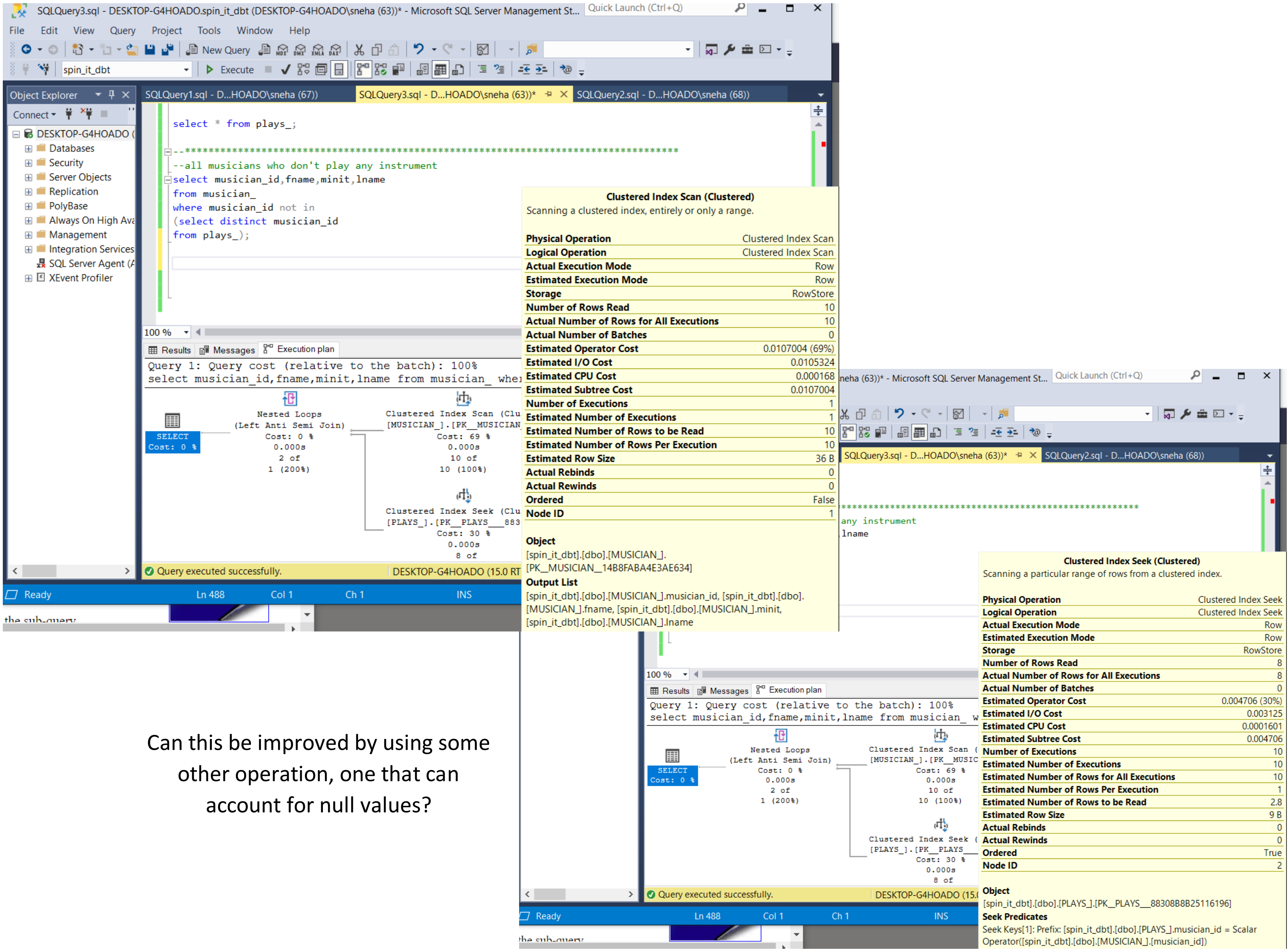
DBT ASSIGNMENT 2

Sneha Hegde

PES1201801157

- Rewriting NOT IN-

When a query is written using NOT IN, we get



```
select * from plays_;

--***********************************************************************************
--all musicians who don't play any instrument
select musician_id,fname,minit,lname
from musician_
where musician_id not in
(select distinct musician_id
from plays_);
```

```
Query 1: Query cost (relative to the batch): 100%
select musician_id,fname,minit,lname from musician_ wher
```

```
SELECT        Nested Loops              Clustered Index Scan (Clu
Cost: 0 %     (Left Anti Semi Join)     [MUSICIAN_].[PK__MUSICIAN
              Cost: 0 %                 Cost: 69 %
              0.000s                    0.000s
              2 of                      10 of
              1 (200%)                  10 (100%)

                                        Clustered Index Seek (Clu
                                        [PLAYS_].[PK__PLAYS___883
                                        Cost: 30 %
                                        0.000s
                                        8 of
```

Query executed successfully. DESKTOP-G4HOADO (15.0 RT

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	10
Actual Number of Rows for All Executions	10
Actual Number of Batches	0
Estimated Operator Cost	0.0107004 (69%)
Estimated I/O Cost	0.0105324
Estimated CPU Cost	0.000168
Estimated Subtree Cost	0.0107004
Number of Executions	1
Estimated Number of Executions	1
Estimated Number of Rows to be Read	10
Estimated Number of Rows Per Execution	10
Estimated Row Size	36 B
Actual Rebinds	0
Actual Rewinds	0
Ordered	False
Node ID	1

Object
[spin_it_dbt].[dbo].[MUSICIAN_].[PK__MUSICIAN__14B8FABA4E3AE634]
Output List
[spin_it_dbt].[dbo].[MUSICIAN_].musician_id, [spin_it_dbt].[dbo].[MUSICIAN_].fname, [spin_it_dbt].[dbo].[MUSICIAN_].minit, [spin_it_dbt].[dbo].[MUSICIAN_].lname

```
Query 1: Query cost (relative to the batch): 100%
select musician_id,fname,minit,lname from musician_ w

SELECT        Nested Loops              Clustered Index Scan (
Cost: 0 %     (Left Anti Semi Join)     [MUSICIAN_].[PK__MUSIC
              Cost: 0 %                 Cost: 69 %
              0.000s                    0.000s
              2 of                      10 of
              1 (200%)                  10 (100%)

                                        Clustered Index Seek (
                                        [PLAYS_].[PK__PLAYS___
                                        Cost: 30 %
                                        0.000s
                                        8 of
```

Query executed successfully. DESKTOP-G4HOADO (15.0

Clustered Index Seek (Clustered)

Scanning a particular range of rows from a clustered index.

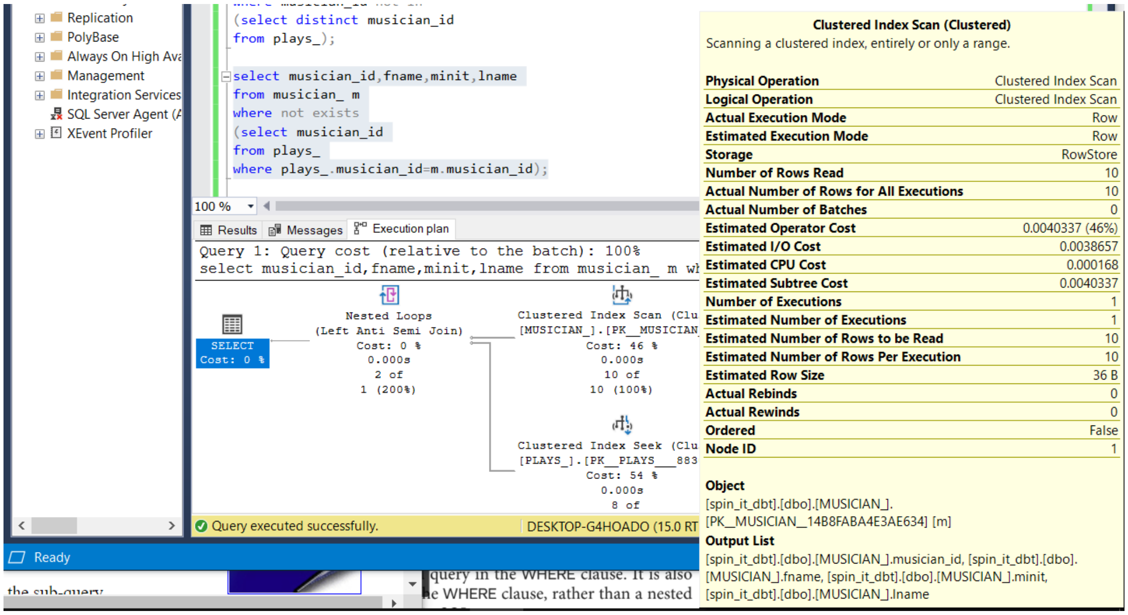
Physical Operation	Clustered Index Seek
Logical Operation	Clustered Index Seek
Actual Execution Mode	Row
Estimated Execution Mode	Row
Storage	RowStore
Number of Rows Read	8
Actual Number of Rows for All Executions	8
Actual Number of Batches	0
Estimated Operator Cost	0.004706 (30%)
Estimated I/O Cost	0.003125
Estimated CPU Cost	0.0001601
Estimated Subtree Cost	0.004706
Number of Executions	10
Estimated Number of Executions	10
Estimated Number of Rows for All Executions	10
Estimated Number of Rows Per Execution	1
Estimated Number of Rows to be Read	2.8
Estimated Row Size	9 B
Actual Rebinds	0
Actual Rewinds	0
Ordered	True
Node ID	2

Object
[spin_it_dbt].[dbo].[PLAYS_].[PK__PLAYS___88308B8B25116196]
Seek Predicates
Seek Keys[1]: Prefix: [spin_it_dbt].[dbo].[PLAYS_].musician_id = Scalar Operator([spin_it_dbt].[dbo].[MUSICIAN_].[musician_id])

Can this be improved by using some other operation, one that can account for null values?

We can see that the plan for NOT EXISTS and LEFT OUTER JOIN is the same, but is considerably better than that of NOT IN

NOT EXISTS-



```
(select distinct musician_id
from plays_);

select musician_id,fname,minit,lname
from musician_ m
where not exists
(select musician_id
from plays_
where plays_.musician_id=m.musician_id);
```

```
Query 1: Query cost (relative to the batch): 100%
select musician_id,fname,minit,lname from musician_ m wh
```

```
SELECT        Nested Loops              Clustered Index Scan (Clu
Cost: 0 %     (Left Anti Semi Join)     [MUSICIAN_].[PK__MUSICIAN
              Cost: 0 %                 Cost: 46 %
              0.000s                    0.000s
              2 of                      10 of
              1 (200%)                  10 (100%)

                                        Clustered Index Seek (Clu
                                        [PLAYS_].[PK__PLAYS__883
                                        Cost: 54 %
                                        0.000s
                                        8 of
```

Query executed successfully. DESKTOP-G4HOADO (15.0 RT

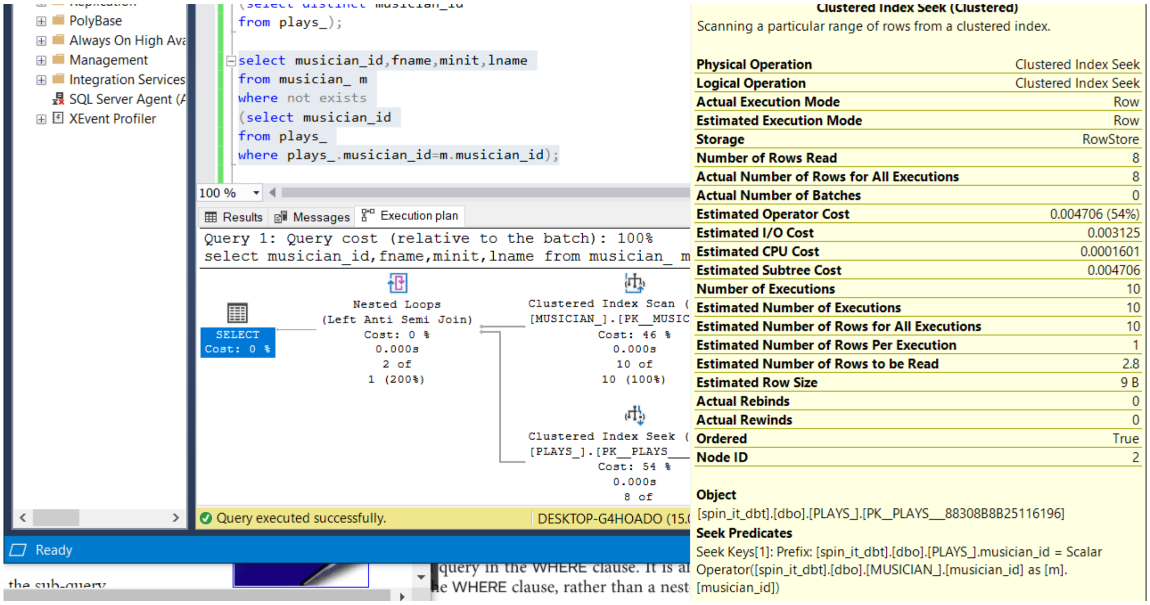
Clustered Index Scan (Clustered)

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	10
Actual Number of Rows for All Executions	10
Actual Number of Batches	0
Estimated Operator Cost	0.0040337 (46%)
Estimated I/O Cost	0.0038657
Estimated CPU Cost	0.000168
Estimated Subtree Cost	0.0040337
Number of Executions	1
Estimated Number of Executions	1
Estimated Number of Rows to be Read	10
Estimated Number of Rows Per Execution	10
Estimated Row Size	36 B
Actual Rebinds	0
Actual Rewinds	0
Ordered	False
Node ID	1

Object
[spin_it_dbt].[dbo].[MUSICIAN_].
[PK__MUSICIAN__14B8FABA4E3AE634] [m]

Output List
[spin_it_dbt].[dbo].[MUSICIAN_].musician_id, [spin_it_dbt].[dbo].
[MUSICIAN_].fname, [spin_it_dbt].[dbo].[MUSICIAN_].minit,
[spin_it_dbt].[dbo].[MUSICIAN_].lname



```
from plays_);

select musician_id,fname,minit,lname
from musician_ m
where not exists
(select musician_id
from plays_
where plays_.musician_id=m.musician_id);
```

```
Query 1: Query cost (relative to the batch): 100%
select musician_id,fname,minit,lname from musician_ m
```

```
SELECT        Nested Loops              Clustered Index Scan (
Cost: 0 %     (Left Anti Semi Join)     [MUSICIAN_].[PK__MUSIC
              Cost: 0 %                 Cost: 46 %
              0.000s                    0.000s
              2 of                      10 of
              1 (200%)                  10 (100%)

                                        Clustered Index Seek (
                                        [PLAYS_].[PK__PLAYS__
                                        Cost: 54 %
                                        0.000s
                                        8 of
```

Query executed successfully. DESKTOP-G4HOADO (15.0

Clustered Index Seek (Clustered)

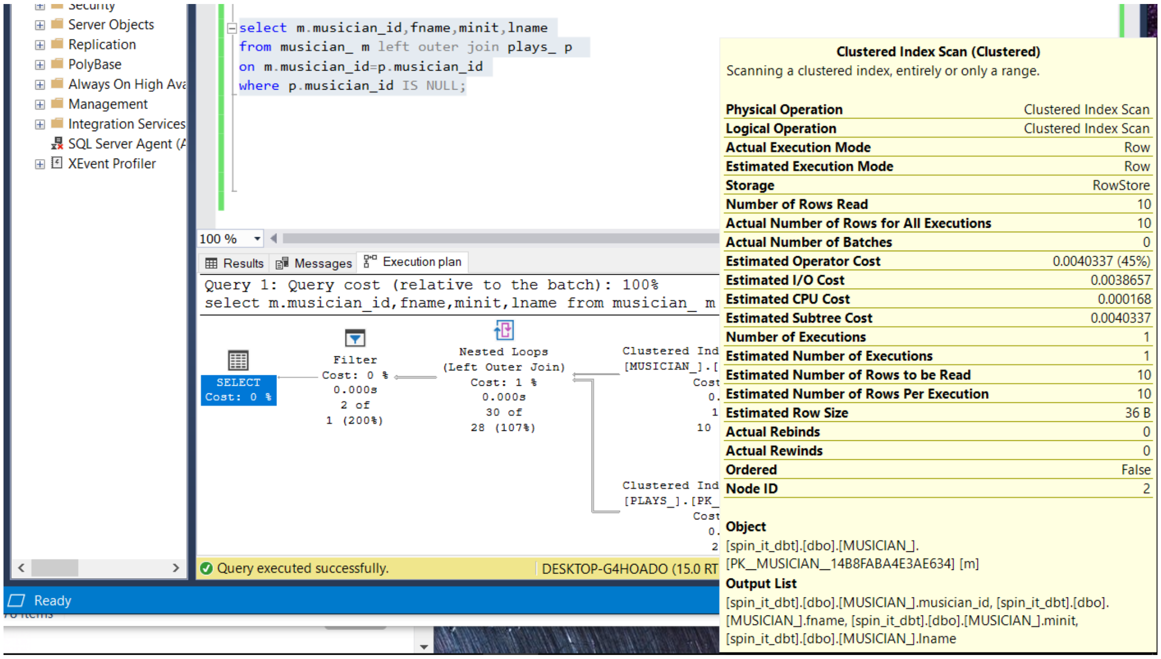
Scanning a particular range of rows from a clustered index.

Physical Operation	Clustered Index Seek
Logical Operation	Clustered Index Seek
Actual Execution Mode	Row
Estimated Execution Mode	Row
Storage	RowStore
Number of Rows Read	8
Actual Number of Rows for All Executions	8
Actual Number of Batches	0
Estimated Operator Cost	0.004706 (54%)
Estimated I/O Cost	0.003125
Estimated CPU Cost	0.0001601
Estimated Subtree Cost	0.004706
Number of Executions	10
Estimated Number of Executions	10
Estimated Number of Rows for All Executions	10
Estimated Number of Rows Per Execution	1
Estimated Number of Rows to be Read	2.8
Estimated Row Size	9 B
Actual Rebinds	0
Actual Rewinds	0
Ordered	True
Node ID	2

Object
[spin_it_dbt].[dbo].[PLAYS_].[PK__PLAYS__88308B8B25116196]

Seek Predicates
Seek Keys[1]: Prefix: [spin_it_dbt].[dbo].[PLAYS_].musician_id = Scalar
Operator([spin_it_dbt].[dbo].[MUSICIAN_].[musician_id] as [m].
[musician_id])

LEFT OUTER JOIN-



```
select m.musician_id,fname,minit,lname
from musician_ m left outer join plays_ p
on m.musician_id=p.musician_id
where p.musician_id IS NULL;
```

```
Query 1: Query cost (relative to the batch): 100%
select m.musician_id,fname,minit,lname from musician_ m
```

Query executed successfully. DESKTOP-G4HOADO (15.0 RT

Clustered Index Scan (Clustered)

Scanning a clustered index, entirely or only a range.

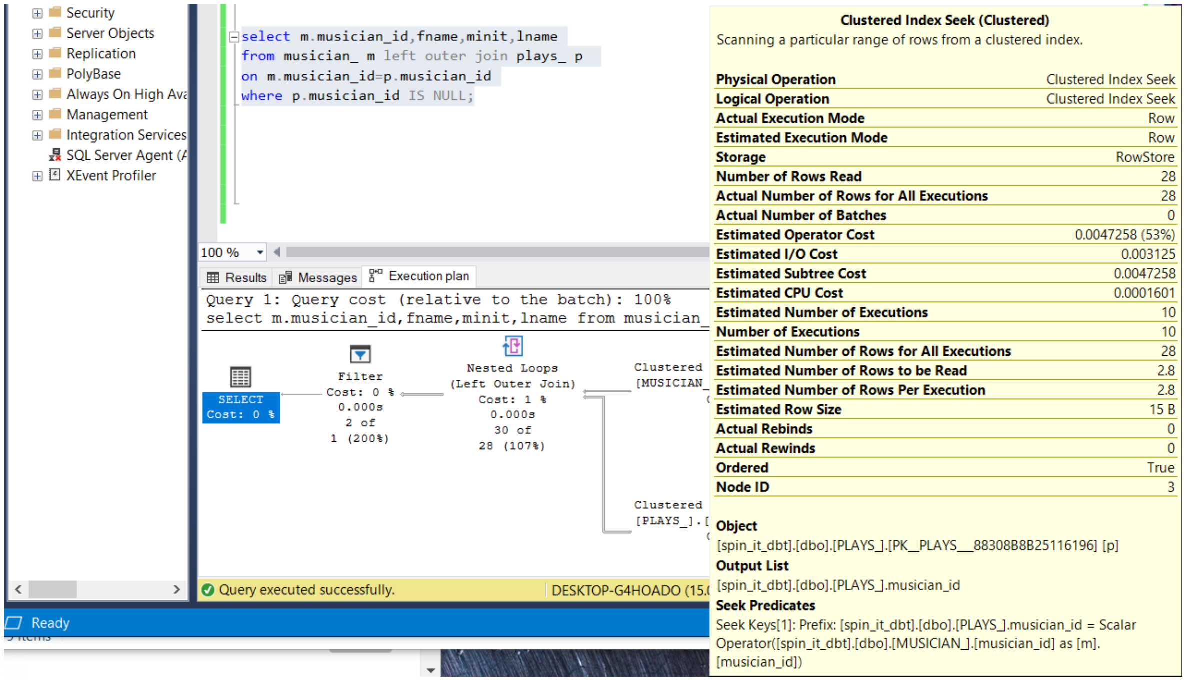
Physical Operation	Clustered Index Scan
Logical Operation	Clustered Index Scan
Actual Execution Mode	Row
Estimated Execution Mode	Row
Storage	RowStore
Number of Rows Read	10
Actual Number of Rows for All Executions	10
Actual Number of Batches	0
Estimated Operator Cost	0.0040337 (45%)
Estimated I/O Cost	0.0038657
Estimated CPU Cost	0.000168
Estimated Subtree Cost	0.0040337
Number of Executions	1
Estimated Number of Executions	1
Estimated Number of Rows to be Read	10
Estimated Number of Rows Per Execution	10
Estimated Row Size	36 B
Actual Rebinds	0
Actual Rewinds	0
Ordered	False
Node ID	2

Object

[spin_it_dbt].[dbo].[MUSICIAN_].
[PK__MUSICIAN__14B8FABA4E3AE634] [m]

Output List

[spin_it_dbt].[dbo].[MUSICIAN_].musician_id, [spin_it_dbt].[dbo].
[MUSICIAN_].fname, [spin_it_dbt].[dbo].[MUSICIAN_].minit,
[spin_it_dbt].[dbo].[MUSICIAN_].lname



```
select m.musician_id,fname,minit,lname
from musician_ m left outer join plays_ p
on m.musician_id=p.musician_id
where p.musician_id IS NULL;
```

```
Query 1: Query cost (relative to the batch): 100%
select m.musician_id,fname,minit,lname from musician_
```

Query executed successfully. DESKTOP-G4HOADO (15.0

Clustered Index Seek (Clustered)

Scanning a particular range of rows from a clustered index.

Physical Operation	Clustered Index Seek
Logical Operation	Clustered Index Seek
Actual Execution Mode	Row
Estimated Execution Mode	Row
Storage	RowStore
Number of Rows Read	28
Actual Number of Rows for All Executions	28
Actual Number of Batches	0
Estimated Operator Cost	0.0047258 (53%)
Estimated I/O Cost	0.003125
Estimated Subtree Cost	0.0047258
Estimated CPU Cost	0.0001601
Estimated Number of Executions	10
Number of Executions	10
Estimated Number of Rows for All Executions	28
Estimated Number of Rows to be Read	2.8
Estimated Number of Rows Per Execution	2.8
Estimated Row Size	15 B
Actual Rebinds	0
Actual Rewinds	0
Ordered	True
Node ID	3

Object

[spin_it_dbt].[dbo].[PLAYS_].[PK__PLAYS__88308B8B25116196] [p]

Output List

[spin_it_dbt].[dbo].[PLAYS_].musician_id

Seek Predicates

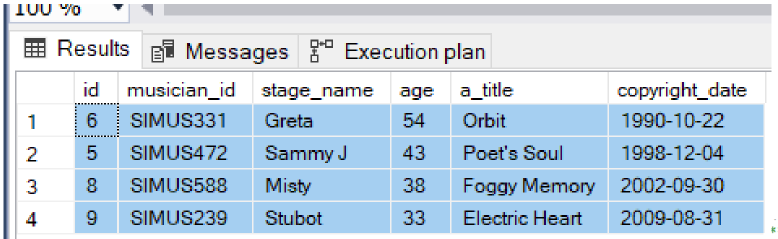
Seek Keys[1]: Prefix: [spin_it_dbt].[dbo].[PLAYS_].musician_id = Scalar
Operator([spin_it_dbt].[dbo].[MUSICIAN_].[musician_id] as [m].
[musician_id])

- Demonstrating select condition being pushed-
  (I've added an extra integer column, id, to the tables)

  If we start with a union:

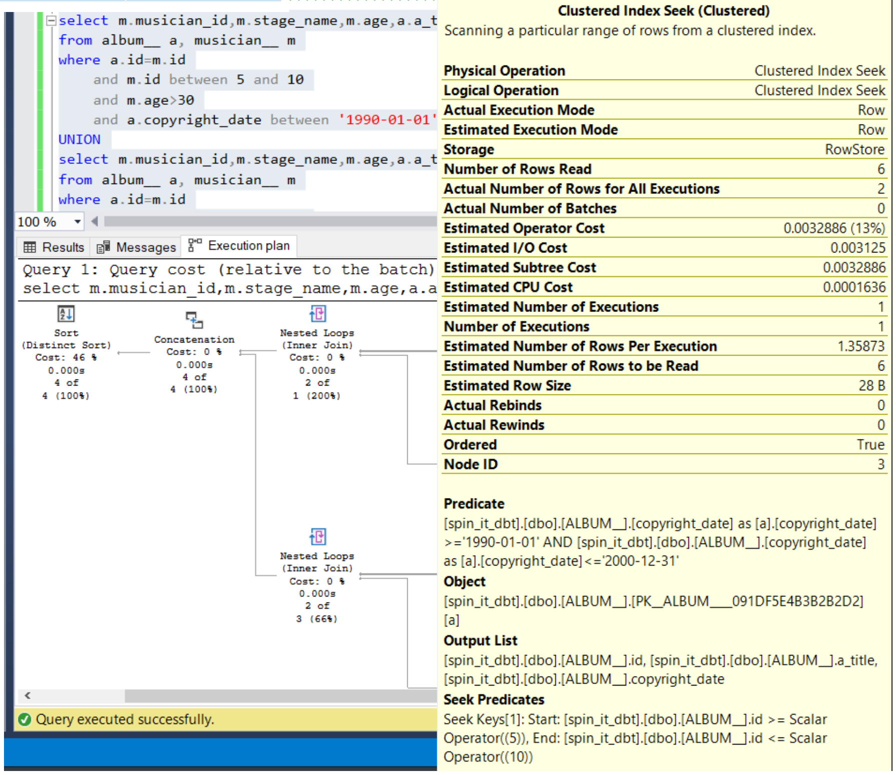
```
select m.musician_id,m.stage_name,m.age,a.a_title,a.copyright_date
from album__ a, musician__ m
where a.id=m.id
       and m.id between 5 and 10
       and m.age>30
       and a.copyright_date between '1990-01-01' and '2000-12-31'
UNION
select m.musician_id,m.stage_name,m.age,a.a_title,a.copyright_date
from album__ a, musician__ m
where a.id=m.id
       and m.id between 5 and 10
       and m.age>30
       and a.copyright_date between '2001-01-01' and '2010-12-31'
order by a.copyright_date;
```

This is the result set generated:



	id	musician_id	stage_name	age	a_title	copyright_date
1	6	SIMUS331	Greta	54	Orbit	1990-10-22
2	5	SIMUS472	Sammy J	43	Poet's Soul	1998-12-04
3	8	SIMUS588	Misty	38	Foggy Memory	2002-09-30
4	9	SIMUS239	Stubot	33	Electric Heart	2009-08-31

The execution plan shows that though we specified m.id in the where clause, we can see that the optimizer chooses to push album__.id between 5 and 10 in the seek predicate for album__, to make the join more efficient. This happens in both parts of the union.



```
select m.musician_id,m.stage_name,m.age,a.a_t
from album__ a, musician__ m
where a.id=m.id
    and m.id between 5 and 10
    and m.age>30
    and a.copyright_date between '1990-01-01'
UNION
select m.musician_id,m.stage_name,m.age,a.a_t
from album__ a, musician__ m
where a.id=m.id
```

```
Query 1: Query cost (relative to the batch)
select m.musician_id,m.stage_name,m.age,a.a
```

Clustered Index Seek (Clustered)

Scanning a particular range of rows from a clustered index.

Physical Operation	Clustered Index Seek
Logical Operation	Clustered Index Seek
Actual Execution Mode	Row
Estimated Execution Mode	Row
Storage	RowStore
Number of Rows Read	6
Actual Number of Rows for All Executions	2
Actual Number of Batches	0
Estimated Operator Cost	0.0032886 (13%)
Estimated I/O Cost	0.003125
Estimated Subtree Cost	0.0032886
Estimated CPU Cost	0.0001636
Estimated Number of Executions	1
Number of Executions	1
Estimated Number of Rows Per Execution	1.35873
Estimated Number of Rows to be Read	6
Estimated Row Size	28 B
Actual Rebinds	0
Actual Rewinds	0
Ordered	True
Node ID	3

Predicate

[spin_it_dbt].[dbo].[ALBUM__].[copyright_date] as [a].[copyright_date] >='1990-01-01' AND [spin_it_dbt].[dbo].[ALBUM__].[copyright_date] as [a].[copyright_date]<='2000-12-31'

Object

[spin_it_dbt].[dbo].[ALBUM__].[PK__ALBUM___091DF5E4B3B2B2D2] [a]

Output List

[spin_it_dbt].[dbo].[ALBUM__].id, [spin_it_dbt].[dbo].[ALBUM__].a_title, [spin_it_dbt].[dbo].[ALBUM__].copyright_date

Seek Predicates

Seek Keys[1]: Start: [spin_it_dbt].[dbo].[ALBUM__].id >= Scalar Operator((5)), End: [spin_it_dbt].[dbo].[ALBUM__].id <= Scalar Operator((10))

What if the union part is nested?

```
select new.id,new.musician_id,new.stage_name, new.age, new.a_title, new.copyright_date
from
(
select m.id,m.musician_id,m.stage_name,m.age,a.a_title,a.copyright_date
from album__ a, musician__ m
where a.id=m.id
       and m.id between 5 and 10
       and m.age>30
       and a.copyright_date between '1990-01-01' and '2000-12-31'
UNION
select m.id,m.musician_id,m.stage_name,m.age,a.a_title,a.copyright_date
from album__ a, musician__ m
where a.id=m.id
       and m.id between 5 and 10
       and m.age>30
       and a.copyright_date between '2001-01-01' and '2010-12-31'
) new
where
new.id between 5 and 10
and new.age>30
order by new.copyright_date;
```

In this case, we have put new.age>30 and new.id between 5 and 10 in the outer query.

It produces the same result set.

In the execution plan, the criteria new.age>30 is pushed to the musician__ table in the inner query, even though it's mentioned only in the outer query. Again, the new.id between 5 and 10 is pushed to both tables in the inner query. This is done on both parts of the union. The query optimizer pushes the criteria in the outer query into the inner query, thus getting rid of the nested query structure. It seems to evaluate it in the same way as the previous one, so this one has extra unnecessary parts. Therefore, the selection criteria is being pushed to the appropriate level.

- Altering join order of multi-joins to see how the optimizer handles it-

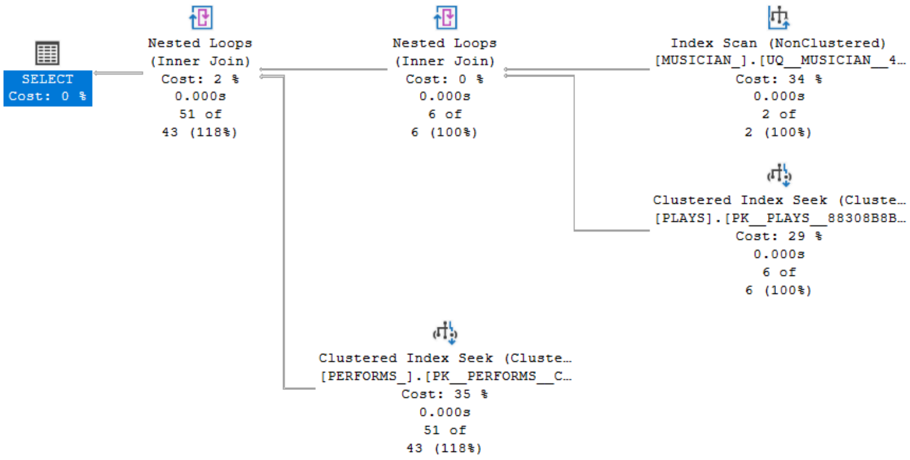
  Starting with a basic example, let's try joining performs_, plays_, and musician_ on musician_id. Though this join produces no meaningful information, it's just to take a look at how the optimizer internally chooses the join order.

  We do a join in three different ways, three different orders.

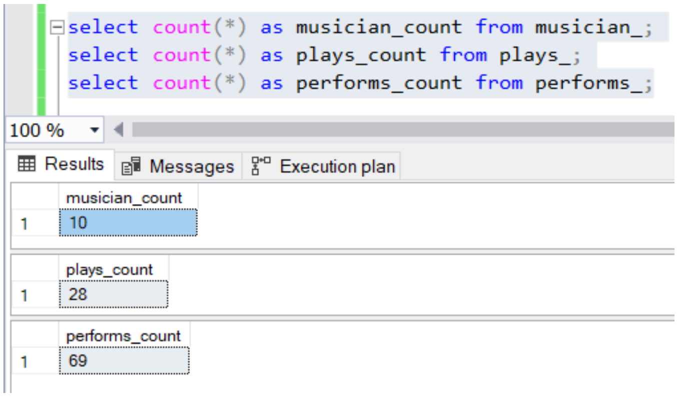
```
select m.musician_id,stage_name,song_id,instrument_id
from((musician_ m join performs_ pe on m.musician_id=pe.musician_id)
                     join plays pl on pl.musician_id=pe.musician_id)
where m.musician_id like '%010' or m.musician_id like '%177';

--******************************
```

```sql
select m.musician_id,stage_name,song_id,instrument_id
from((musician_ m join plays_ pl on m.musician_id=pl.musician_id)
                    join performs pe on pl.musician_id=pe.musician_id)
where m.musician_id like '%010' or m.musician_id like '%177';

--***********************************

select m.musician_id,stage_name,song_id,instrument_id
from((plays_ pl join performs_ pe on pl.musician_id=pe.musician_id)
                    join musician_ m on m.musician_id=pl.musician_id)
              where m.musician_id like '%010' or m.musician_id like '%177';
```

In all three cases, irrespective of how we specify the join order, it automatically generates the same order.



The number of rows in each of the tables can be seen in the picture, so it makes sense to join performs_ at the end, after joining musician_ and plays_, to make sure that the intermediate result set is the smallest.



```sql
select count(*) as musician_count from musician_;
select count(*) as plays_count from plays_;
select count(*) as performs_count from performs_;
```

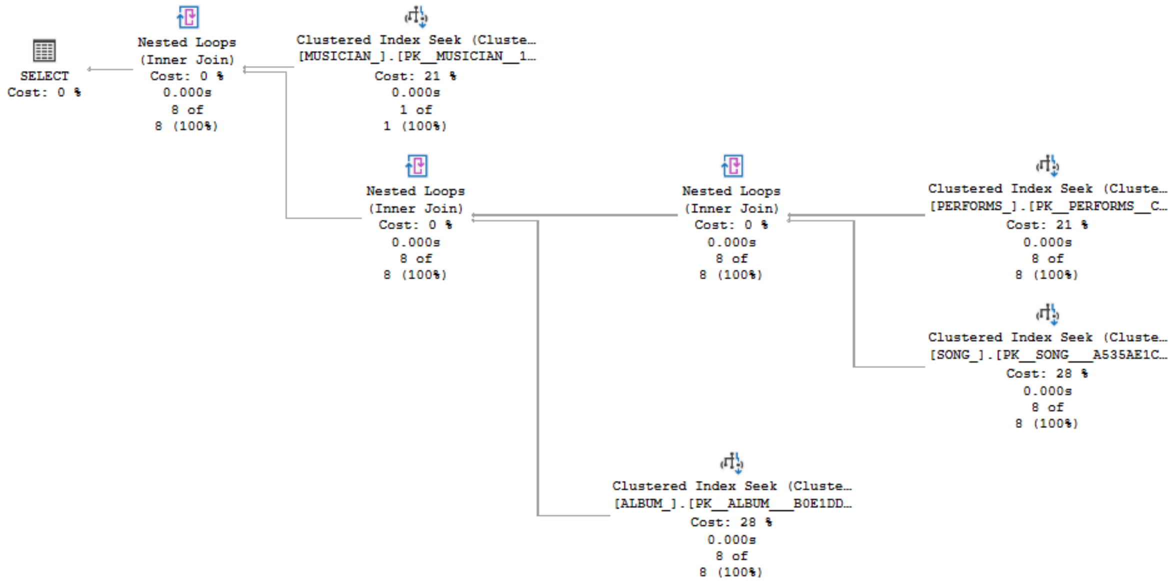
```
   musician_count
1  10

   plays_count
1  28

   performs_count
1  69
```

For another example, let’s involve musician_, song_, performs_ and album_

If we write

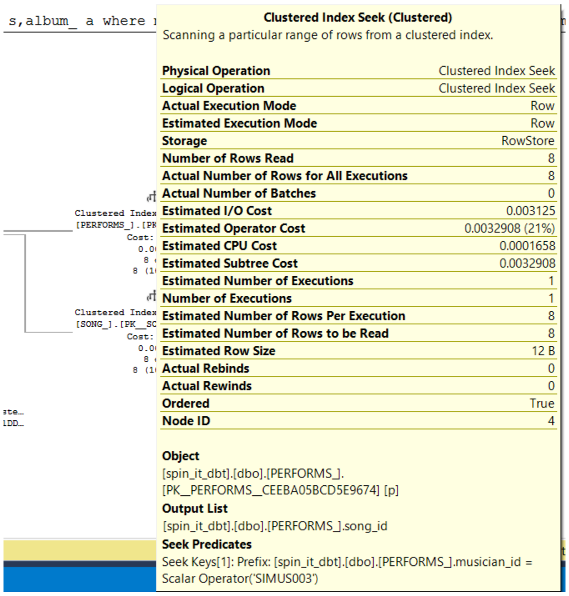
```
--display all the songs performed by a certain musician, along with the album titles of
the songs
select m.musician_id,m.stage_name,s.s_title,a.a_title
from musician_ m,performs_ p,song_ s,album_ a
where m.musician_id = 'SIMUS003'
       and p.musician_id = m.musician_id
       and p.song_id = s.song_id
                     and s.album_id = a.album_id;
```

specifying the search criteria on musician_id of m, we get



performs_ and song_ have been joined first, followed by album_, and finally, musician_.

We can see that the search criteria, that is, m.musician_id=’SIMUS003’ has been pushed to the base (performs_), though no criteria was specified on it. This helps limit the number of rows in the result set of that first join, making it more efficient. If this hadn’t happened, it would’ve matched all the songs with their respective musicians (instead of just the musician specified), producing far more rows than necessary, and the filter for the specific musician_id would’ve only been applied at the end. Therefore, we’re able to select only the rows that we need at the very beginning itself.



Clustered Index Seek (Clustered)

Scanning a particular range of rows from a clustered index.

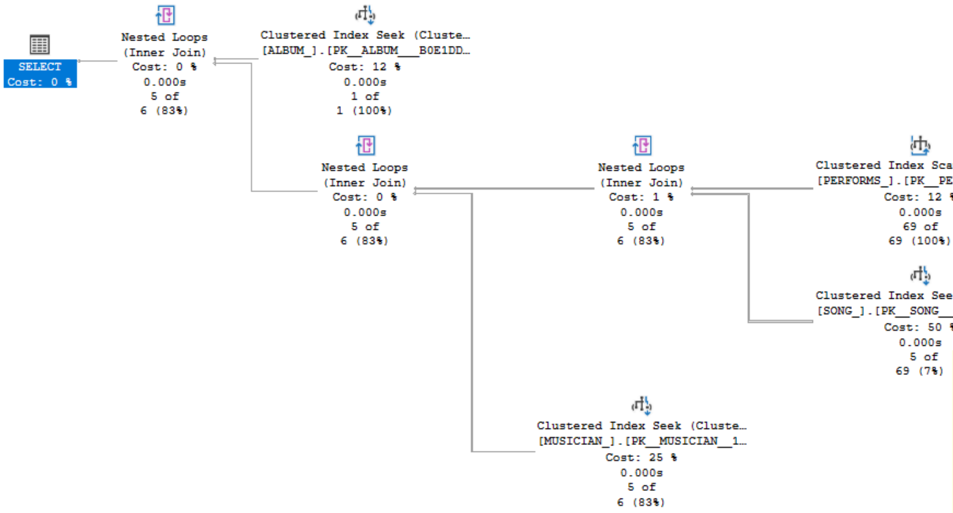
Physical Operation	Clustered Index Seek
Logical Operation	Clustered Index Seek
Actual Execution Mode	Row
Estimated Execution Mode	Row
Storage	RowStore
Number of Rows Read	8
Actual Number of Rows for All Executions	8
Actual Number of Batches	0
Estimated I/O Cost	0.003125
Estimated Operator Cost	0.0032908 (21%)
Estimated CPU Cost	0.0001658
Estimated Subtree Cost	0.0032908
Estimated Number of Executions	1
Number of Executions	1
Estimated Number of Rows Per Execution	8
Estimated Number of Rows to be Read	8
Estimated Row Size	12 B
Actual Rebinds	0
Actual Rewinds	0
Ordered	True
Node ID	4

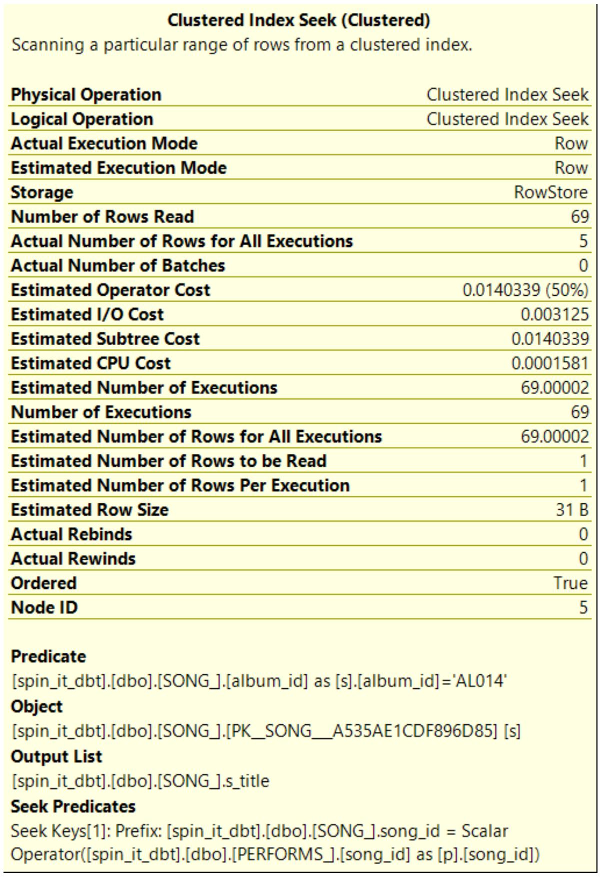
Object
[spin_it_dbt].[dbo].[PERFORMS_].
[PK__PERFORMS__CEEBA05BCD5E9674] [p]
Output List
[spin_it_dbt].[dbo].[PERFORMS_].song_id
Seek Predicates
Seek Keys[1]: Prefix: [spin_it_dbt].[dbo].[PERFORMS_].musician_id =
Scalar Operator('SIMUS003')

If we change the select criteria,

```
--based on an album, display all musicians who sing songs of that album, along with the
names of those songs
select m.musician_id,m.stage_name,s.s_title,a.a_title
from musician_ m,performs_ p,song_ s,album_ a
where a.album_id ='AL014'
        and p.musician_id = m.musician_id
        and p.song_id = s.song_id
        and s.album_id = a.album_id;
```

we get the following plan





Clustered Index Seek (Clustered)

Scanning a particular range of rows from a clustered index.

Physical Operation	Clustered Index Seek
Logical Operation	Clustered Index Seek
Actual Execution Mode	Row
Estimated Execution Mode	Row
Storage	RowStore
Number of Rows Read	69
Actual Number of Rows for All Executions	5
Actual Number of Batches	0
Estimated Operator Cost	0.0140339 (50%)
Estimated I/O Cost	0.003125
Estimated Subtree Cost	0.0140339
Estimated CPU Cost	0.0001581
Estimated Number of Executions	69.00002
Number of Executions	69
Estimated Number of Rows for All Executions	69.00002
Estimated Number of Rows to be Read	1
Estimated Number of Rows Per Execution	1
Estimated Row Size	31 B
Actual Rebinds	0
Actual Rewinds	0
Ordered	True
Node ID	5

Predicate
[spin_it_dbt].[dbo].[SONG_].[album_id] as [s].[album_id]='AL014'
Object
[spin_it_dbt].[dbo].[SONG_].[PK__SONG__A535AE1CDF896D85] [s]
Output List
[spin_it_dbt].[dbo].[SONG_].s_title
Seek Predicates
Seek Keys[1]: Prefix: [spin_it_dbt].[dbo].[SONG_].song_id = Scalar Operator([spin_it_dbt].[dbo].[PERFORMS_].[song_id] as [p].[song_id])

Once again, it can be seen that the select criteria, a.album_id='AL014' is pushed to the song_ relation in order to limit the number of tuples being produced in the first step, hence, making it a better plan, as only songs belonging to the specified album are selected.

Thus, the optimizer seems to be selecting the best possible plan for join orders, while pushing the select criteria to the appropriate level to minimize the number of tuples being generated at each step.