

PES UNIVERSITY

ELECTIVE 1: DATABASE TECHNOLOGIES (UE18CS315)

ASSIGNMENT 2

NAME: SHAAZIN SHEIKH SHUKOOR

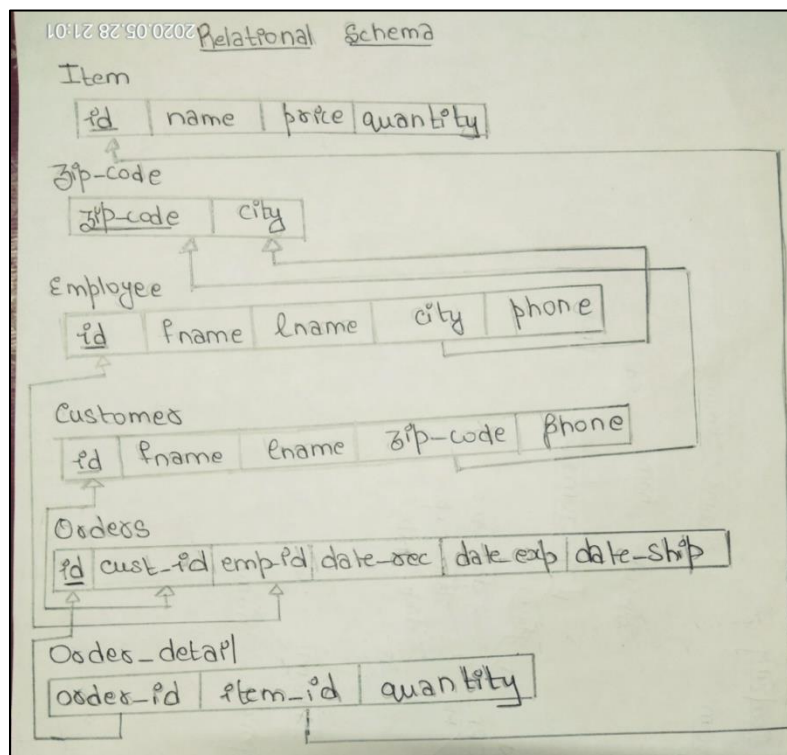
SRN: PES1201801754

SEMESTER: 5

SECTION: J

Problem statement: Understand the performance improvement of queries by writing it in different ways and comparing the execution plans

E-Commerce Company



```

7.1.sql - IDEA-PC\SQL... (idea-PC\user (53))
commit transaction T6
-----
dbcc checktable(item)
go
dbcc checktable(zip_code)
go
dbcc checktable(customer)
go
dbcc checktable(employee)
go
dbcc checktable(orders)
go
-----
100 %
Messages
DBCC results for 'item'.
There are 10000 rows in 58 pages for object "item".
DBCC execution completed. If DBCC printed error messages, contact your system administrator.
DBCC results for 'zip_code'.
There are 1000000 rows in 3819 pages for object "zip_code".
DBCC execution completed. If DBCC printed error messages, contact your system administrator.
DBCC results for 'customer'.
There are 100000 rows in 728 pages for object "customer".
DBCC execution completed. If DBCC printed error messages, contact your system administrator.
DBCC results for 'employee'.
There are 1000000 rows in 7167 pages for object "employee".
DBCC execution completed. If DBCC printed error messages, contact your system administrator.
DBCC results for 'orders'.
There are 1000000 rows in 5587 pages for object "orders".
DBCC execution completed. If DBCC printed error messages, contact your system administrator.
DBCC results for 'order_detail'.
There are 1000000 rows in 4465 pages for object "order_detail".
DBCC execution completed. If DBCC printed error messages, contact your system administrator.
-----
100 %
Query executed successfully.
IDEA-PC\SSQL (14.0 RTM) | idea-PC\user (53) | DT3 | 00:00:08 | 0 rows

```

Size of tables: item < customer < orders, order_detail, zip_code, employee

Use case 1: SQL syntax using NOT IN

1) except

```

--query
SET STATISTICS IO ON
SET STATISTICS TIME ON

DBCC DROPCLEANBUFFERS
GO
DBCC FREEPROCCACHE
GO

use [DT3]
select t.quantity
from
(
select od.item_id,od.quantity
from order_detail od
except
select i.id,i.quantity
from item i
)t
where t.quantity <= 900000
go

```

```

SET STATISTICS IO OFF
SET STATISTICS TIME OFF

```

9_use_case3_1.sql -... (idea-PC\user (58))

```

(
select od.item_id,od.quantity
from order_detail od
except
select i.id,i.quantity
from item i
)t
where t.quantity <= 900000
go

```

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

select t.quantity from (select od.item_id,od.quantity from order_detail od except as
Missing Index (Impact 9.7275): CREATE NONCLUSTERED INDEX [<Name of Missing Index, sys

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	1000000
Actual Number of Rows for All Executions	900000
Actual Number of Batches	0
Estimated I/O Cost	3.30979
Estimated Operator Cost	4.40995 (11%)
Estimated CPU Cost	1.10016
Estimated Subtree Cost	4.40995
Number of Executions	1
Estimated Number of Executions	1
Estimated Number of Rows Per Execution	899261
Estimated Number of Rows to be Read	1000000
Estimated Row Size	25 B
Actual Rebinds	0
Actual Rewinds	0
Ordered	False
Node ID	3

Predicate
[DT3].[dbo].[order_detail].[quantity] as [od].[quantity] <=(900000.)

Object
[DT3].[dbo].[order_detail].[PK_order_de_837942D48F938519] [od]

Output List
[DT3].[dbo].[order_detail].[item_id], [DT3].[dbo].[order_detail].[quantity]

Query executed successfully.

II) not in

```

--query
SET STATISTICS IO ON
SET STATISTICS TIME ON

DBCC DROPCLEANBUFFERS
GO
DBCC FREEPROCCACHE
GO

use [DT3]
select t.quantity
from
(
select od.item_id,od.quantity
from order_detail od
where od.item_id not in
(
select i.id
from item i
)
)t
where t.quantity <= 900000
go

SET STATISTICS IO OFF
SET STATISTICS TIME OFF

```

9_use_case3_3.sql - (idea-PC\user (53)) 9_use_case3_2.sql - (idea-PC\user (51)) 9_use_case3_1.sql - (idea-PC\user (58))

```

where od.item_id not in
(
select i.id
from item i
)
)t
where t.quantity <= 900000
go

```

100 %

Results Messages Execution plan

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

select t.quantity from (select od.item_id,od.quantity from order_

Missing Index (Impact 33.3158): CREATE NONCLUSTERED INDEX [<Name

Hash Match (Right Anti Semi Join)

Cost: 0 %

Cost: 63 %

1.388s

890000 of

890670 (99%)

Index Scan (NonClustered)

[item].[i_id_idx] [i]

Cost: 0 %

0.029s

10000 of

10000 (100%)

Clustered Index Scan (Clustered)

[order_detail].[PK_order_de

Cost: 37 %

1.033s

900000 of

899261 (100%)

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	1000000
Actual Number of Rows for All Executions	900000
Actual Number of Batches	0
Estimated I/O Cost	3.30979
Estimated Operator Cost	4.40995 (37%)
Estimated CPU Cost	1.10016
Estimated Subtree Cost	4.40995
Number of Executions	1
Estimated Number of Executions	1
Estimated Number of Rows Per Execution	899261
Estimated Number of Rows to be Read	1000000
Estimated Row Size	25 B
Actual Rebinds	0
Actual Rewinds	0
Ordered	False
Node ID	2

Predicate

[DT3].[dbo].[order_detail].[quantity] as [od].[quantity]<=(900000.)

Object

[DT3].[dbo].[order_detail].[PK_order_de_837942D48F938519] [od]

Output List

[DT3].[dbo].[order_detail].item_id, [DT3].[dbo].[order_detail].quantity

Query executed successfully.

00:00:07 | 890,000 rows

III) not exists

```

--query
SET STATISTICS IO ON
SET STATISTICS TIME ON

DBCC DROPCLEANBUFFERS
GO
DBCC FREEPROCCACHE
GO

use [DT3]
select t.quantity
from
(
select od.item_id,od.quantity
from order_detail od
where not exists
(
select i.id,i.quantity
from item i
where od.item_id=i.id
)
)t
where t.quantity <= 900000
go

SET STATISTICS IO OFF
SET STATISTICS TIME OFF

```

9_use_case3_4.sql -... (idea-PC\user (55)) 9_use_case3_3.sql -... (idea-PC\user (53)) 9_use_case3_2.sql -... (idea-PC\user (51)) 9_use_case3_1.sql -... (idea-PC\user (58))

```

use [DT3]
select t.quantity
from
(
select od.item_id, od.quantity
from order_detail od
where not exists

```

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

Missing Index (Impact 33.3158): CREATE NONCLUSTERED INDEX [<Ne

Hash Match (Right Anti Semi Join)
Cost: 63 %
1.328s
890000 of 890670 (99%)

Index Scan (NonClus [item].[i_id_idx])
Cost: 0 %
0.031s
10000 of 10000 (100%)

Clustered Index Scan (C [order_detail].[PK_order_detail])
Cost: 37 %
0.923s
900000 of 899261 (100%)

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	1000000
Actual Number of Rows for All Executions	900000
Actual Number of Batches	0
Estimated I/O Cost	3.30979
Estimated Operator Cost	4.40995 (37%)
Estimated CPU Cost	1.10016
Estimated Subtree Cost	4.40995
Number of Executions	1
Estimated Number of Executions	1
Estimated Number of Rows Per Execution	899261
Estimated Number of Rows to be Read	1000000
Estimated Row Size	25 B
Actual Rebinds	0
Actual Rewinds	0
Ordered	False
Node ID	2

Predicate
[DT3].[dbo].[order_detail].[quantity] as [od].[quantity] <= (900000.)

Object
[DT3].[dbo].[order_detail].[PK_order_detail_837942D48F938519].[od]

Output List
[DT3].[dbo].[order_detail].[item_id], [DT3].[dbo].[order_detail].[quantity]

Query executed successfully.

DT3 | 00:00:08 | 890,000 rows

IV) outer apply

```

--query
SET STATISTICS IO ON
SET STATISTICS TIME ON

DBCC DROP CLEANBUFFERS
GO
DBCC FREEPROCCACHE
GO

use [DT3]
select t.quantity
from
(
select od.item_id, od.quantity
from order_detail od
outer apply
(
select i.id, i.quantity
from item i
where od.item_id = i.id
) as h
where h.id is NULL
)t
where t.quantity <= 900000
go

SET STATISTICS IO OFF
SET STATISTICS TIME OFF

```

9_use_case3_5.sql -... (idea-PC\user (56)) 9_use_case3_4.sql -... (idea-PC\user (55)) 9_use_case3_3.sql -... (idea-PC\user (53)) 9_use_case3_2.sql -... (idea-PC\user (51))

```

(
select od.item_id,od.quantity
from order_detail od
outer apply
(
select i.id,i.quantity
from item i
where od.item_id=i.id
)as h
where h.id is NULL
)t
where t.quantity <= 900000

```

100 %

Results Messages Execution plan

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

select t.quantity from (select od.item_id,od.quantity from order_detail od outer apply

Missing Index (Impact 31.9891): CREATE NONCLUSTERED INDEX [<Name of Missing Index>, sysna

Index Scan (NonClustered)

Scan a nonclustered index, entirely or only a range.

Physical Operation	Index Scan
Logical Operation	Index Scan
Actual Execution Mode	Row
Estimated Execution Mode	Row
Storage	RowStore
Number of Rows Read	10000
Actual Number of Rows for All Executions	10000
Actual Number of Batches	0
Estimated I/O Cost	0.0164583
Estimated Operator Cost	0.0276153 (0%)
Estimated CPU Cost	0.011157
Estimated Subtree Cost	0.0276153
Number of Executions	1
Estimated Number of Executions	1
Estimated Number of Rows Per Execution	10000
Estimated Number of Rows to be Read	10000
Estimated Row Size	16 B
Actual Rebinds	0
Actual Rewinds	0
Ordered	False
Node ID	3
Object	[DT3].[dbo].[item].[i_id_idx] [i]
Output List	[DT3].[dbo].[item].id

Query executed successfully. IDEA-PC/SSQ

V) left outer join

```

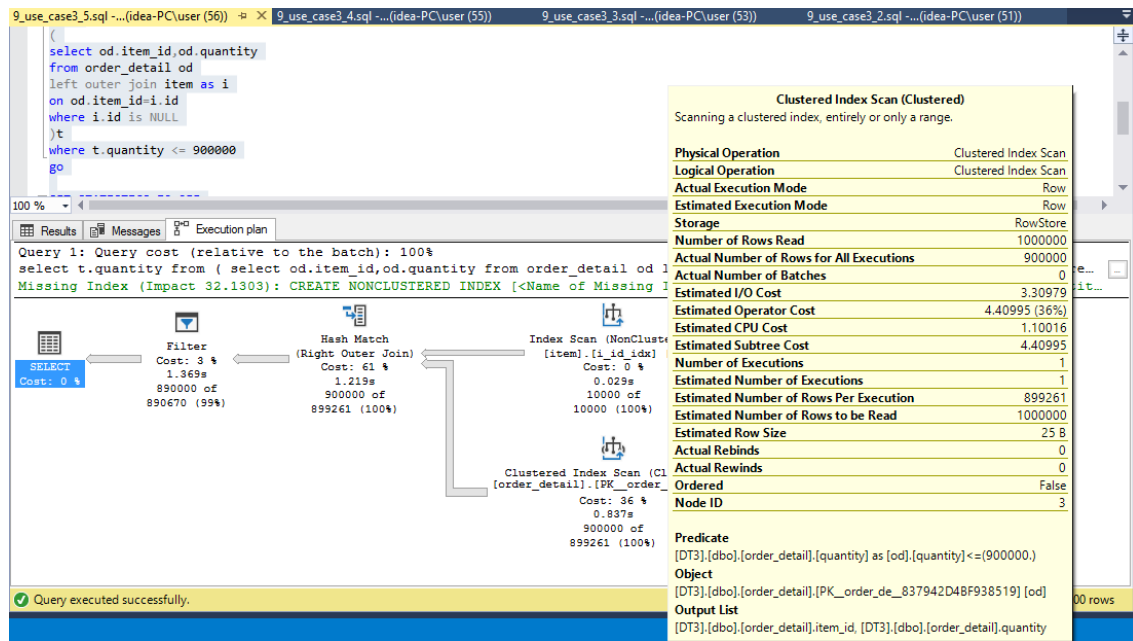
--query
SET STATISTICS IO ON
SET STATISTICS TIME ON

DBCC DROPCLEANBUFFERS
GO
DBCC FREEPROCCACHE
GO

use [DT3]
select t.quantity
from
(
select od.item_id,od.quantity
from order_detail od
left outer join item as i
on od.item_id=i.id
where i.id is NULL
)t
where t.quantity <= 900000
go

SET STATISTICS IO OFF
SET STATISTICS TIME OFF

```



Comparing each of the five different types of writing 'NOT IN':

	<u>Type 1</u>	<u>Type 2</u>	<u>Type 3</u>	<u>Type 4</u>	<u>Type 5</u>
<u>Operator</u>	except	Not in	Not exists	Outer apply	Left outer join
<u>Rows</u>	890,000	890,000	890,000	890,000	890,000
<u>Time</u>	25 sec	7 sec	8 sec	8 sec	8 sec
<u>Logical reads</u>	60	21	21	21	21
<u>Selection criterion</u>	Quantity <= 90000	Quantity <= 90000	Quantity <= 90000	Quantity <= 90000	Quantity <= 90000
<u>Selection criteria pushed to</u>	Item, order_detail	Order_detail	Order_detail	Order_detail	Order_detail
<u>Remarks</u>	Uses explicit distinct. Plan similar to not in.	Bad choice if target column is nullable. Uses right anti-semi join	Prone to nulls and duplicates. Plans similar to not in and except.	Uses right outer join. More expensive as it brings all matching and unmatched rows and then filters.	Similar plan as outer apply. Need to be careful while selecting columns.

Inference:

- While executing queries, only those columns are retrieved, that needs to be projected in the result set
- Select criteria on a particular relation, were pushed into the inner queries.
- The logical and physical reads on all the tables in each of the case were similar except in type 1 (i.e., in except the logical reads were higher)
- The filters were never specified in the sub queries but we see that the selection criteria's were pushed and the sub queries were eliminated.
- The plan generated for (type 1, type 2, type 3 are similar and type 4, type 5 had similar plans).
- The estimated cost followed the following order:
(type 2, type 3 < type 5, type 6 < type)
- Overall we see that the 'not exists' (type 3) has a better performance to its alternatives.

Therefore we see how a query optimizer makes plan for each of the query. It is also intelligent to make an appropriate logical by pushing down the selection criteria to make the relations result fewer tuples and for effective query execution.

Use case 2:

SQL using UNION and confirming that selection criteria are pushed to the appropriate level

1) selection criterion directly applied and no union used

```
query
SET STATISTICS IO ON
SET STATISTICS TIME ON

DBCC DROPCLEANBUFFERS
GO
DBCC FREEPROCCACHE
GO

USE [DT3]
GO

select c.fname,
       c.zip_code,
       sum(od.quantity) as amount
from   customer c,
       order_detail od,
       orders o
where  c.id=o.cust_id and
       c.id between 90 and 95 and
       o.date_rec between '2020-09-01' and '2020-09-02'
group by c.fname,c.zip_code
order by 3 desc
go

SET STATISTICS IO OFF
SET STATISTICS TIME OFF
```


8_3.sql - IDEA-PC\S... (idea-PC\user (56)) 8_sql - IDEA-PC\S... (idea-PC\user (55)) 8_use_case2_sql - L... (idea-PC\user (53))

20 %

Results Messages Execution plan

Query 1: Query cost (relative to the batch): 100%
 select c.fname, c.zip_code, sum(od.quantity) as amount from customer c, order_detail
 Missing Index (Impact 51.2889): CREATE NONCLUSTERED INDEX [Name of Missing Index, s]

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		1000000
Actual Number of Rows for All Executions		6
Actual Number of Batches		0
Estimated I/O Cost		4.1409
Estimated Operator Cost		5.24106 (45%)
Estimated CPU Cost		1.10016
Estimated Subtree Cost		5.24106
Number of Executions		1
Estimated Number of Executions		1
Estimated Number of Rows Per Execution		6.30966
Estimated Number of Rows to be Read		1000000
Estimated Row Size		19 B
Actual Rebinds		0
Actual Rewinds		0
Ordered		False
Node ID		7

Predicate
 [DT3].[dbo].[orders].[cust_id] as [o].[cust_id]>=(90.) AND [DT3].[dbo].[orders].[cust_id] as [o].[cust_id]<=(95.) AND [DT3].[dbo].[orders].[date_rec] as [o].[date_rec]>='2020-09-01' AND [DT3].[dbo].[orders].[date_rec] as [o].[date_rec]<='2020-09-02'

Object
 [DT3].[dbo].[orders].[PK_orders_3213E83FEFC942BD] [o]

Output List
 [DT3].[dbo].[orders].cust_id

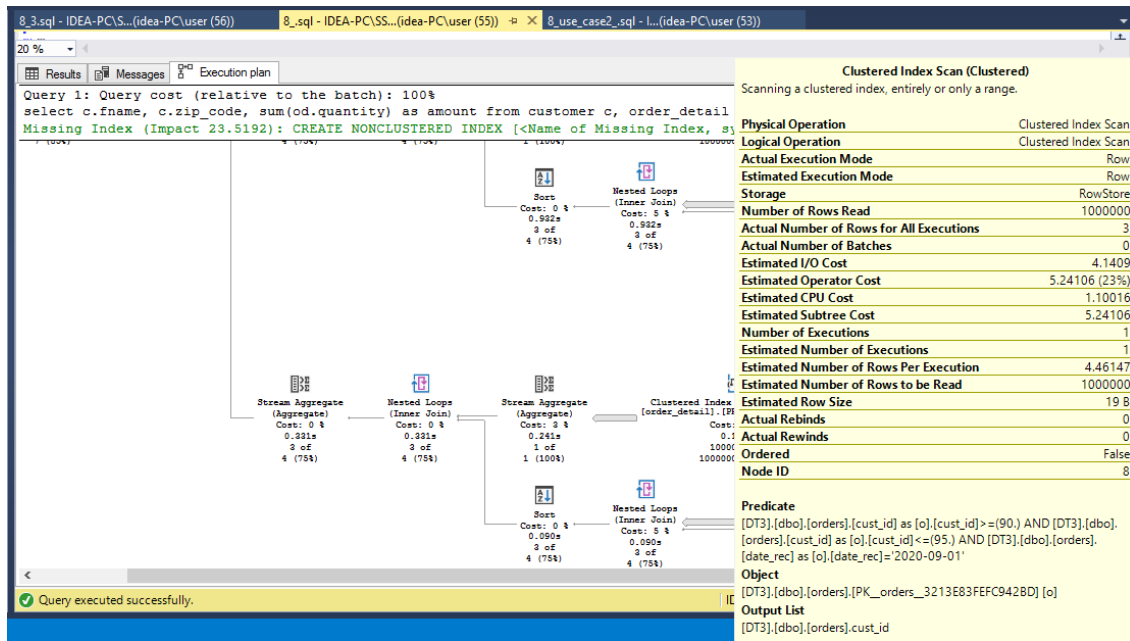
Query executed successfully.

II) selection criterion applied and union operator used

```
GO

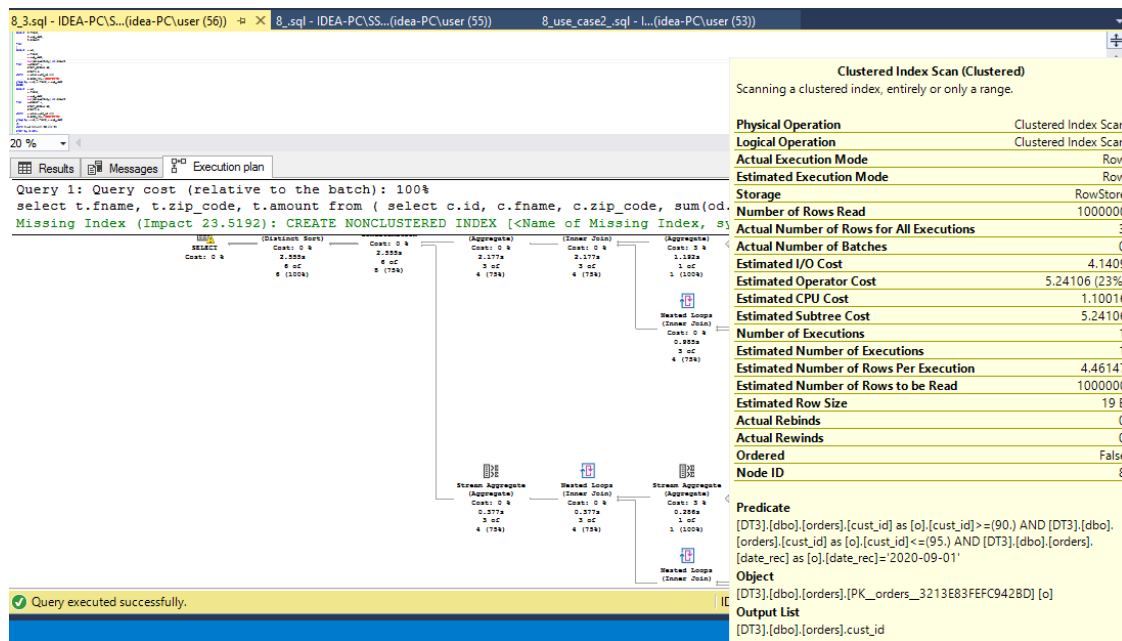
select c.fname,
       c.zip_code,
       sum(od.quantity) as amount
from   customer c,
       order_detail od,
       orders o
where  c.id=o.cust_id and
       c.id between 90 and 95 and
       o.date_rec='2020-09-01'
group by c.fname,c.zip_code
UNION
select c.fname,
       c.zip_code,
       sum(od.quantity) as amount
from   customer c,
       order_detail od,
       orders o
where  c.id=o.cust_id and
       c.id between 90 and 95 and
       o.date_rec='2020-09-02'
group by c.fname,c.zip_code
order by 3 desc
go

SET STATISTICS IO OFF
```



III) sub queries used, selection criterion applied to outer query, union operator used

```
select t.fname,
       t.zip_code,
       t.amount
from
(
  select c.id,
         c.fname,
         c.zip_code,
         sum(od.quantity) as amount
  from customer c,
       order_detail od,
       orders o
  where c.id=o.cust_id and
        o.date_rec='2020-09-01'
  group by c.id,c.fname,c.zip_code
  UNION
  select c.id,
         c.fname,
         c.zip_code,
         sum(od.quantity) as amount
  from customer c,
       order_detail od,
       orders o
  where c.id=o.cust_id and
        o.date_rec='2020-09-02'
  group by c.id,c.fname,c.zip_code
)t
where t.id between 90 and 95
order by 3 desc
```



Comparing each of the three different types of writing 'UNION':

	Type 1	Type 2	Type 3
Rows	6	6	6
Time	3 sec	2 sec	5 sec
Selection criteria	Customer id, date_rec	Customer id, date_rec	Customer id, date_rec
Select criteria pushed to	Orders, customer	Orders, customer	Orders, customer
Remarks	The selection criteria were pushed on the tables to be joined	In both part of the union query, the selection criteria were pushed on the tables to be joined	The selection criterions for the outer query were pushed to the inner queries. It's the worst case if the filters weren't pushed.

Inference:

- While executing queries, only those columns are retrieved, that needs to be projected in the result set
- Select criteria on a particular relation, were pushed into the inner queries.
- The logical and physical reads on all the tables in each of the case were similar

- The first tables to be scanned were the tables to be joined on which the selection criterion was applied.
- Even though the queries were written in three different ways, the execution plans for all three were identical.
- The filters were never specified in the sub queries but we see that the selection criteria's were pushed and the sub queries were eliminated.
- The last case, which is a poorly written query, is executed efficiently because of the query optimizer.

Therefore we see how a query optimizer is intelligent enough to make an appropriate logical by pushing down the selection criteria to make the relations result fewer tuples and for effective query execution.

Use case 3:

Multi table join (minimum of 4 tables) – Review the join order of the tables

Number of tables joined: 6

I) selection criteria on customer table

```

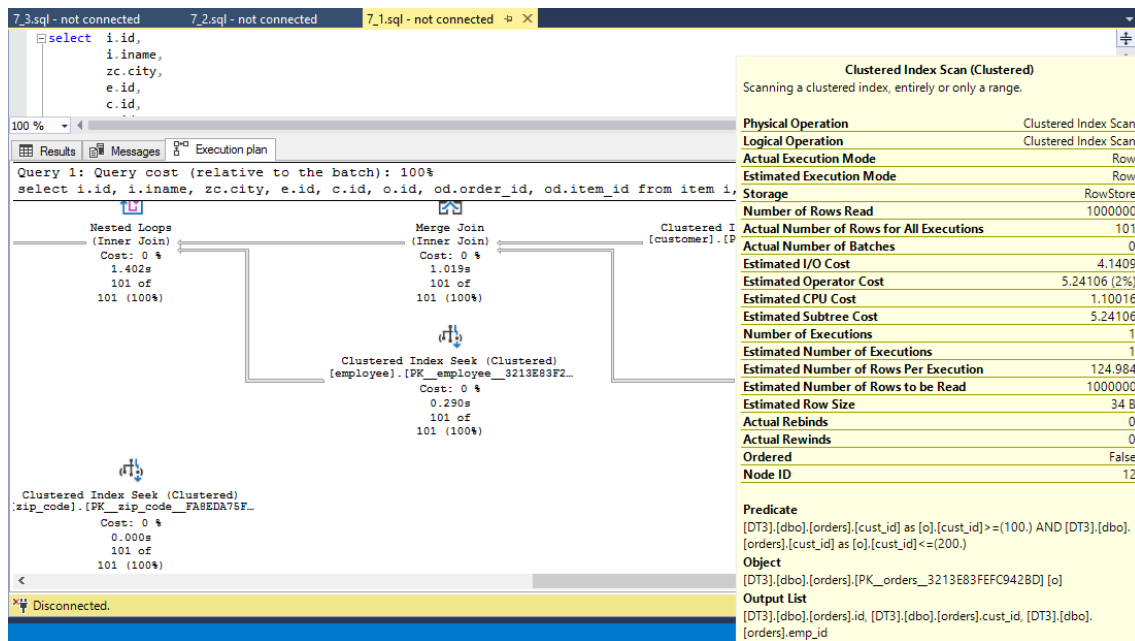
use_case1_join.sql... (idea-PC\user (54))
--query
SET STATISTICS IO ON
SET STATISTICS TIME ON

DBCC DROPCLEANBUFFERS
GO
DBCC FREEPROCCACHE
GO

select i.id,
       i.iname,
       zc.city,
       e.id,
       c.id,
       o.id,
       od.order_id,
       od.item_id
from   item i,
       zip_code zc,
       employee e,
       customer c,
       orders o,
       order_detail od
where  c.id between 100 and 200 and
       c.id=o.cust_id and
       od.item_id=i.id and
       e.id=o.emp_id and
       zc.zip_code=c.zip_code
order by 3 desc
go

SET STATISTICS IO OFF
SET STATISTICS TIME OFF

```

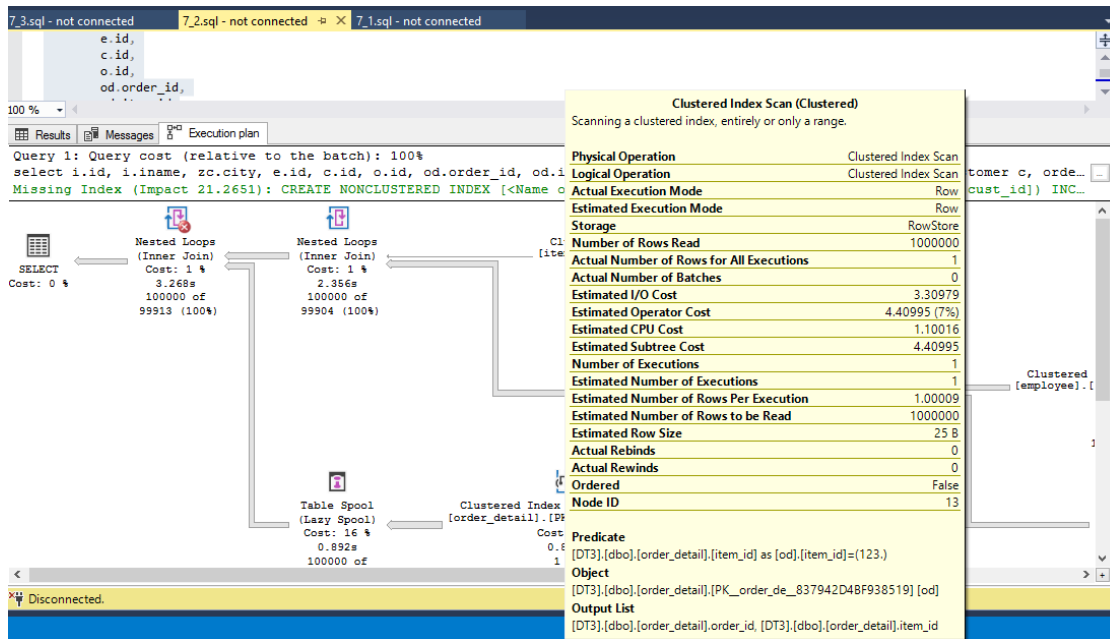


II) selection criteria on item table

```
7_use_case1_join.sql... (idea-PC\user (54))
--query
SET STATISTICS IO ON
SET STATISTICS TIME ON

DBCC DROPCLEANBUFFERS
GO
DBCC FREEPROCCACHE
GO

select i.id,
       i.iname,
       zc.city,
       e.id,
       c.id,
       o.id,
       od.order_id,
       od.item_id
from   item i,
       zip_code zc,
       employee e,
       customer c,
       orders o,
       order_detail od
where  --c.id between 200100 and 200110 and
       c.id=o.cust_id and
       od.item_id=i.id and
       e.id=o.emp_id and
       i.id=123 and
       zc.zip_code=c.zip_code
order by 3 desc
go
```

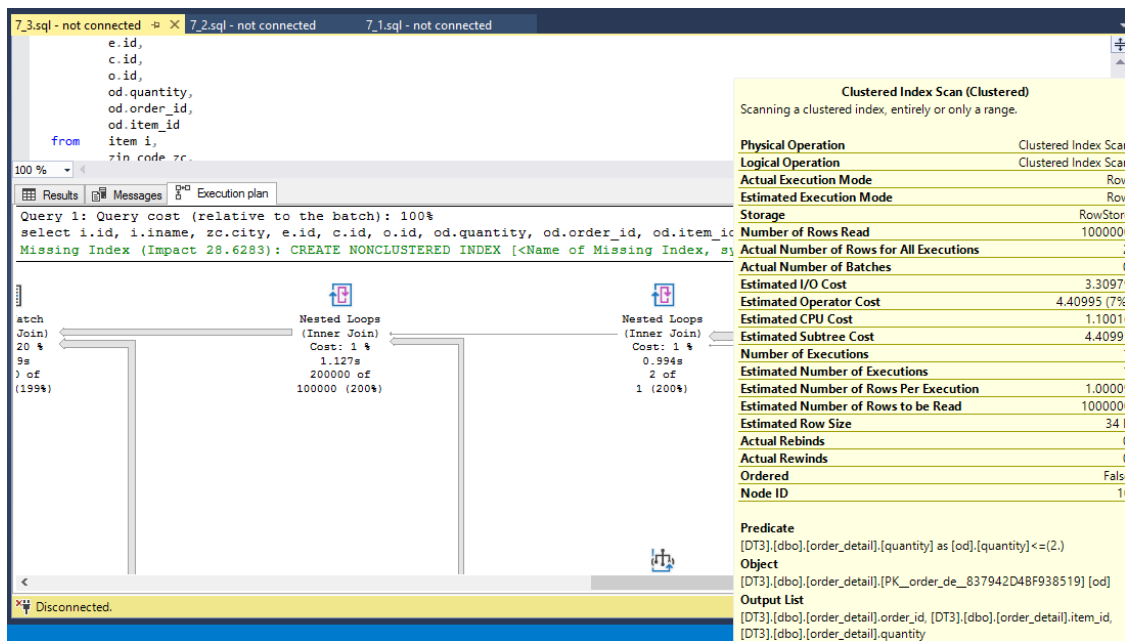


III) selection criteria on order_detail table

```
--query
SET STATISTICS IO ON
SET STATISTICS TIME ON

DBCC DROPCLEANBUFFERS
GO
DBCC FREEPROCCACHE
GO

select i.id,
i.iname,
zc.city,
e.id,
c.id,
o.id,
od.quantity,
od.order_id,
od.item_id
from item i,
zip_code zc,
employee e,
customer c,
orders o,
order_detail od
where --c.id between 200100 and 200110 and
c.id = o.cust_id and
od.item_id = i.id and
e.id = o.emp_id and
--i.id = 123 and
od.quantity <= 2 and
zc.zip_code = c.zip_code
order by 3 desc
go
```



Comparing each of the three 'JOIN' orders:

	Type 1	Type 2	Type 3
Join criteria	Customer id	Item id	Order quantity
Rows	1,010,000	100,000	200,000
Time	28 sec	4 sec	9 sec
First scanned table	customer	Zip_code	Order_detail
Select criteria pushed to	orders	Order_detail	item
Join order	Customer, orders -> employee -> zip_code -> item -> order_detail	Zip_code, customer -> orders -> employee -> item -> order_detail	Order_detail, item -> customer -> orders -> employee -> zip_code

Inference:

- While joining, only those columns are retrieved, that needs to be projected in the result set
- Select criteria on a particular join, was pushed to the table which it was being joined with
- The logical and physical reads on all the tables in each of the case were similar
- The first tables to be scanned and joined were those on which selection criteria was applied. But in the case of 'type 2' query it was not the case.

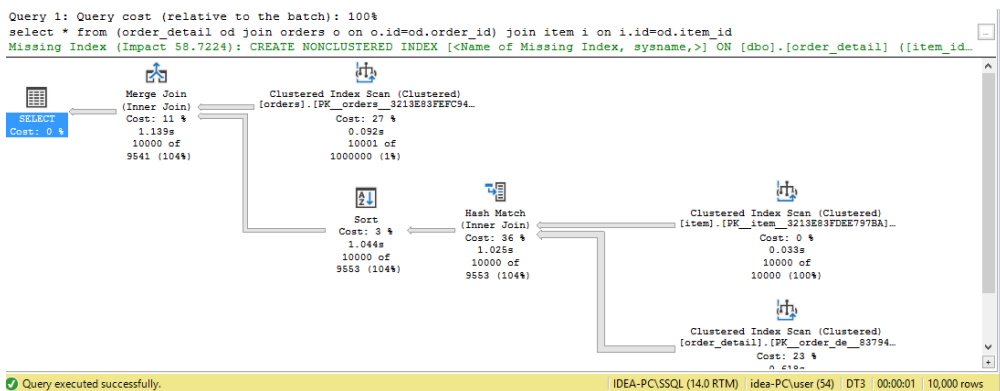
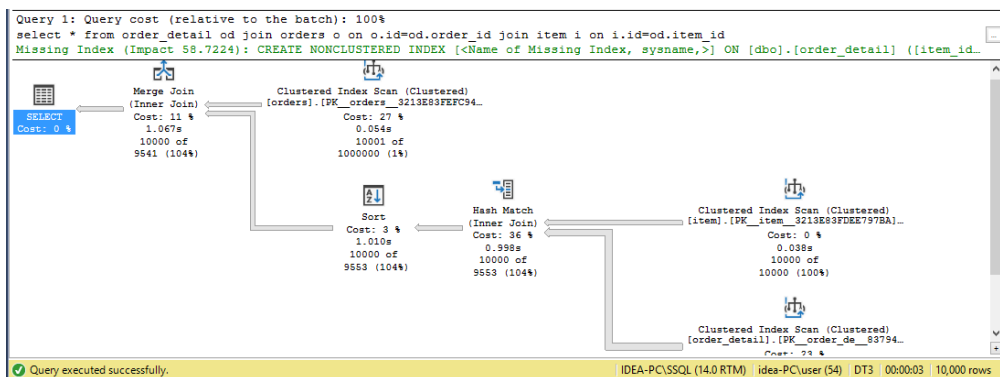
Another simple example showing the join order,

Shown below are two queries that are equal, but a join order was forced on the second query by using a nested join:

The logical reads, no of rows returned and time taken by both are similar.

```
select *
from order_detail od
join orders o on o.id=od.order_id
join item i on i.id=od.item_id
```

```
select *
from (order_detail od join orders o on o.id=od.order_id)
join item i on i.id=od.item_id
```



Even though we have specified different join orders in query, the query optimizer follows the same join order in both the cases.

Observation:

* In the execution plans, we normally see two kinds of joins:

I) nested loop inner join

The Nested Loop Join uses one joining table as an outer input table and the other one as the inner input table. The Nested Loop Join gets a row from the outer table and searches for the row in the inner table; this process continues until all the output rows of the outer table are searched in the inner table.

II) Hash match inner join

The Hash Match represents the building of a hash table of computed hash values from each row in the input

The major difference between a hash join and a nested loops join is the use of a full-table scan with the hash join. For certain types of SQL, the hash join will execute faster than a nested loop join, but the hash join uses more RAM resources.

Nested loop is generally chosen when table is significantly small and the larger table has an index on the join key. Hash Match is also better suited for large tables; it uses a hash table and hash match function to match rows. This requires less I/O, but need more CPU and requires lot of memory.

* One of the most important rules of efficient query processing is to move the selection down the tree as far as they will go without changing what the expression does

Some of the selection rules that we have used were:

- $(\sigma)_{C1 \text{ AND } C2} (R) = (\sigma)_{C1} ((\sigma)_{C2} (R))$
- $(\sigma)_C (R \cup S) = ((\sigma)_C R \cup (\sigma)_C S)$
- $(\sigma)_C (R - S) = ((\sigma)_C R - (\sigma)_C S)$
- $(\sigma)_C (R \text{ JOIN } S) = ((\sigma)_C R \text{ JOIN } (\sigma)_C S)$

Conclusion:

Therefore we see how a query optimizer is intelligent enough to make an appropriate logical plan pertaining to the order and type of join and pushing down the selection criteria to make the relations result fewer tuples and for effective query execution.