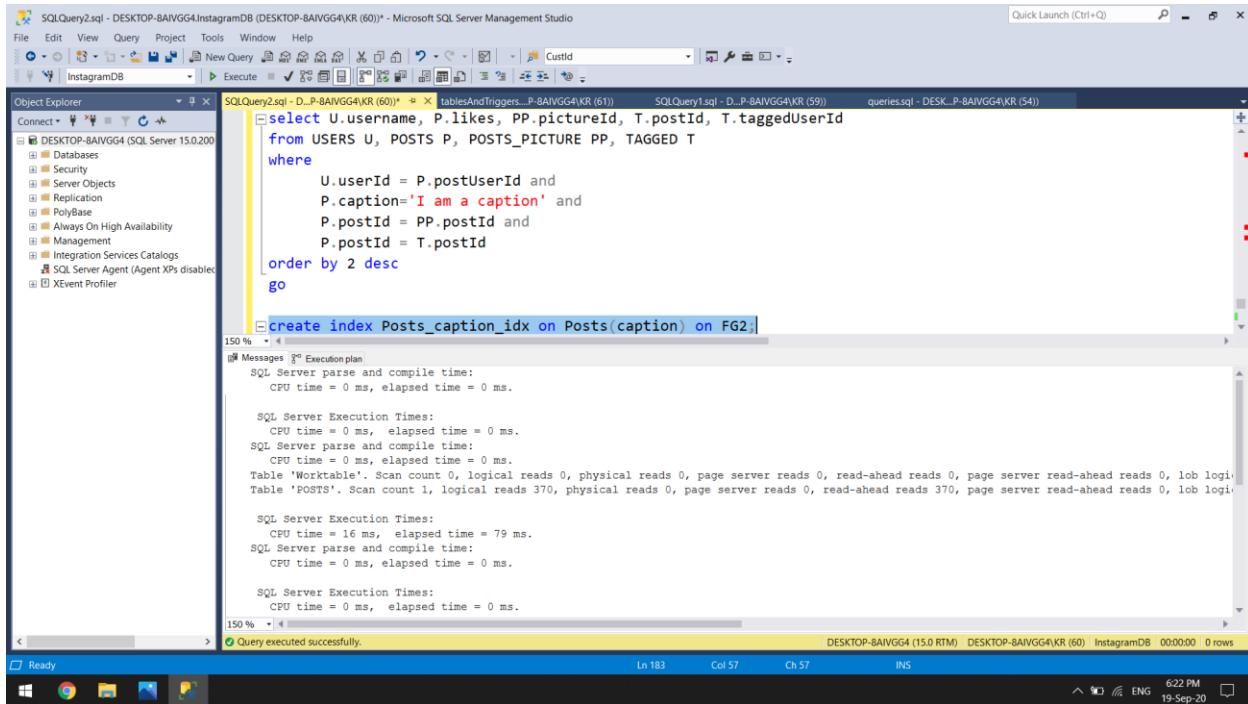
The detailed execution plan-







Now we create a non-clustered index on P.caption and check the execution plan.



Notice the changes in the execution time- elapsed time was previously 315 ms, it is now 185 ms.

SQLQuery2.sql - DESKTOP-8AIVGG4.InstagramDB (DESKTOP-8AIVGG4\KR (60)) - Microsoft SQL Server Management Studio

File Edit View Query Project Tools Window Help

Object Explorer SQLQuery2.sql - D...P-8AIVGG4\KR (60)* tablesAndTriggers...P-8AIVGG4\KR (61) SQLQuery1.sql - D...P-8AIVGG4\KR (59) queries.sql - DESKTOP-8AIVGG4\KR (54)

Connect Connect to Server Refresh Connect to Database

InstgramDB

Execute New Query New Results Grid New Execution Plan New Diagram New File New Table

Object Explorer

Connect

DESKTOP-8AIVGG4 (SQL Server 15.0.200)

- Databases
- Security
- Server Objects
- Replication
- PolyBase
- Always On High Availability
- Management
- Integration Services Catalogs
- SQL Server Agent (Agent XPs disabled)
- XEvent Profiler

SQL Server parse and compile time:
CPU time = 0 ms, elapsed time = 0 ms.

SQL Server Execution Times:
CPU time = 0 ms, elapsed time = 0 ms.

SQL Server parse and compile time:
CPU time = 0 ms, elapsed time = 19 ms.

(7427 rows affected)

Table 'POSTS_PICTURE'. Scan count 804, logical reads 3663, physical reads 2, page server reads 0, read-ahead reads 485, page server read-ahead reads 1, lob logic reads 0, lob logical reads 0, lob updates 0.
Table 'USERS'. Scan count 0, logical reads 3151, physical reads 8, page server reads 0, read-ahead reads 448, page server read-ahead reads 0, lob log reads 0, lob logical reads 0, lob updates 0.
Table 'Workfile'. Scan count 0, logical reads 0, physical reads 0, page server reads 0, read-ahead reads 0, page server read-ahead reads 0, lob logic reads 0, lob logical reads 0, lob updates 0.
Table 'TAGGED'. Scan count 1, logical reads 53, physical reads 1, page server reads 0, read-ahead reads 44, page server read-ahead reads 0, lob logic reads 0, lob logical reads 0, lob updates 0.
Table 'POSTS'. Scan count 1, logical reads 371, physical reads 1, page server reads 0, read-ahead reads 363, page server read-ahead reads 0, lob logic reads 0, lob logical reads 0, lob updates 0.

(1 row affected)

SQL Server Execution Times:
CPU time = 16 ms, elapsed time = 185 ms.

SQL Server parse and compile time:
CPU time = 0 ms, elapsed time = 0 ms.

SQL Server Execution Times:
CPU time = 0 ms, elapsed time = 0 ms.

Completion time: 2020-09-19T18:20:58.4439083+05:30

150 %

Results Messages Execution plan

Query executed successfully.

In 28 Col 1 Ch 1 INS

DESKTOP-8AIVGG4 (15.0 RTM) | DESKTOP-8AIVGG4\KR (60) | InstgramDB | 00:00:00 | 7,427 rows

Ready

621 PM ENG 19-Sep-20

Previously, the cost of clustered index scan on POSTS was 54%. It has now reduced to 0%.

