



Ζήτηση 1^ο

Η αρχική εκτέλεση του ερωτήματος (χωρίς τη χρήση κάποιου ευρετηρίου) μας δίνει τα εξής αποτελέσματα:

SQLQuery1.sql - GWGWPC\MSSQLSERVER01.RETAILDB (GWGWPC\Gwggw (72)) - Microsoft SQL Server Management Studio

Connect - GWGWPC\MSSQLSERVER01 (SQL Server 16.0.1000)

Object Explorer

SQLQuery1.sql - GWGWPC\Gwggw (72)*

```

checkpoint dbcc dropcleanbuffers

set statistics io, time on

Select cname, orders.orderkey, sum(price) as total
from customers, orders, lineitem
where
customers.custkey = orders.custkey and
lineitem.orderkey = orders.orderkey and
orderdate <= '1993-12-31' and
shipdate between '1994-01-01' and '1994-02-28'
group by cname, orders.orderkey
order by total desc

set statistics io, time off
  
```

100 %

Results Messages Execution plan

DBCC execution completed. If DBCC printed error messages, contact your system administrator.

(53995 rows affected)

Table 'customers'. Scan count 11, logical reads 2686, physical reads 2, page server reads 0, read-ahead reads 2162, page server read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob page server reads 0, lob read-ahead reads 0.

Table 'orders'. Scan count 11, logical reads 16363, physical reads 1, page server reads 0, read-ahead reads 16739, page server read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob page server reads 0, lob read-ahead reads 0.

Table 'lineitem'. Scan count 11, logical reads 60387, physical reads 1, page server reads 0, read-ahead reads 59343, page server read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob page server reads 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 logical reads 0, lob physical reads 0, lob page server reads 0, lob read-ahead reads 0.

(1 row affected)

SQL Server Execution Times:

CPU time = 1616 ms, elapsed time = 803 ms.

Completion time: 2023-05-13T15:57:48.8188656+03:00

100 %

Query executed successfully.

SQLQuery1.sql - GWGWPC\MSSQLSERVER01.RETAILDB (GWGWPC\Gwggw (72)) - Microsoft SQL Server Management Studio

Connect - GWGWPC\MSSQLSERVER01 (SQL Server 16.0.1000)

Object Explorer

SQLQuery1.sql - GWGWPC\Gwggw (72)*

```

checkpoint dbcc dropcleanbuffers

set statistics io, time on

Select cname, orders.orderkey, sum(price) as total
from customers, orders, lineitem
where
customers.custkey = orders.custkey and
lineitem.orderkey = orders.orderkey and
orderdate <= '1993-12-31' and
shipdate between '1994-01-01' and '1994-02-28'
group by cname, orders.orderkey
order by total desc
  
```

100 %

Results Messages Execution plan

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

Select cname, orders.orderkey, sum(price) as total from customers, orders, lineitem where customers.custkey = orders.custkey and lineitem.orderkey = orders.orderkey and orderdate <= '1993-12-31' and shipdate between '1994-01-01' and '1994-02-28' group by cname, orders.orderkey order by total desc

Missing Index (Impact 19.7234): CREATE NONCLUSTERED INDEX [<Name of Missing Index, sysname,>] ON [dbo].[orders] ([orderdate]) INCLUDE ([custkey])

Parallelism (Gather Streams) Cost: 0 % 0.5204 53995 of 11472 (470%)

Sort Cost: 0 % 0.3142 53995 of 11472 (470%)

Hash Match (Inner Join) Cost: 0 % 0.0244 53995 of 11472 (470%)

Hash Match (Aggregate) Cost: 1 % 0.2744 74291 of 37327 (195%)

Clustered Index Scan (Clustered) [lineitem].[PK__lineitem__43B943B1E..] Cost: 75 % 0.2744 148055 of 146743 (100%)

Adaptive Join (Inner Join) Cost: 0 % 0.1235 53995 of 4546 (1187%)

Clustered Index Scan (Clustered) [orders].[PK__orders__721CF6816FAF1..] Cost: 21 % 0.0315 53995 of 4546 (1187%)

Clustered Index Scan (Clustered) [customers].[PK__customer__7398ED57..] Cost: 3 % 0.0014 21341 of 150000 (14%)

Clustered Index Seek (Clustered) [customers].[PK__customer__7398ED57..] Cost: 3 % 0.0004 0 of 4546 (0%)

100 %

Query executed successfully.



Παρατηρούμε δηλαδή ότι έχουμε

Table 'customers'. Scan count 11, logical reads 2685,

Table 'orders'. Scan count 11, logical reads 16961,

Table 'lineitem'. Scan count 11, logical reads 60387.

Για να γίνει γρηγορότερα το groupby cname μπορούμε να δημιουργήσουμε index στον πίνακα customers ως προς το cname.

```
CREATE INDEX index_on_cust ON customers (cname)
```

Γνωρίζουμε ότι κάθε πίνακας έχει ένα clustered index στο primary key του. Αυτό σημαίνει ότι ο πίνακας customers θα έχει ένα clustered index στο custkey. (Αρα δεν χρειάζεται δημιουργία κάποιου index στο custkey του πίνακα customers)

Για την περαιτέρω βελτίωση θα δημιουργήσουμε και ένα index στον πίνακα orders ως προς το orderdate (Για να μπορέσει να γίνει το φιλτράρισμα γρηγορότερα) το οποίο θα περιέχει και το custkey για την βελτιστοποίηση του join .

```
CREATE INDEX index_on_orders ON orders (orderdate) include(custkey)
```

Τέλος, όπως αναφέρθηκε και νωρίτερα γνωρίζουμε ότι κάθε πίνακας έχει ένα clustered index στο primary key του. Αυτό σημαίνει ότι ο πίνακας lineitem θα έχει ένα clustered index στο orderkey.

Αυτό το index βελτιώνει το join με τον πίνακα orders.

Ωστόσο επειδή θέλουμε να γίνει φιλτράρισμα ως προς το shipdate (και να μπορούμε να ανακτήσουμε το price γρηγορότερα) θα δημιουργήσουμε ένα index στον πίνακα lineitem στο shipdate το οποίο θα κάνει include το price.

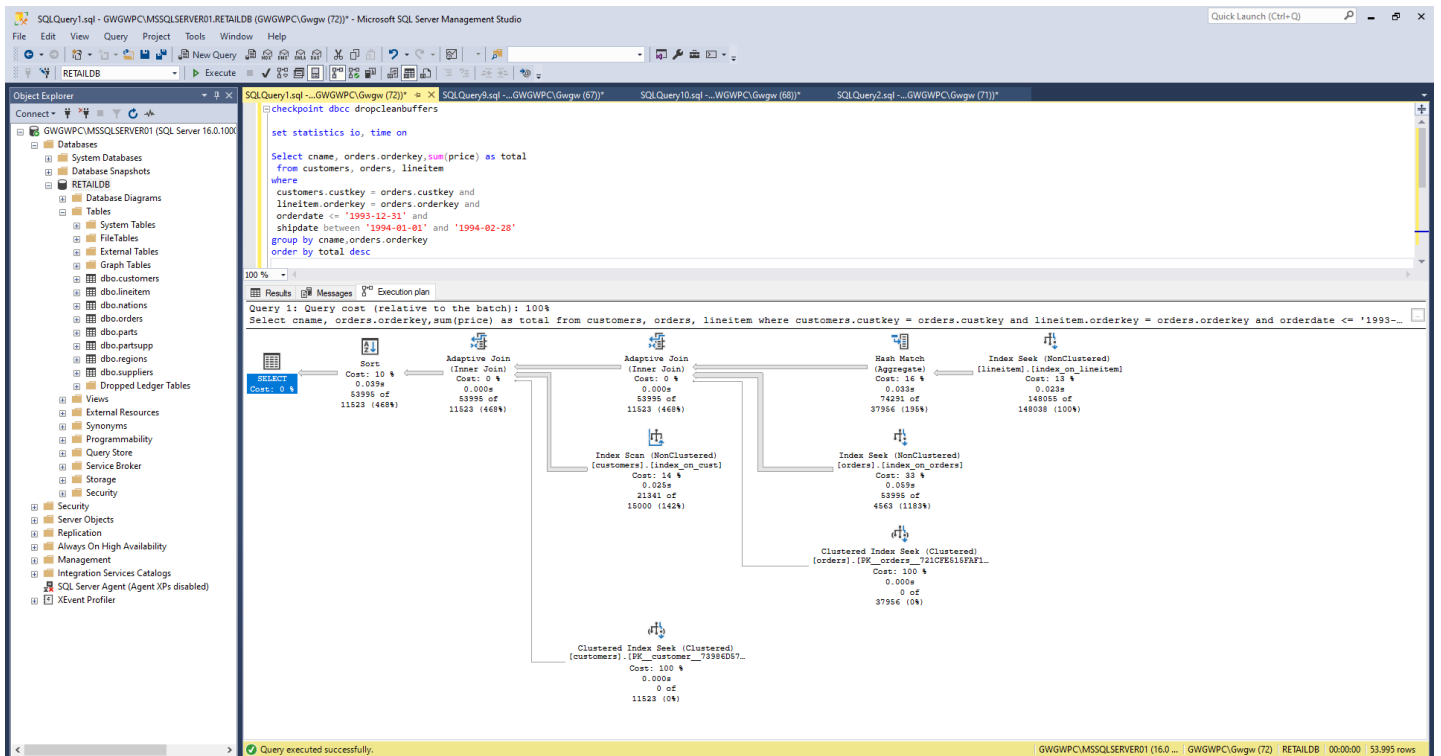
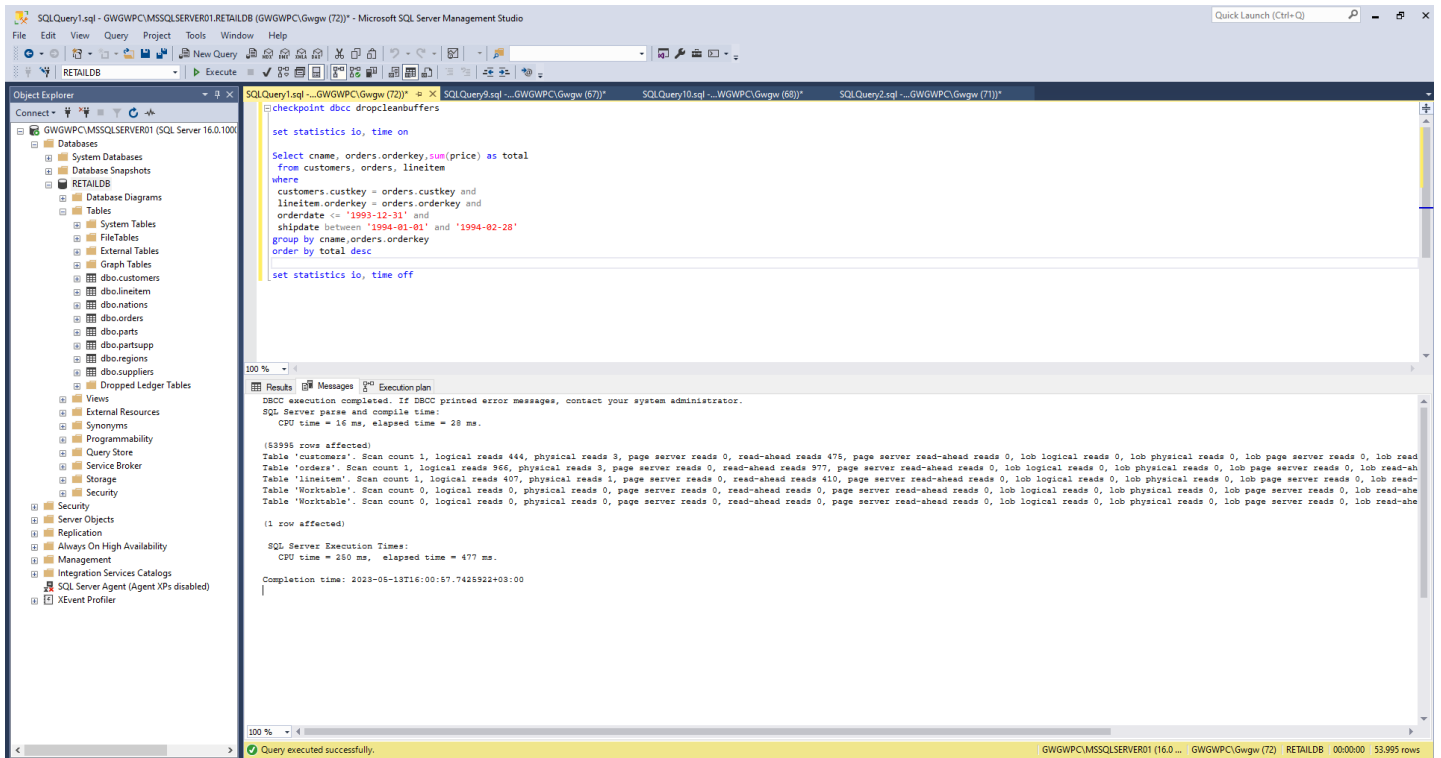
```
CREATE INDEX index_on_lineitem ON lineitem (shipdate) include(price)
```

Μετά από τη δημιουργία αυτών των indexes λοιπόν, παρατηρούμε μεγάλη διαφορά στα logical reads που εκτελεί η επερώτηση. Συγκεκριμένα έχουμε:

Table 'customers'. Scan count 1, logical reads 444,

Table 'orders'. Scan count 1, logical reads 966,

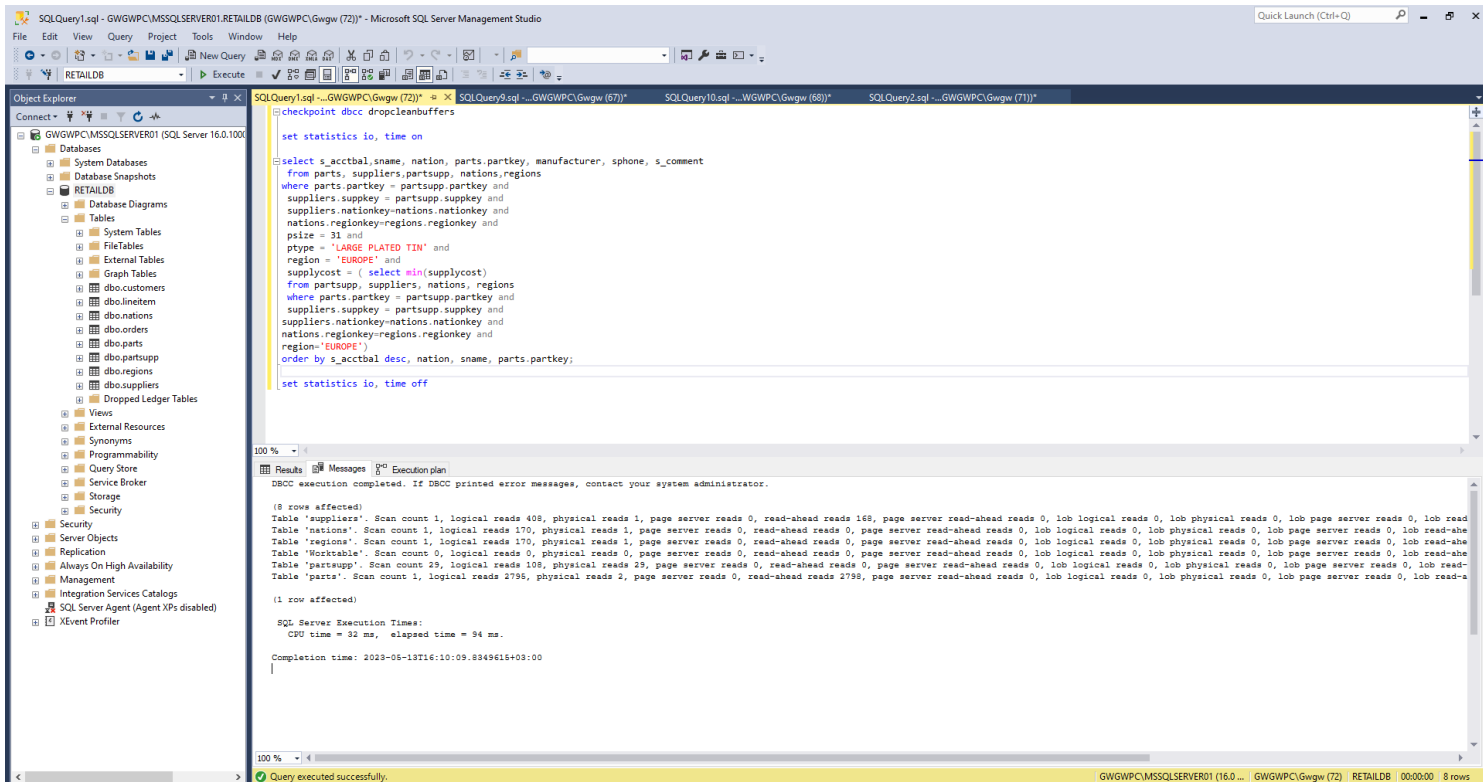
Table 'lineitem'. Scan count 1, logical reads 407.



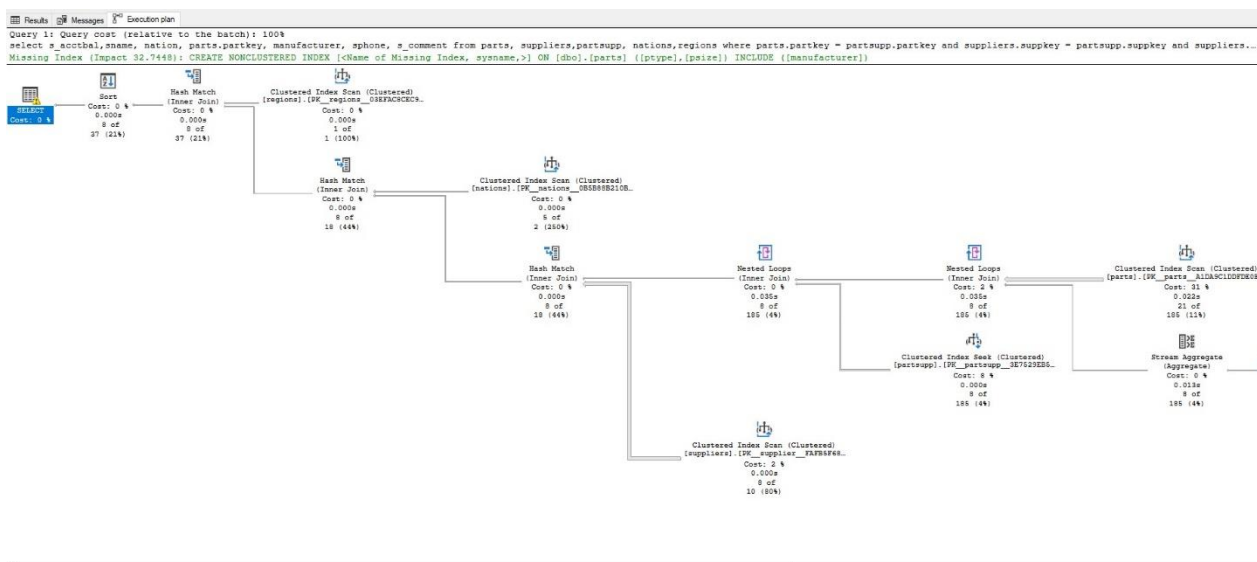


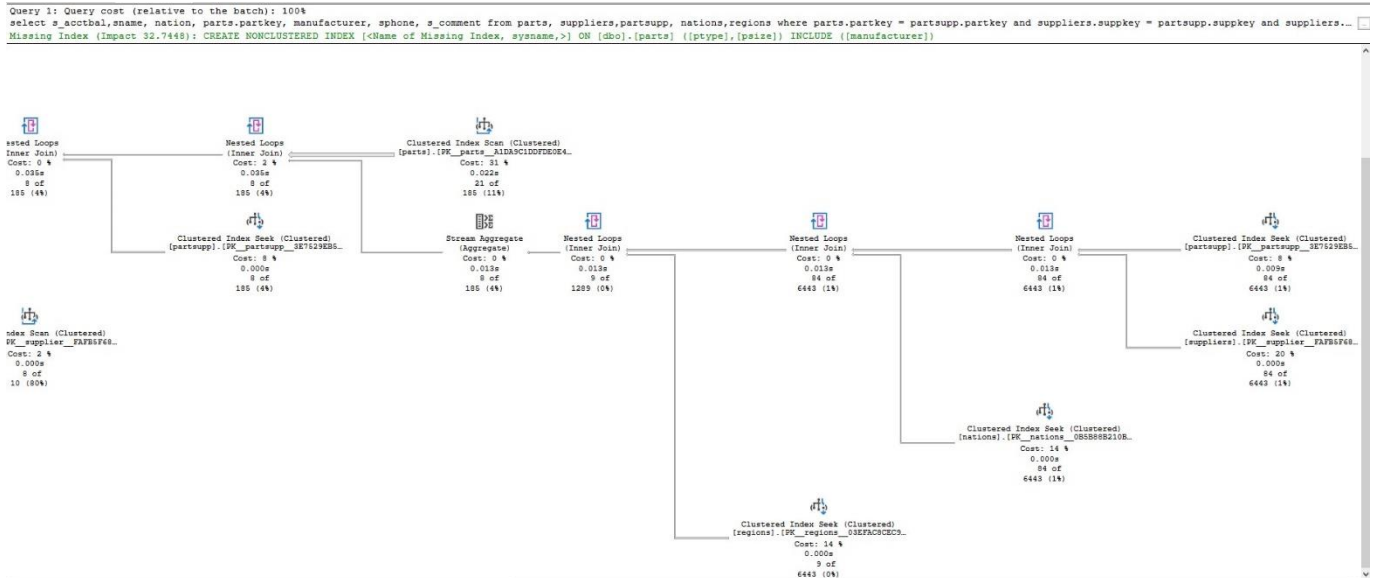
Ζήτημα 2ο

Η αρχική εκτέλεση του ερωτήματος (χωρίς τη χρήση κάποιου ευρετηρίου) μας δίνει τα εξής αποτελέσματα:



όπως και το εξής πλάνο εκτέλεσης





Παρατηρούμε λοιπόν ότι έχουμε:

Table 'regions'. Scan count 0, logical reads 170, physical reads 1,
 Table 'nations'. Scan count 0, logical reads 170, physical reads 1
 Table 'suppliers'. Scan count 0, logical reads 408, physical reads 1,
 Table 'partsupp'. Scan count 29, logical reads 108, physical reads 29,
 Table 'parts'. Scan count 1, logical reads 2795, physical reads 3.

Παρατηρούμε δηλαδή ότι είναι πολλά τα logical reads του πίνακα parts. Επομένως θα δημιουργήσουμε ευρετήριο σε αυτόν.

Στο επερώτημα μας ενδιαφέρει να εντοπίσουμε τους προμηθευτές που μπορούν να προμηθεύσουν την εταιρεία με προϊόντα συγκεκριμένου τύπου "LARGE PLATED IN" και μεγέθους "31", σε μια συγκεκριμένη περιοχή "EUROPE", με το χαμηλότερο κόστος.

Άρα θα χρειαστούμε ένα ευρετήριο στον πίνακα parts ως προς το ptype και επείτα το psize. Επίσης θα κάνει include τους manufacturers (βρίσκονται στο select οπότε είναι καλή ιδέα να μπορούμε να τους ανακτάμε γρήγορα).

Άρα έχουμε:

```
create index index_on_parts on parts (ptype, psize) include (manufacturer)
```

Μετά και τη δημιουργία αυτού του ευρετηρίου λοιπόν έχουμε τα εξής αποτελέσματα:

Table 'regions'. Scan count 0, logical reads 184, physical reads 1,
 Table 'nations'. Scan count 0, logical reads 184, physical reads 1
 Table 'suppliers'. Scan count 0, logical reads 272, physical reads 1,
 Table 'partsupp'. Scan count 29, logical reads 108, physical reads 29,
 Table 'parts'. Scan count 1, logical reads 3, physical reads 3.

Έχουμε δηλαδή μεγάλη μείωση στα logical reads στον πίνακα parts



The screenshot shows the SQL Server Management Studio interface. The Object Explorer on the left displays the database structure, including the RETAILDB database. The main window shows a query window with the following SQL code:

```

--checkpoint
--dbcc dropcleanbuffers

set statistics io, time on

select s_acctbal, sname, nation, parts.partkey, manufacturer, sphone, s_comment
from parts, suppliers, partsupp, nations, regions
where parts.partkey = partsupp.partkey and
suppliers.supkey = partsupp.supkey and
suppliers.nationkey = nations.nationkey and
nations.regionkey = regions.regionkey and
psize = 31 and
ptype = 'LARGE PLATED TIN' and
region = 'EUROPE' and
supplycost = (select min(supplycost)
from partsupp, suppliers, nations, regions
where parts.partkey = partsupp.partkey and
suppliers.supkey = partsupp.supkey and
suppliers.nationkey = nations.nationkey and
nations.regionkey = regions.regionkey and
region = 'EUROPE')
order by s_acctbal desc, nation, sname, parts.partkey;

set statistics io, time off

```

The Results pane shows the execution plan and the following output:

```

DBCC execution completed. If DBCC printed error messages, contact your system administrator.

(8 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 logical reads 0, lob physical reads 0, lob page server reads 0, lob read-ahead reads 0.
Table 'regions'. Scan count 0, logical reads 164, physical reads 1, page server reads 0, read-ahead reads 0, page server read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob page server reads 0, lob read-ahead reads 0.
Table 'nations'. Scan count 0, logical reads 164, physical reads 1, page server reads 0, read-ahead reads 0, page server read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob page server reads 0, lob read-ahead reads 0.
Table 'suppliers'. Scan count 0, logical reads 272, physical reads 1, page server reads 0, read-ahead reads 169, page server read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob page server reads 0, lob read-ahead reads 0.
Table 'partsupp'. Scan count 29, logical reads 106, physical reads 29, page server reads 0, read-ahead reads 0, page server read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob page server reads 0, lob read-ahead reads 0.
Table 'parts'. Scan count 1, logical reads 3, physical reads 3, page server reads 0, read-ahead reads 0, page server read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob page server reads 0, lob read-ahead reads 0.

(1 row affected)

SQL Server Execution Times:
CPU time = 0 ms, elapsed time = 94 ms.

Completion time: 2023-05-14T13:06:37.0778000+03:00

```

The status bar at the bottom indicates: Query executed successfully. GWGWPC\MSSQLSERVER01 (16.0) ... GWGWPC\Gwgv (68) RETAILDB 00:00:00 8 rows

Ζήτηση 3^ο

Η αρχική εκτέλεση του ερωτήματος (χωρίς τη χρήση κάποιου ευρετηρίου) μας δίνει τα εξής αποτελέσματα:

The screenshot shows the SQL Server Management Studio interface. The Object Explorer on the left displays the database structure, including the RETAILDB database. The main window shows a query window with the following SQL code:

```

--checkpoint dbcc dropcleanbuffers

set statistics io, time on

select market_segment, sum(totalprice)
from customers, orders
where customers.custkey=orders.custkey and YEAR(orderdate) = 1996
group by market_segment

set statistics io, time off

```

The Results pane shows the execution plan and the following output:

```

DBCC execution completed. If DBCC printed error messages, contact your system administrator.

(5 rows affected)
Table 'customers'. Scan count 11, logical reads 2686, physical reads 2, page server reads 0, read-ahead reads 2662, page server read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob page server reads 0, lob read-ahead reads 0.
Table 'orders'. Scan count 11, logical reads 16961, physical reads 1, page server reads 0, read-ahead reads 16727, page server read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob page server reads 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 logical reads 0, lob physical reads 0, lob page server reads 0, lob read-ahead reads 0.

(1 row affected)

SQL Server Execution Times:
CPU time = 616 ms, elapsed time = 188 ms.

Completion time: 2023-05-13T16:12:55.8423324+03:00

```

The status bar at the bottom indicates: Query executed successfully. GWGWPC\MSSQLSERVER01 (16.0) ... GWGWPC\Gwgv (72) RETAILDB 00:00:00 5 rows

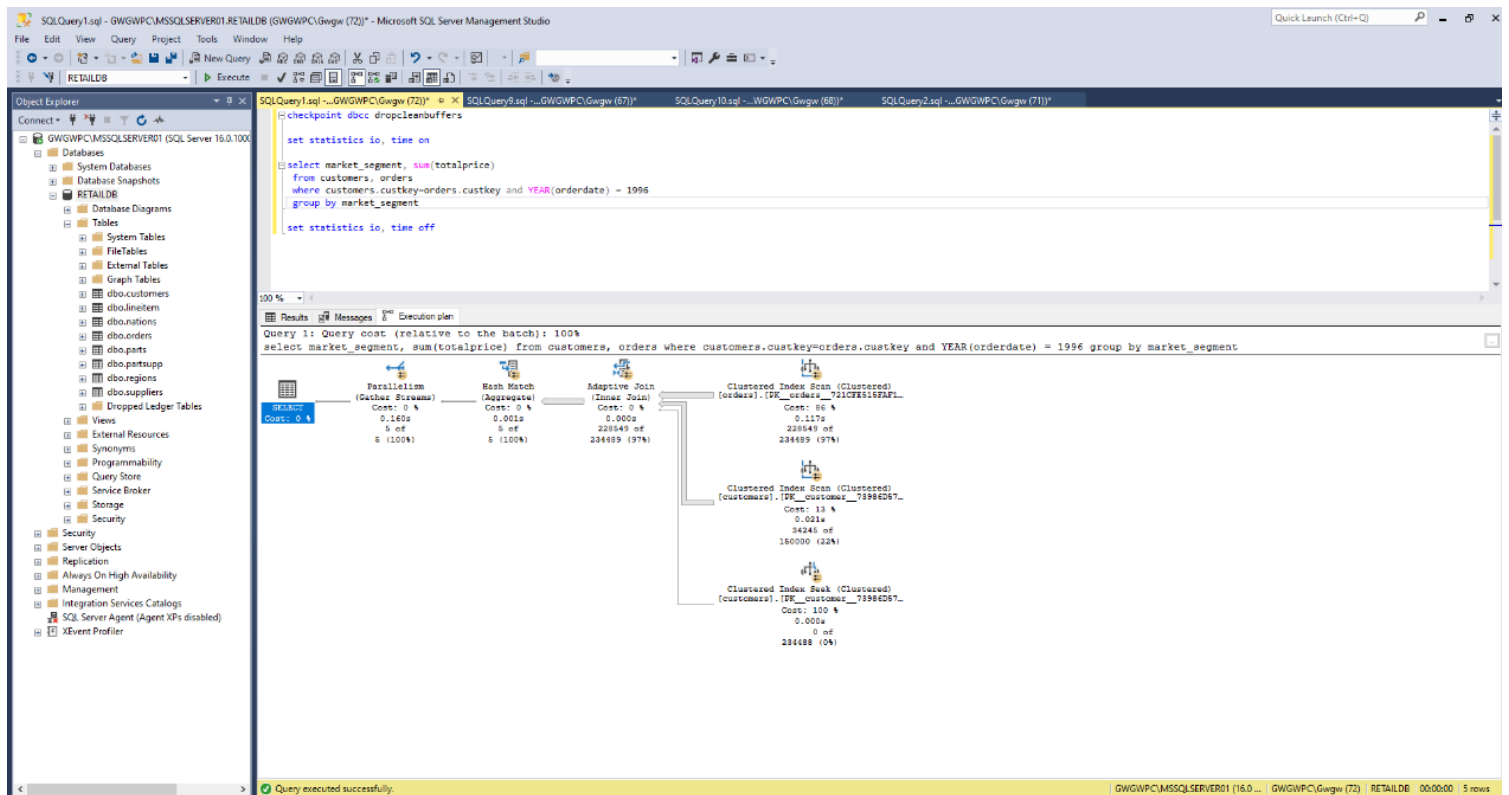


Table 'customers'. Scan count 11, logical reads 2685, physical reads 2,

Table 'orders'. Scan count 11, logical reads 16961,

Αρχικά, θα προσπαθήσουμε να βελτιστοποιήσουμε το ερώτημα με τη χρήση ευρετηρίων. Έτσι, θα προσθέσουμε ένα ευρετήριο στον πίνακα orders ώστε να μπορέσουμε να το χρησιμοποιήσουμε για να φιλτράρουμε τις εγγραφές που έχουν έτος 1996 γρηγορότερα, και επίσης συμπεριλαμβάνουμε και το custkey έτσι ώστε σε περίπτωση ισοτήτας του orderdate να είναι ταξινομημένα ως προς το custkey (το οποίο χρειάζεται και για το join). Επιπλέον θα συμπεριλάβουμε και το totalprice γιατί μας εξυπηρετεί προκειμένου να υπολογιστεί το SUM.

```
create index index_on_orders on orders (orderdate, custkey) include (totalprice)
```

Το επόμενο ευρετήριο που θα βάλουμε είναι στον πίνακα customers, και είναι ταξινομημένο ως προς το market_segment (εφόσον η ομαδοποίηση γίνεται με βάση το market_segment) και το custkey.

```
create index index_on_customers on customers (market_segment, custkey)
```

Επιπλέον, θα προσπαθήσουμε να βελτιστοποιήσουμε το επερώτημα ώστε να γίνει πιο αποδοτικό. Έτσι, έχουμε:



Αρχικά φιλτράρουμε τις εγγραφές ώστε να πάρουμε μόνο όσες έχουν έτος 1996. Για να το επιτύχουμε αυτό δεν χρησιμοποιούμε την συνάρτηση YEAR() διότι αυτή δεν είναι αποδοτική όταν θέλουμε να χρησιμοποιήσουμε indexes

(όταν χρησιμοποιούμε τη συνάρτηση year() σε where clause η βάση θα χρειαστεί να κάνει full table scan για να υπολογίσει το year() σε κάθε row).

Έπειτα κάνουμε το join στους 2 πίνακες ως προς το custkey, έπειτα την ομαδοποίηση ανά market_segment και τον υπολογισμό του sum(totalprice)

Το επερώτημα λοιπόν που δημιουργήθηκε είναι το εξής:

```
SELECT customers.market_segment, SUM(filtered_orders.totalprice)
FROM customers
JOIN (select totalprice, custkey
from orders
WHERE orderdate >= '1996-01-01' AND orderdate < '1997-01-01') AS
filtered_orders ON customers.custkey = filtered_orders.custkey
GROUP BY market_segment
```

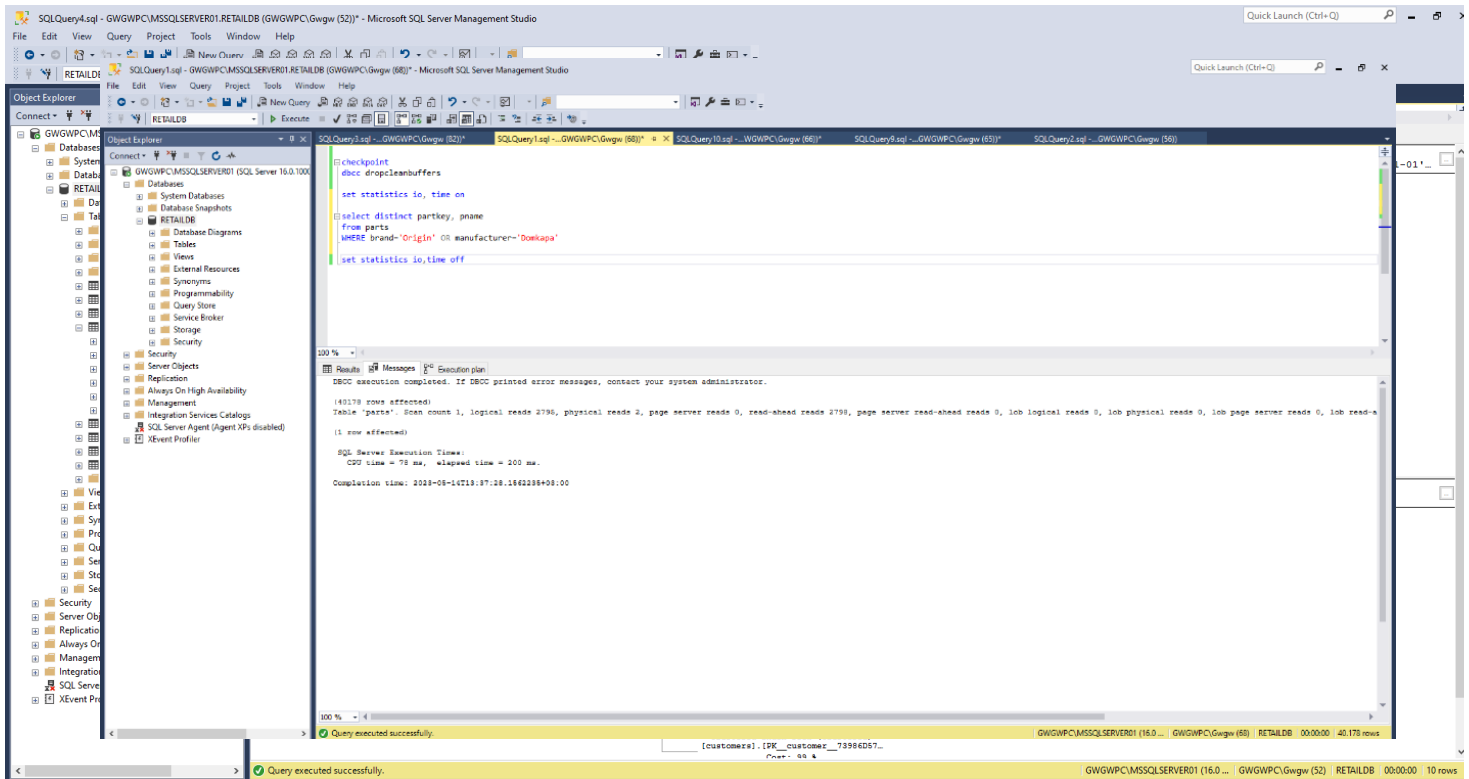
Συγκρίνοντας λοιπόν τα δύο ερωτήματα έχουμε:

The screenshot displays the SQL Server Enterprise Manager interface. The left pane shows the 'Object Explorer' with the 'Databases' folder expanded, showing the 'RETAILDB' database. The right pane shows the 'Query Editor' with the SQL query. Below the query, the 'Execution Plan' is visible, showing a full table scan for the 'orders' table. The 'Results' pane at the bottom shows the execution statistics, including CPU time and elapsed time.

Query execution statistics:

- SQL Server Execution Times: CPU time = 62 ms, elapsed time = 101 ms.
- DBCC execution completed. If DBCC printed error messages, contact your system administrator.
- SQL Server Execution Times: CPU time = 0 ms, elapsed time = 3 ms.
- DBCC execution completed. If DBCC printed error messages, contact your system administrator.
- SQL Server Execution Times: CPU time = 16 ms, elapsed time = 11 ms.

Query executed successfully. GWGWPC\MSSQLSERVER01 (16.0 ... GWGWPC\Gwggw (32) RETAILDB 00:00:00 10 rows



Παρατηρούμε λοιπόν ότι τα logical reads μειώθηκαν σε μεγάλο βαθμό, όπως και ότι το πλάνο εκτέλεσης έχει λιγότερα βήματα (παρόλο που το κόστος είναι 50-50)

Table 'customers'. Scan count 1, logical reads 433

Table 'orders'. Scan count 1, logical reads 629

Σε σύγκριση με το δοσμένο ερώτημα, μετά και τη χρήση των ευρετηρίων:

Table 'customers'. Scan count 11, logical reads 1718,

Table 'orders'. Scan count 11, logical reads 4152,

(Όπως φαίνεται και από την παραπάνω εικόνα)

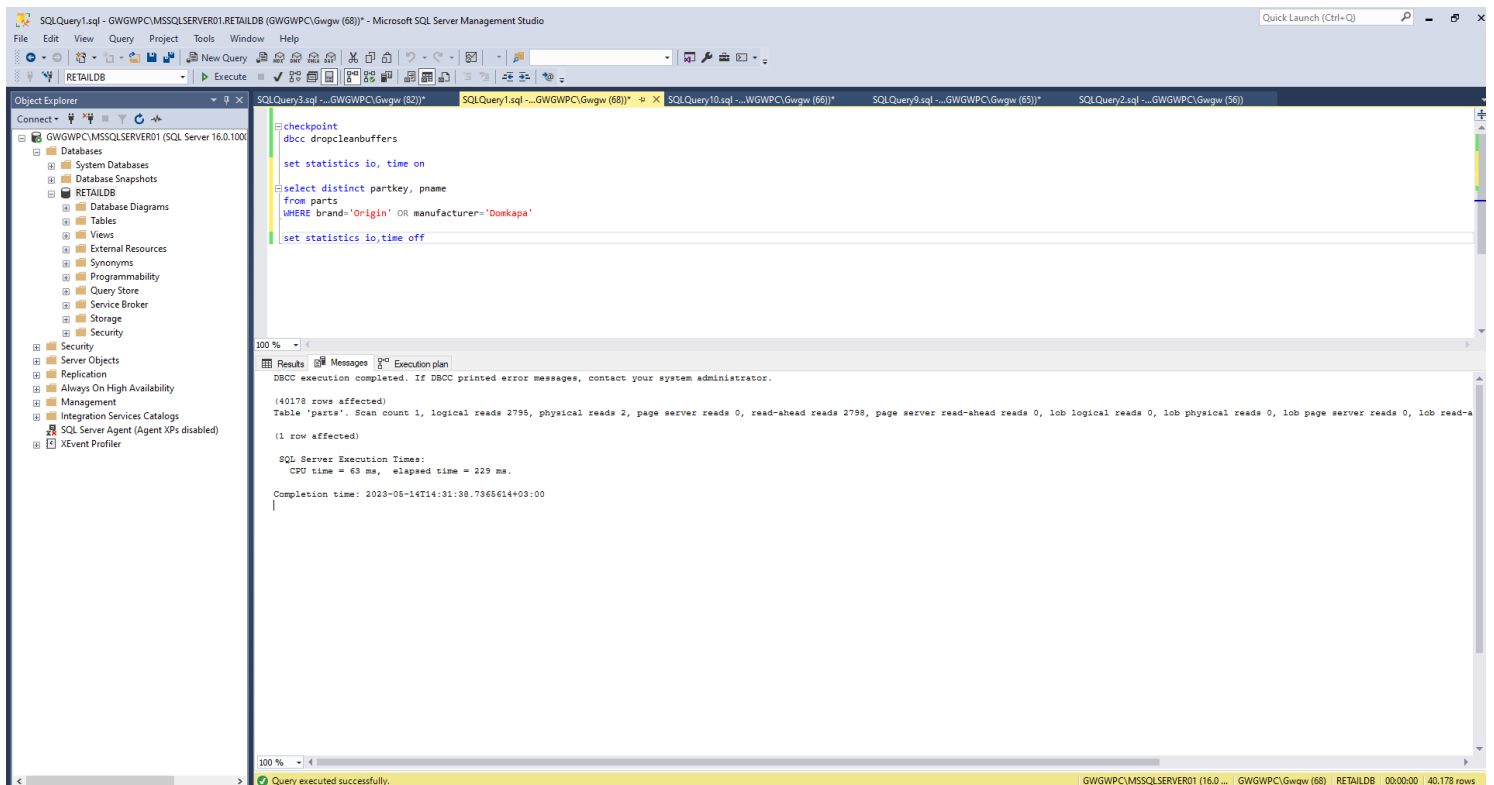
Ζήτηση 4^ο

A1) Για το πρώτο επερώτημα έχουμε:

✚ Πριν την δημιουργία των ευρετηρίων:

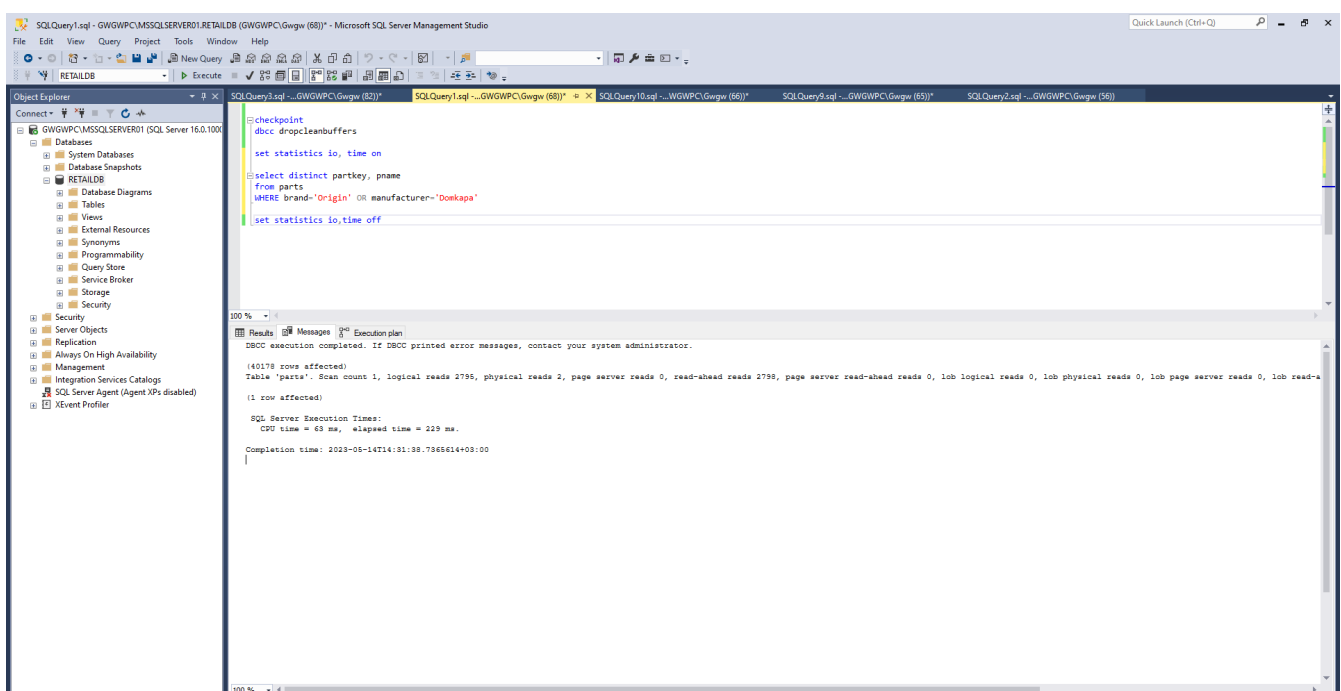
Παρατηρούμε ότι συνολικά έχουμε:

Table 'parts'. Scan count 1, logical reads 2795.



✚ Δημιουργία idx1 (only)

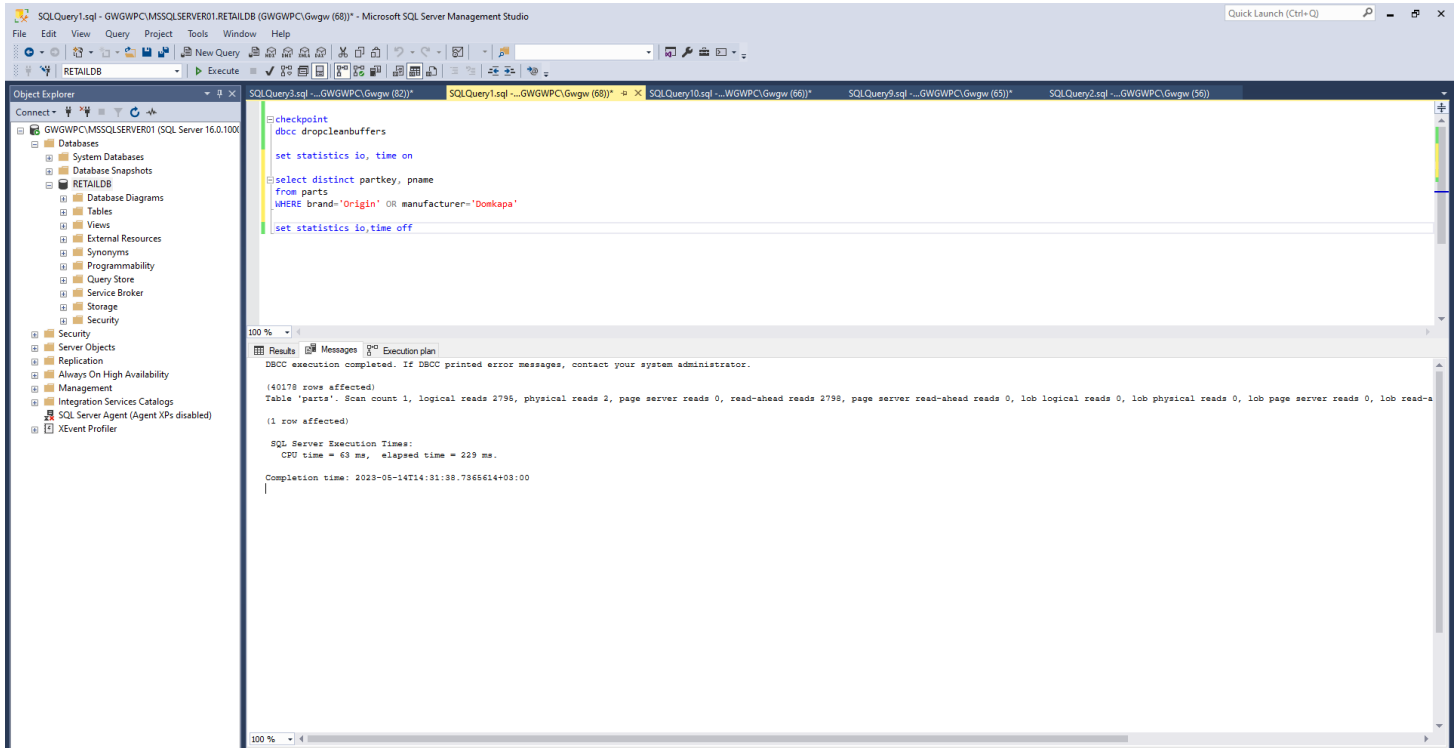
Εδώ παρατηρούμε ότι εφόσον υπάρχει τελεστής OR στο επερώτημα, παρότι μπορεί να ελέγξει ως προς αυτό το ένα ευρετήριο δεν συμφέρει γιατί δεν ελέγχει ως προς το άλλο. Οπότε είναι πιο αποδοτικό να κάνει ένα scan σε όλο μαζί τον πίνακα. Αυτό επιβεβαιώνεται και με την εκτέλεση του επερωτήματος εφόσον δημιουργήσαμε το idx1. (Τα logical reads είναι πάλι 2795)





🚧 Δημιουργία idx2 (only)

Το ίδιο ισχύει και για το δεύτερο ευρετήριο. Δηλαδή παρατηρούμε ότι εφόσον υπάρχει τελεστής OR στο επερώτημα, παρότι μπορεί να ελέγξει ως προς αυτό το ευρετήριο δεν συμφέρει γιατί δεν ελέγχει ως προς το άλλο. Οπότε είναι πιο αποδοτικό να κάνει ένα scan σε όλο μαζί. Αυτό επιβεβαιώνεται και με την εκτέλεση του επερωτήματος εφόσον δημιουργήσαμε το idx2. (Τα logical reads είναι πάλι 2795)



- Δημιουργία idx1, idx2.

Εδώ παρατηρούμε ότι εφόσον έχουμε ευρετήρια και στα μέρη του OR, αυτά θα χρησιμοποιηθούν για την εκτέλεση του επερωτήματος. Αυτό επιβεβαιώνεται και από τα logical reads που εκτελούνται μετά τη δημιουργία των ευρετηρίων.



(Τα logical reads είναι 346)

SQLQuery1.sql - GWGWPC\MSSQLSERVER01\RETAILDB (GWGWPC\Gwgw (68)) - Microsoft SQL Server Management Studio

Object Explorer: GWGWPC\MSSQLSERVER01 (SQL Server 16.0.100) > Databases > RETAILDB > Tables

SQLQuery1.sql - GWGWPC\Gwgw (68):

```
checkpoint
dbcc dropcleanbuffers
set statistics io, time on
select distinct partkey, pname
from parts
where brand='Origin' or manufacturer='Dunkap'
set statistics io,time off
```

Results: DBCC execution completed. If DBCC printed error messages, contact your system administrator. SQL Server parse and compile time: CPU time = 7 ms, elapsed time = 7 ms. (40178 rows affected) Table 'parts'. Scan count 2, logical reads 346, physical reads 6, page server reads 0, read-ahead reads 366, page server read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob page server reads 0, lob read-ahead reads 0. (1 row affected) SQL Server Execution Times: CPU time = 62 ms, elapsed time = 268 ms. Completion time: 2023-05-14T14:40:59.7296817+03:00

Query executed successfully. GWGWPC\MSSQLSERVER01 (16.0.100) GWGWPC\Gwgw (68) RETAILDB 00:00:00 40,178 rows

A2) Για το δεύτερο ερωτήματα έχουμε:

- ✚ Πριν την δημιουργία των ευρετηρίων:
Παρατηρούμε ότι συνολικά έχουμε
Table 'parts'. Scan count 1, logical reads **5590**.



```

SQLQuery3.sql - GWGWPC\MSSQLSERVER01\RETAILDB (GWGWPC\Gwggw (66)) - Microsoft SQL Server Management Studio
File Edit View Query Project Tools Window Help
Object Explorer
Connect - GWGWPC\MSSQLSERVER01 (SQL Server 16.0.1000)
  Databases
    System Databases
    Database Snapshots
    RETAILDB
      Database Diagrams
      Tables
      Views
      External Resources
      Synonyms
      Programmability
      Query Store
      Service Broker
      Storage
      Security
  Security
  Server Objects
  Replication
  Always On High Availability
  Management
  Integration Services Catalogs
  SQL Server Agent (Agent XPs disabled)
  XEvent Profiler
SQLQuery3.sql - GWGWPC\Gwggw (66)
SQLQuery10.sql - GWGWPC\Gwggw (69)
SQLQuery9.sql - GWGWPC\Gwggw (68)
SQLQuery2.sql - GWGWPC\Gwggw (67)
SQLQuery3.sql - GWGWPC\Gwggw (66)
  checkpoint
  dbcc dropcleanbuffers
  set statistics io, time on
  select partkey, pname
  from parts
  where brand='origin'
  intersect
  select partkey, pname
  from parts
  where manufacturer='Domkapa'
  set statistics io,time off
Results Messages Execution plan
DBCC execution completed. If DBCC printed error messages, contact your system administrator.
(8096 rows affected)
Table 'parts'. Scan count 2, logical reads 2556, physical reads 2, page server reads 0, read-ahead reads 2798, page server read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob page server reads 0, lob read-a
(1 row affected)
SQL Server Execution Times:
CPU time = 78 ms, elapsed time = 117 ms.
Completion time: 2023-05-14T18:54:36.9367537+03:00
100 %
Query executed successfully. GWGWPC\MSSQLSERVER01 (16.0.1000) GWGWPC\Gwggw (66) RETAILDB 00:00:00 8,096 rows

```

✚ Δημιουργία idx1 (only)

Εδώ παρατηρούμε ότι υπάρχει INTERSECT στο επερωτήμα, οπότε παρόλο που έχουμε ένα index, είναι πιο αποδοτικό να ψάξει εκεί και αν η συνθήκη δεν αληθεύει απλώς συνεχίζει να ψάχνει.

Αυτό επιβεβαιώνεται και με την εκτέλεση του επερωτήματος εφόσον δημιουργήσαμε το idx1. (Τα logical reads έγιναν 2856)

```

SQLQuery3.sql - GWGWPC\MSSQLSERVER01\RETAILDB (GWGWPC\Gwggw (66)) - Microsoft SQL Server Management Studio
File Edit View Query Project Tools Window Help
Object Explorer
Connect - GWGWPC\MSSQLSERVER01 (SQL Server 16.0.1000)
  Databases
    System Databases
    Database Snapshots
    RETAILDB
      Database Diagrams
      Tables
      Views
      External Resources
      Synonyms
      Programmability
      Query Store
      Service Broker
      Storage
      Security
  Security
  Server Objects
  Replication
  Always On High Availability
  Management
  Integration Services Catalogs
  SQL Server Agent (Agent XPs disabled)
  XEvent Profiler
SQLQuery3.sql - GWGWPC\Gwggw (66)
SQLQuery10.sql - GWGWPC\Gwggw (69)
SQLQuery9.sql - GWGWPC\Gwggw (68)
SQLQuery2.sql - GWGWPC\Gwggw (67)
SQLQuery3.sql - GWGWPC\Gwggw (66)
  checkpoint
  dbcc dropcleanbuffers
  set statistics io, time on
  select partkey, pname
  from parts
  where brand='origin'
  intersect
  select partkey, pname
  from parts
  where manufacturer='Domkapa'
  set statistics io,time off
Results Messages Execution plan
DBCC execution completed. If DBCC printed error messages, contact your system administrator.
(8096 rows affected)
Table 'parts'. Scan count 2, logical reads 2856, physical reads 5, page server reads 0, read-ahead reads 2870, page server read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob page server reads 0, lob read-a
(1 row affected)
SQL Server Execution Times:
CPU time = 47 ms, elapsed time = 86 ms.
Completion time: 2023-05-14T18:57:26.6922774+03:00
100 %
Query executed successfully. GWGWPC\MSSQLSERVER01 (16.0.1000) GWGWPC\Gwggw (66) RETAILDB 00:00:00 8,096 rows

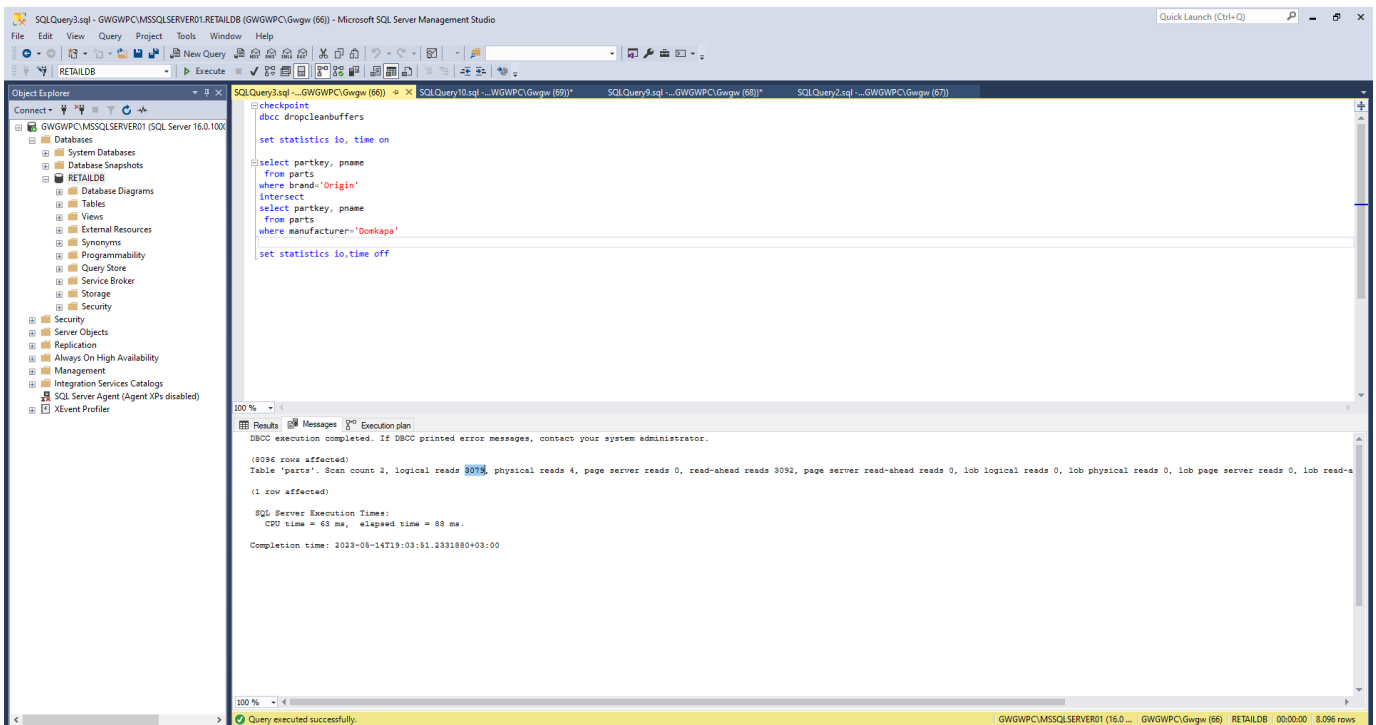
```



✚ Δημιουργία idx2 (only)

Κι εδώ παρατηρούμε ότι υπάρχει INTERSECT στο ερωτήμα, οπότε παρόλο που έχουμε ένα index είναι πιο αποδοτικό να ψάξει εκεί και αν η συνθήκη δεν αληθεύει, απλώς συνεχίζει να ψάχνει.

Αυτό επιβεβαιώνεται και με την εκτέλεση του ερωτήματος εφόσον δημιουργήσαμε το idx2. (Τα logical reads έγιναν 3079)

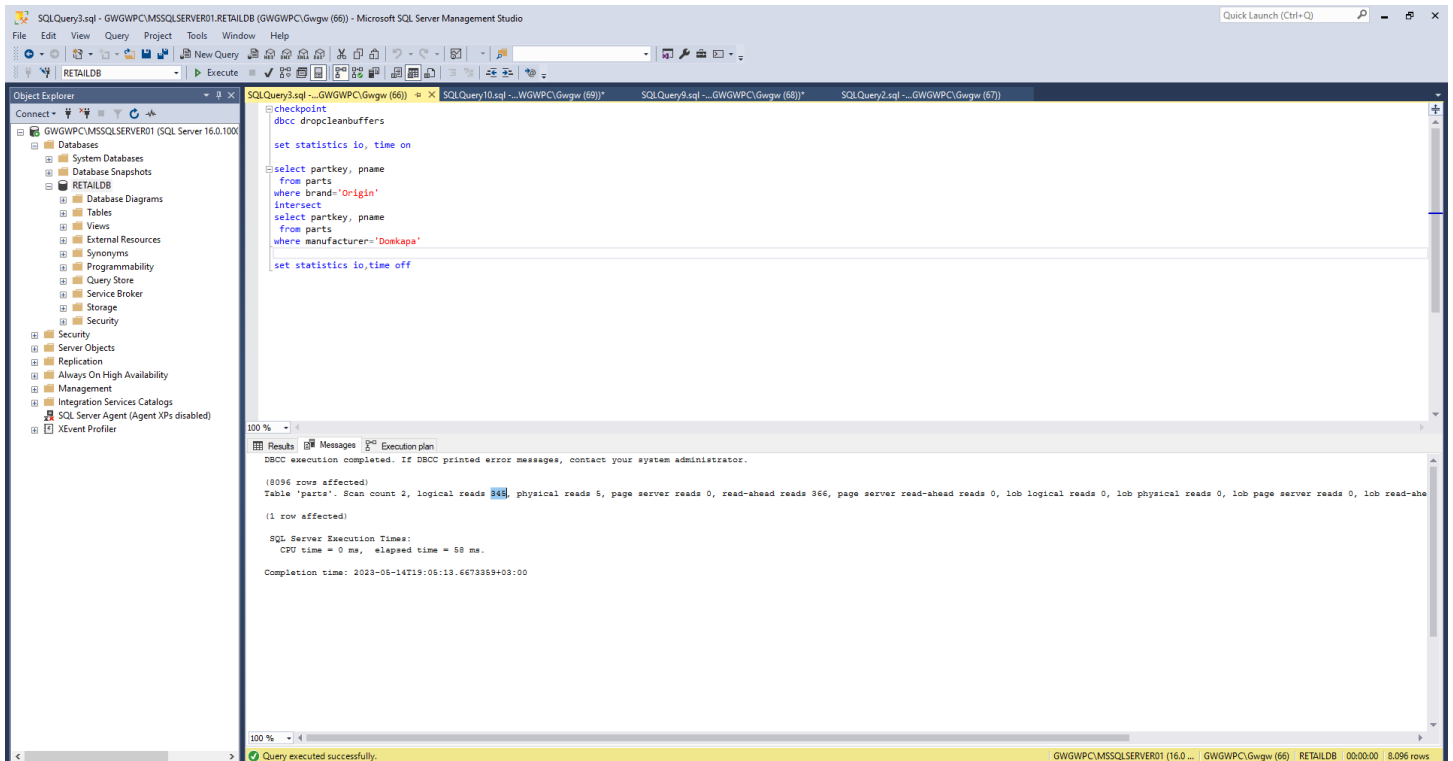


✚ Δημιουργία idx1, idx2.

Εδώ ο αριθμός των logical reads μειώνεται περισσότερο εφόσον υπάρχουν indexes και στα 2 σκέλη του intersect (brand, manufacturer).

Αυτό επιβεβαιώνεται και από τα logical reads που εκτελούνται μετά τη δημιουργία των ευρετηρίων.

(Τα logical reads είναι 345)



B1) Για το πρώτο ερωτήματα έχουμε:

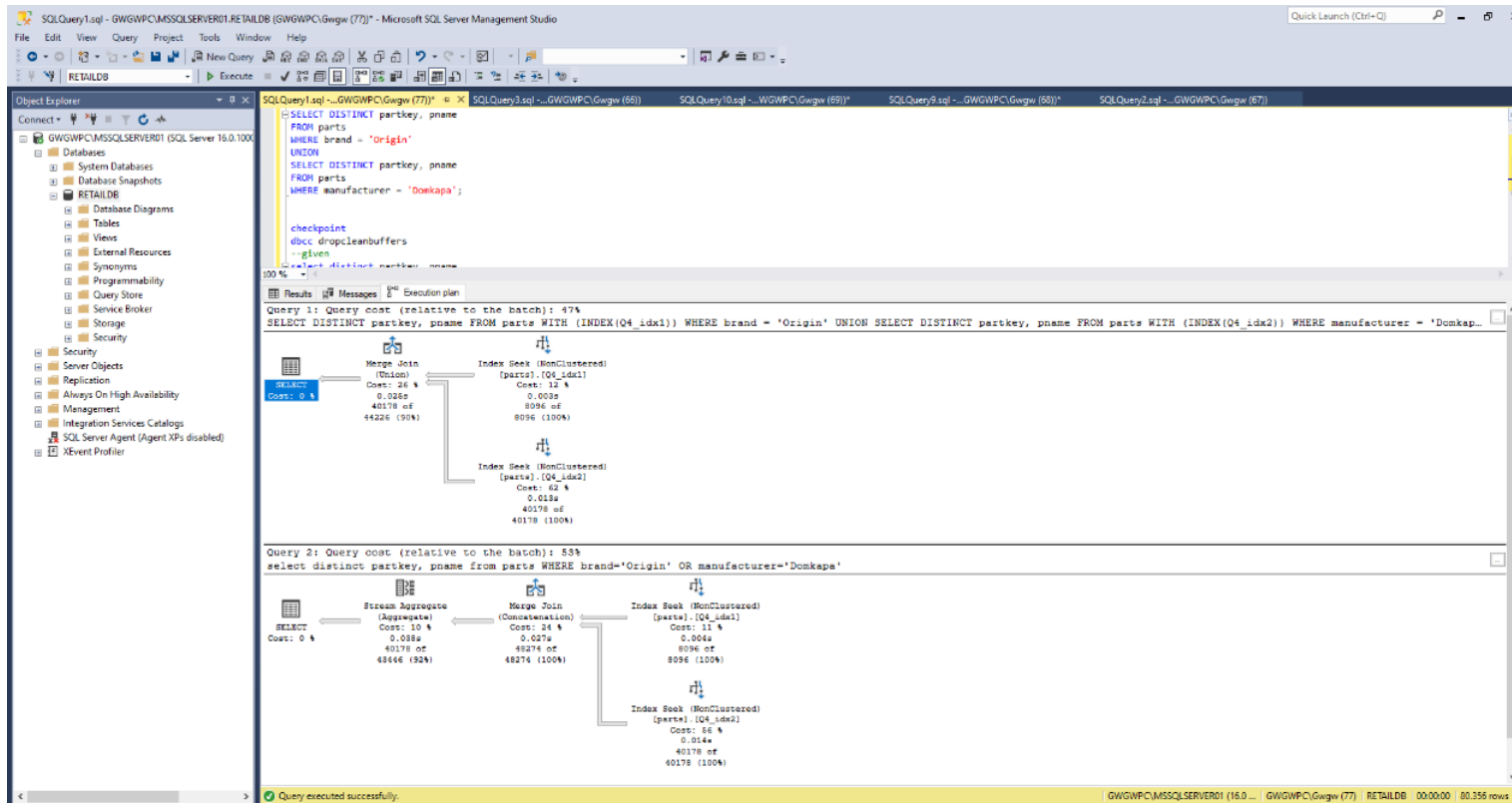
Τα indexes που υπάρχουν δεν θεωρούμε ότι χρειάζονται βελτίωση, ούτε ότι μπορεί να χρησιμοποιηθεί κάποιο άλλο εφόσον από αυτά παίρνουμε τις πληροφορίες που χρειαζόμαστε για να βελτιστοποιήσουμε τα logical reads. Ωστόσο, το ερωτήματα μπορεί να βελτιωθεί ως εξής:

```
SELECT DISTINCT partkey, pname
FROM parts
WHERE brand = 'Origin'
UNION
SELECT DISTINCT partkey, pname
FROM parts
WHERE manufacturer = 'Domkapa';
```

Με αυτό το ερωτήματα χρησιμοποιούμε δύο ξεχωριστά SELECT τα οποία συνενώνουμε με τον τελεστή UNION (προκειμένου να έχουμε το ίδιο αποτέλεσμα με τον τελεστή OR). Κάθε SELECT φιλτράρει τα rows με βάση την εκάστοτε συνθήκη. Αυτό επιτρέπει στον optimizer να χρησιμοποιήσει τα αντίστοιχα indexes πιο αποτελεσματικά, καθώς κάθε τμήμα SELECT-FROM-WHERE clause του UNION λειτουργεί μόνο σε ένα υποσύνολο δεδομένων.

Σε σύγκριση με το δοσμένο ερωτήματα, παρόλο που και αυτό κάνει χρήση των indexes, δεν είναι το ίδιο αποτελεσματικό με το προτεινόμενο επειδή συνδυάζει και τις δύο συνθήκες σε ένα μόνο βήμα (το OR). Ο optimizer ενδέχεται να μην μπορεί να αξιοποιήσει πλήρως τα μεμονωμένα ευρετήρια, με αποτέλεσμα δυνητικά λιγότερο καλά σχέδια εκτέλεσης.

Αυτό επιβεβαιώνεται και με τα αντίστοιχα κόστη που έχουμε, και είναι τα εξής (τα logical reads παραμένουν ίδια):



Παρατηρούμε λοιπόν ότι συγκρίνοντας τα 2 κόστη των ερωτήσεων, το άνωθεν προτεινόμενο καταλαμβάνει το 47% του συνολικού κόστους και το δοσμένο από την εκφώνηση 53%. (Άρα είναι προτιμότερο το άνωθεν προτεινόμενο).

B2) Για το δεύτερο ερωτήμα έχουμε:

Αρχικά, εφόσον έχουμε intersect, δηλαδή τομή 2 συνθηκών είναι προτιμότερο να γραφεί το ερωτήμα ως:

```
SELECT DISTINCT partkey, pname
FROM parts
WHERE brand = 'Origin' AND manufacturer = 'Domkapa'
```

Και να δημιουργηθεί ένα μόνο index

```
CREATE INDEX index_on_parts ON parts (manufacturer, brand, partkey) INCLUDE
(pname)
```

Έτσι, αρχικά δημιουργούμε μόνο ένα ευρετήριο (και όχι 2 όπως προτείνεται από την εκφώνηση), το οποίο ταξινομείται κατά (manufacturer, brand, partkey). Επομένως, η βάση μπορεί να πραγματοποιήσει index seek στο σχετικό τμήμα του ευρετηρίου και να ανακτήσει γρήγορα μόνο τα απαραίτητα rows.

Επιπλέον η στήλη pname περιλαμβάνεται στο index (με το include). Η βάση λοιπόν μπορεί να ανακτήσει όλα τα δεδομένα που χρειάζεται για την εκτέλεση του ερωτήματος απευθείας από το ευρετήριο χωρίς να χρειάζεται πρόσθετες αναζητήσεις.



Επίσης δεν χρειάζεται να εκτελεί πρόσθετες λειτουργίες, όπως να υπολογίζει πρώτα το όσα brands είναι 'Origin', έπειτα όσους manufacturers είναι 'Domkapa' και έπειτα να κάνει intersect τα 2 υποσύνολα

Με αυτές τις βελτιστοποιήσεις λοιπόν παρατηρούμε ότι η διαφορά στα κόστη των 2 ερωτημάτων είναι 2%-98% (αρκετά υψηλή διαφορά!) και ότι τα logical reads με το προτεινόμενο ερώτημα είναι 70 ενώ με το δωσμένο 2021.

```
SQLQuery1.sql - GWGWPC\MSSQLSERVER01.RETAILDB (GWGWPC\Gwggw (77)) - Microsoft SQL Server Management Studio
File Edit View Query Project Tools Window Help
New Query Execute
Object Explorer
Connect
GWGWPC\MSSQLSERVER01 (SQL Server 16.0.100)
Databases
System Databases
Database Snapshots
RETAILDB
Database Diagrams
Tables
Views
External Resources
Synonyms
Programmability
Query Store
Service Broker
Storage
Security
Server Objects
Replication
Always On High Availability
Management
Integration Services Catalogs
SQL Server Agent (Agent XPs disabled)
XEvent Profiler
SQLQuery1.sql - GWGWPC\Gwggw (77)
SQLQuery3.sql - GWGWPC\Gwggw (66)
SQLQuery10.sql - GWGWPC\Gwggw (69)
SQLQuery9.sql - GWGWPC\Gwggw (68)
SQLQuery2.sql - GWGWPC\Gwggw (67)
--checkpoint
dbcc dropcleanbuffers
set statistics io, time on
--line
SELECT DISTINCT partkey, pname
FROM parts
WHERE brand = 'Origin' AND manufacturer = 'Domkapa'
checkpoint
dbcc dropcleanbuffers
--given
select partkey, pname
from parts
where brand='Origin'
intersect
select partkey, pname
from parts
Results
Messages
Execution plan
DBCC execution completed. If DBCC printed error messages, contact your system administrator.
(8096 rows affected)
Table 'parts'. Scan count 1, logical reads 70, physical reads 3, page server reads 0, read-ahead reads 80, page server read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob page server reads 0, lob read-ahead
(1 row affected)
SQL Server Execution Times:
CPU time = 16 ms, elapsed time = 75 ms.
SQL Server Execution Times:
CPU time = 0 ms, elapsed time = 4 ms.
DBCC execution completed. If DBCC printed error messages, contact your system administrator.
SQL Server Execution Times:
CPU time = 16 ms, elapsed time = 9 ms.
(8096 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 logical reads 0, lob physical reads 0, lob page server reads 0, lob read-ahead
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 logical reads 0, lob physical reads 0, lob page server reads 0, lob read-ahead
Table 'parts'. Scan count 2, logical reads 2021, physical reads 1, page server reads 0, read-ahead reads 1713, page server read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob page server reads 0, lob read-a
(1 row affected)
SQL Server Execution Times:
CPU time = 62 ms, elapsed time = 187 ms.
Completion time: 2023-05-18T01:56:14.0717708+03:00
100 %
Query executed successfully.
GWGWPC\MSSQLSERVER01 (16.0.100) - GWGWPC\Gwggw (77) - RETAILDB 00:00:00 16.192 rows
```



SQLQuery1.sql - GWGWPC\MSSQLSERVER01.RETAILDB (GWGWPC\Gwgw (77)) - Microsoft SQL Server Management Studio

Object Explorer: GWGWPC\MSSQLSERVER01 (SQL Server 16.0.1000)

Query: `SELECT DISTINCT partkey, pname FROM parts WHERE brand = 'Origin' AND manufacturer = 'Domkapa'`

Execution plan: Query 1: Query cost (relative to the batch): 2%
SELECT DISTINCT partkey, pname FROM parts WHERE brand = 'Origin' AND manufacturer = 'Domkapa'

Index Seek (NonClustered) (parts).index_on_parts
Cost: 100 %
0.004s
8096 of 8000 (101%)

Query 2: Query cost (relative to the batch): 98%
select partkey, pname from parts where brand='Origin' intersect select partkey, pname from parts where manufacturer='Domkapa'

Missing Index (Impact 61.728): CREATE NONCLUSTERED INDEX [<Name of Missing Index, sysname>] ON [dbo].[parts] (([brand])) INCLUDE (([pname]))

Hash Match (Inner Join)
Cost: 28 %
0.060s
8096 of 1650 (490%)

Index Scan (NonClustered) (parts).index_on_parts
Cost: 60 %
0.026s
8096 of 8215 (98%)

Index Seek (NonClustered) (parts).index_on_parts
Cost: 12 %
0.005s

Query executed successfully.

Σε αυτό το σημείο να αναφερθεί ότι ο συνδυασμός επερωτήματος και index που προτείνεται είναι προτιμότερος και από τον συνδυασμό του επερωτήματος και των indexes που δίνονται από την εκφώνηση εφόσον χρησιμοποιείται ένα ευρετήριο (αντί για 2) και τα logical reads είναι 70 (αντί για 345 όπως είδαμε στο A2) .

Ζήτηση 5°

1° επερώτημα)

Θέλουμε να βρούμε τα συνολικά έσοδα από τις πωλήσεις της επιχείρησης σε κάθε region τα τελευταία 3 χρόνια.

Μία τέτοια πληροφορία πιθανώς μας βοηθάει να επανεξετάσουμε το αν θέλουμε να παραμείνουμε στο εκάστοτε region ή όχι (ανάλογα με το αν από χρόνο σε χρόνο έχουμε περισσότερα ή λιγότερα κέρδη για παράδειγμα).

Το επερώτημα λοιπόν που δημιουργούμε είναι το εξής:

```
SELECT r.region, YEAR(o.orderdate) AS order_year, SUM(l.quantity * l.price * (1 - l.discount)) AS total_revenue
FROM regions r
JOIN nations n ON r.regionkey = n.regionkey
JOIN customers c ON n.nationkey = c.nationkey
JOIN orders o ON c.custkey = o.custkey
JOIN lineitem l ON o.orderkey = l.orderkey
WHERE o.orderdate >= '1996-01-01' AND o.orderdate < '1999-01-01'
GROUP BY r.region, YEAR(o.orderdate)
ORDER BY r.region, YEAR(o.orderdate);
```



1^ο επερώτημα, indexes)

Τα indexes που δημιουργούμε είναι τα εξής:

```
create INDEX index_on_orders ON orders (orderidate) INCLUDE (custkey);

create INDEX index_on_customers1 ON customers (nationkey) INCLUDE (custkey);
create INDEX index_on_customers2 ON customers (custkey) INCLUDE (cname);

create INDEX index_on_lineitem ON lineitem (orderidate) INCLUDE (discount, price,
quantity);
```

Πριν από την δημιουργία των indexes έχουμε

The screenshot displays the SQL Server Enterprise Manager interface. The left pane shows the Object Explorer with the RETAILDB database selected. The central pane shows a query window with the following SQL code:

```
checkpoint dbcc dropcleanbuffers
set statistics io, time on

SELECT r.region, YEAR(o.orderdate) AS order_year, SUM(l.quantity * l.price * (1 - l.discount)) AS total_revenue
FROM regions r
JOIN nations n ON r.regionkey = n.regionkey
JOIN customers c ON n.nationkey = c.nationkey
JOIN orders o ON c.custkey = o.custkey
JOIN lineitem l ON o.orderkey = l.orderkey
WHERE o.orderdate >= '1996-01-01' AND o.orderdate < '1999-01-01'
GROUP BY r.region, YEAR(o.orderdate)
ORDER BY r.region, YEAR(o.orderdate);

set statistics io, time off
```

The bottom pane shows the execution plan and the results of the query. The results table has the following data:

region	order_year	total_revenue
AFRICA	1996	267325752536.2098
AFRICA	1997	265486051036.1354
AFRICA	1998	154778406592.7651
AMERICA	1996	226967623771.2402
AMERICA	1997	225494800525.1854
AMERICA	1998	129020496996.8673
ASIA	1996	224326030776.8875
ASIA	1997	226342284368.8656
ASIA	1998	131112036631.3336
EUROPE	1996	219805887651.8439
EUROPE	1997	220337902851.6299
EUROPE	1998	128466081249.8513
MIDDLE EAST	1996	180255038856.7313
MIDDLE EAST	1997	177696141628.9401
MIDDLE EAST	1998	105197472463.2891

The status bar at the bottom indicates: Query executed successfully. GWGWPC\MSSQLSERVER01 (16.0 ... | GWGWPC\Gwggw (78) | RETAILDB 00:00:01 15 rows



SQLQuery4.sql - GWGWPC\MSSQLSERVER01\RETAILDB (GWGWPC\Gwggw (78)) - Microsoft SQL Server Management Studio

Object Explorer: GWGWPC\MSSQLSERVER01 (SQL Server 16.0.1000)

Query: `checkpoint dbcc dropcleanbuffers
set statistics io, time on
SELECT r.region, YEAR(o.orderdate) AS order_year, SUM(1.quantity * 1.price * (1 - 1.discount)) AS total_revenue
FROM regions r
JOIN nations n ON r.regionkey = n.regionkey
JOIN customers c ON n.nationkey = c.nationkey
JOIN orders o ON c.custkey = o.custkey
JOIN lineitem l ON o.orderkey = l.orderkey
WHERE o.orderdate >= '1990-01-01' AND o.orderdate < '1999-01-01'
GROUP BY r.region, YEAR(o.orderdate)
ORDER BY r.region, YEAR(o.orderdate);
set statistics io, time off`

Results: (15 rows affected)

Table 'customers'. Scan count 11, logical reads 2685, physical reads 2, page server reads 0, read-ahead reads 2562, page server read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob page server reads 0, lob r

Table 'nations'. Scan count 11, logical reads 4, physical reads 1, page server reads 0, read-ahead reads 0, page server read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob page server reads 0, lob read-ah

Table 'regions'. Scan count 11, logical reads 4, physical reads 1, page server reads 0, read-ahead reads 0, page server read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob page server reads 0, lob read-ah

Table 'orders'. Scan count 11, logical reads 16961, physical reads 1, page server reads 0, read-ahead reads 16727, page server read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob page server reads 0, lob re

Table 'lineitem'. Scan count 11, logical reads 60387, physical reads 1, page server reads 0, read-ahead reads 59304, page server read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob page server reads 0, lob

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 logical reads 0, lob physical reads 0, lob page server reads 0, lob read-ah

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 logical reads 0, lob physical reads 0, lob page server reads 0, lob read-ah

SQL Server Execution Times:
CPU time = 3187 ms, elapsed time = 878 ms.
Completion time: 2023-06-15T15:57:38.4062078+03:00

Query executed successfully.

Συγκεκριμένα έχουμε

Table 'lineitem'. Scan count 11, logical reads 60387,

Table 'orders'. Scan count 11, logical reads 16961,

Table 'customers'. Scan count 11, logical reads 2685,

Table 'nations'. Scan count 11, logical reads 4,

Table 'regions'. Scan count 11, logical reads 4,

Ενώ μετά τη δημιουργία των indexes παρατηρούμε ότι έχουμε

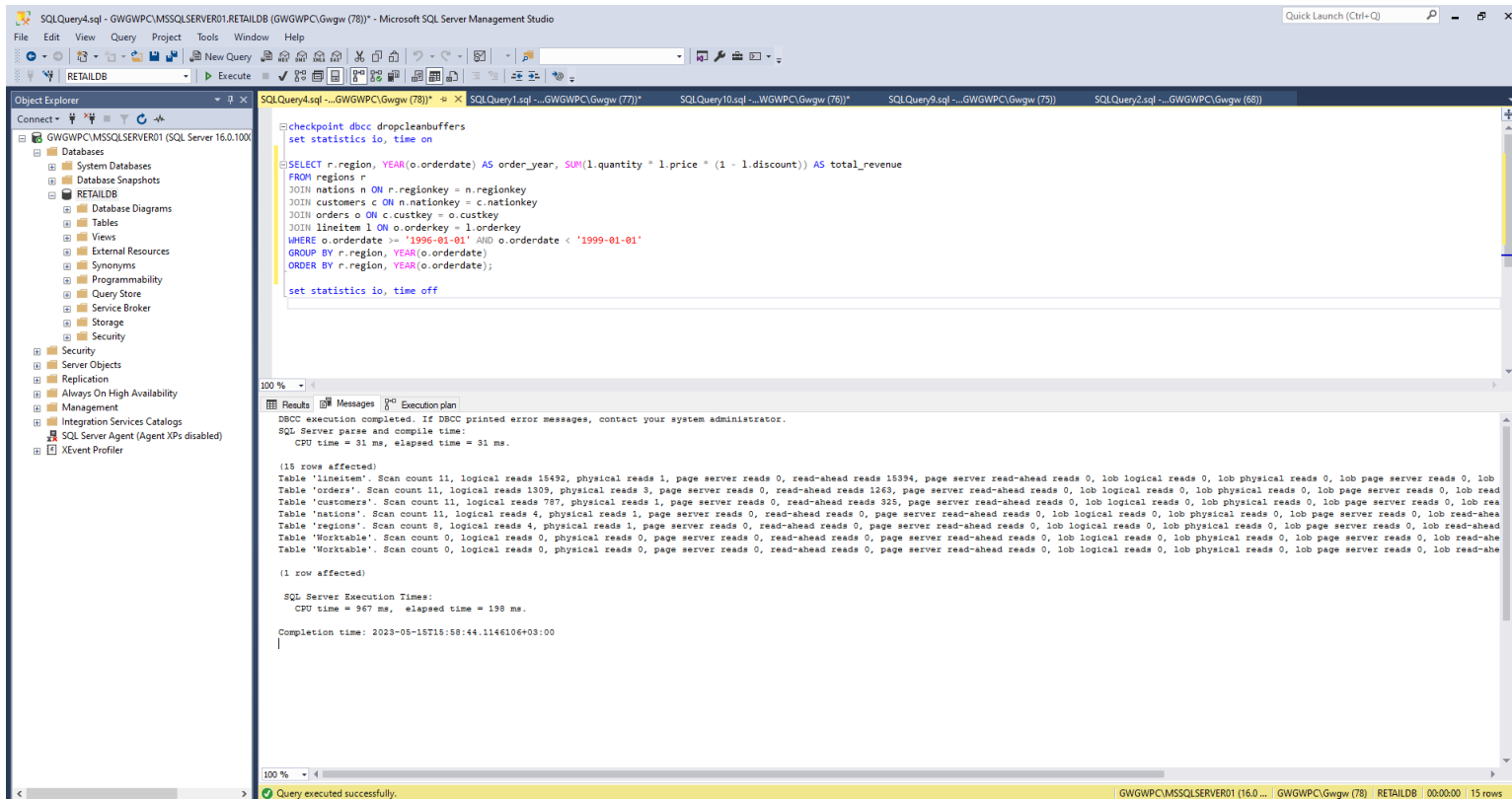
Table 'lineitem'. Scan count 11, logical reads 15492,

Table 'orders'. Scan count 11, logical reads 1309,

Table 'customers'. Scan count 11, logical reads 787

Table 'nations'. Scan count 11, logical reads 4

Table 'regions'. Scan count 8, logical reads 4



2^ο επερώτημα)

Ένα δεύτερο χρήσιμο ερώτημα θα μπορούσε να είναι το εξής:

Θέλουμε να βρούμε τα top 7 (ενδεικτικός αριθμός) έθνη (nations) με το μεγαλύτερο υπόλοιπο όλων των καταναλωτών μαζί που ανήκουν σε αυτό (μαζί με τον αριθμό των καταναλωτών που ανήκουν στο εκάστοτε έθνος).

Ένα τέτοιο ερώτημα θα μπορούσε να φανεί χρήσιμο εάν θέλουμε για παράδειγμα να δούμε ποια έθνη «χρωστάνε» περισσότερα χρήματα (και πόσα ακριβώς είναι αυτά) στην εταιρία ώστε να χρησιμοποιηθεί αργότερα για υπολογισμό ποσοστών για παράδειγμα ή οτιδήποτε άλλο φανεί χρήσιμο για την εταιρία.

Το ερώτημα λοιπόν που δημιουργούμε είναι το εξής:

```
SELECT n.nation, COUNT(c.custkey) as number_of_customers, SUM(c.c_acctbal) AS
total_account_balance_of_customers
FROM customers c
JOIN nations n ON c.nationkey = n.nationkey
WHERE n.nation IN (
    SELECT TOP 7 n.nation
    FROM customers c
    JOIN nations n ON c.nationkey = n.nationkey
    GROUP BY n.nation
    ORDER BY SUM(c.c_acctbal) DESC
)
GROUP BY n.nation
ORDER BY total_account_balance_of_customers DESC;
```



2^ο επερώτημα, indexes)

Το index που μπορούμε να δημιουργήσουμε για την βελτιστοποίηση αυτού του επερωτήματος είναι το εξής:

```
CREATE INDEX index_on_customers ON customers (nationkey, c_acctbal) INCLUDE (custkey);
```

Ενισχύοντας την απόδοση για το join αλλά και το φιλτράρισμα του επερωτήματος.

Πριν από τη δημιουργία των indexes λοιπόν έχουμε τα εξής αποτελέσματα:

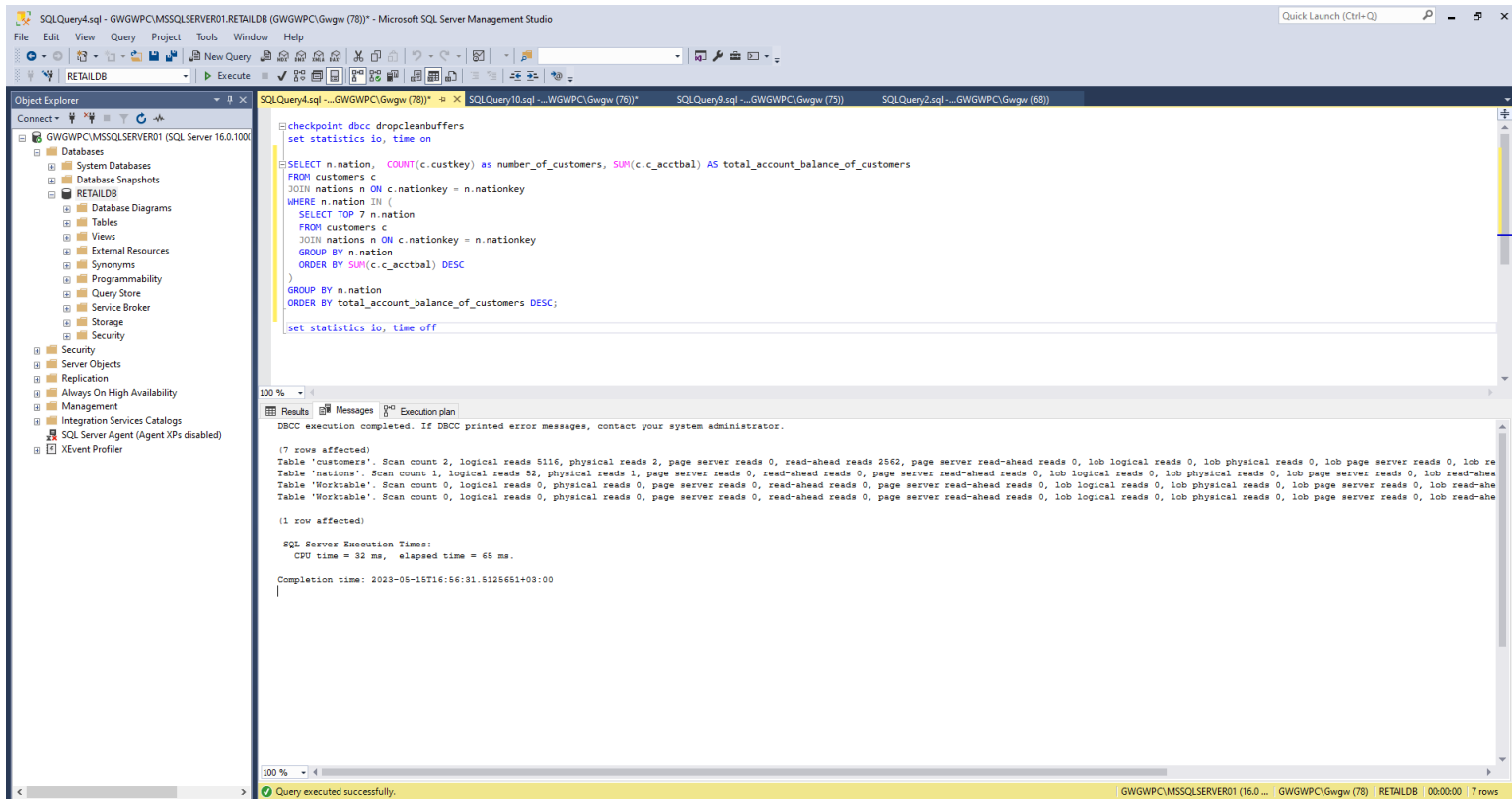
The screenshot displays the SQL Server Management Studio interface. The left pane shows the Object Explorer with the RETAILDB database selected. The central pane shows the SQL query editor with the following query:

```
checkpoint dbcc dropcleanbuffers
set statistics io, time on
SELECT n.nation, COUNT(c.custkey) AS number_of_customers, SUM(c.c_acctbal) AS total_account_balance_of_customers
FROM customers c
JOIN nations n ON c.nationkey = n.nationkey
WHERE n.nation IN (
  SELECT TOP 7 n.nation
  FROM customers c
  JOIN nations n ON c.nationkey = n.nationkey
  GROUP BY n.nation
  ORDER BY SUM(c.c_acctbal) DESC
)
GROUP BY n.nation
ORDER BY total_account_balance_of_customers DESC;
set statistics io, time off
```

The bottom pane shows the Results tab with the following data:

nation	number_of_customers	total_account_balance_of_customers
IRAN	6158	27714730.26
CHINA	6138	27682524.25
JORDAN	6001	27316534.53
INDIA	6083	27265231.00
EGYPT	6031	27225792.44
ALGERIA	5997	27155320.05
MOZAMBIQUE	6023	27154414.47

The status bar at the bottom indicates "Query executed successfully." and "7 rows".



Πιο συγκεκριμένα:

Table 'nations'. Scan count 1, logical reads 52

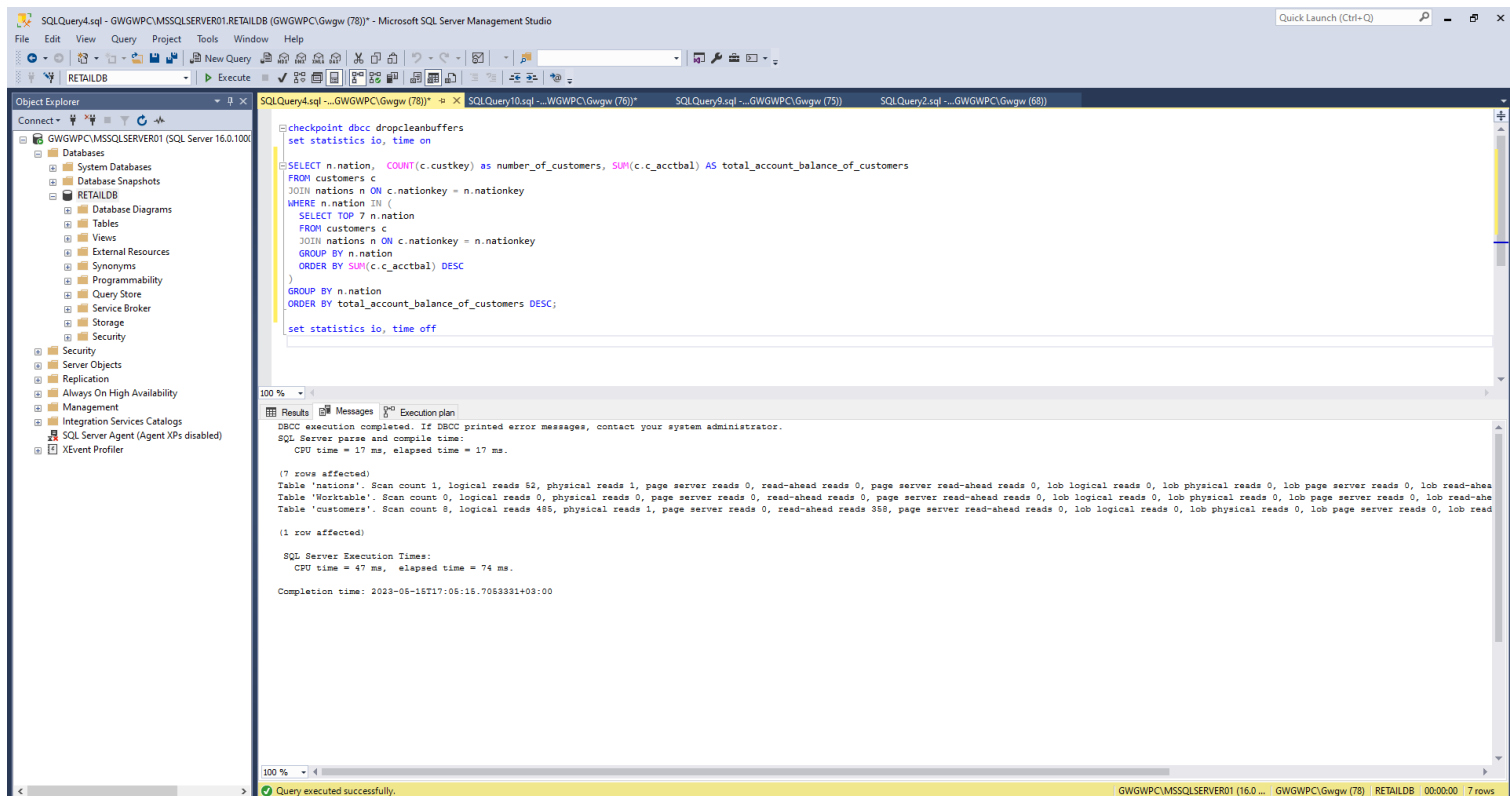
Table 'customers'. Scan count 8, logical reads 5116

Ενώ μετά τη δημιουργία του index έχουμε:

Table 'nations'. Scan count 1, logical reads 52

Table 'customers'. Scan count 8, logical reads 485.

(όπως φαίνεται και στη παρακάτω εικόνα)



Σημειώσεις που ισχύουν για όλα τα ζητήματα:

- Επειδή ο πίνακας regions έχει 5 εγγραφές και ο πίνακας nations 25 θεωρούμε ότι δεν χρειάζεται η δημιουργία Indexes για τη βελτιστοποίηση των joins στα οποία αυτοί οι πίνακες χρησιμοποιούνται.
- Σε όλα τα ερωτήματα τα ευρετήρια που χρησιμοποιήθηκαν για προηγούμενα ερωτήματα γίνονται drop πριν από την εκτέλεση των καιούριων επερωτημάτων και κάθε φορά το επιβεβαιώνω με την εντολή `select * from sys.indexes` (τα default indexes που ήταν δημιουργημένα από το σύστημα ήταν 194, οπότε κάθε φορά επιβεβαιώνω ότι έχουν διαγραφεί όλα με το αν ο αριθμός των indexes παραμένει 194 ή όχι)
- Σε όλα τα ερωτήματα γίνεται η χρήση του `checkpoint dbcc dropcleanbuffers`