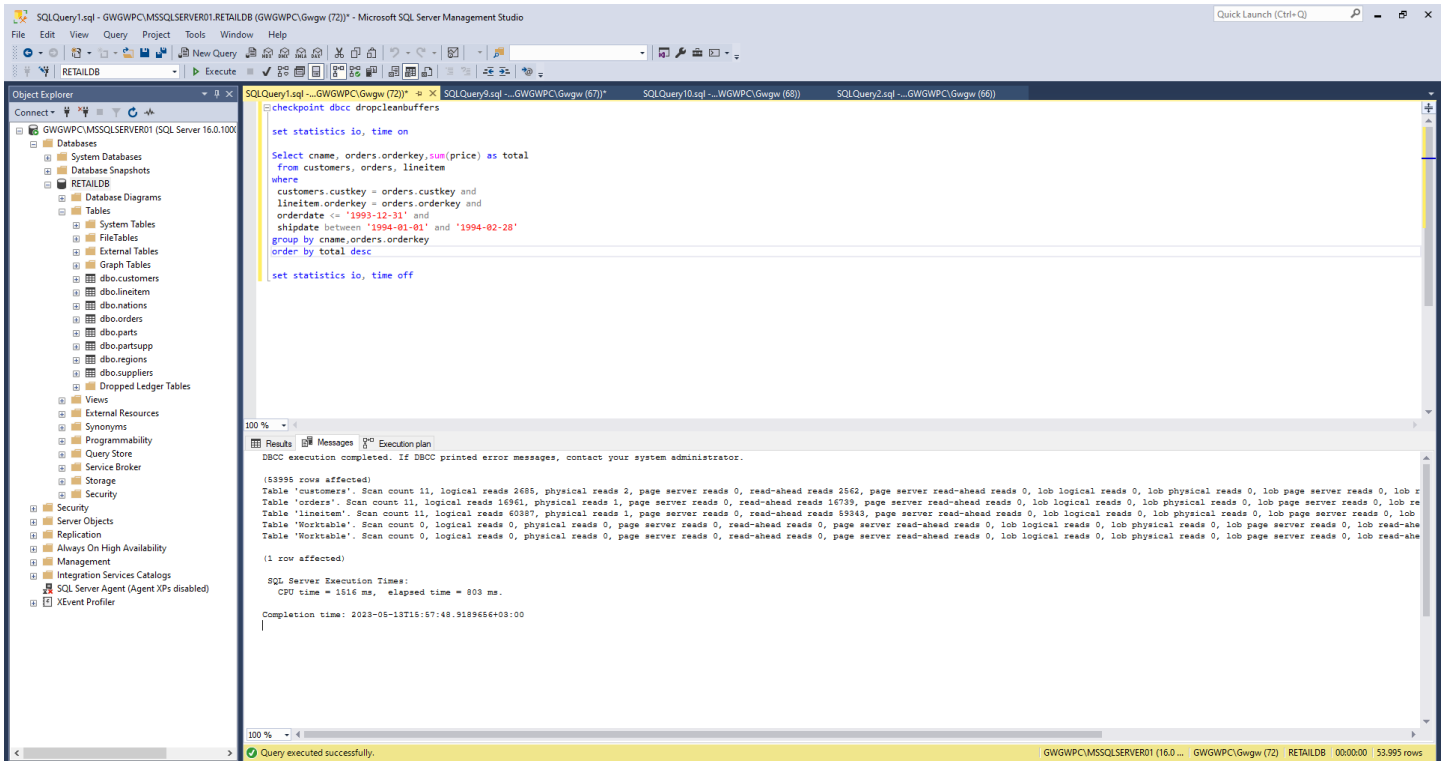
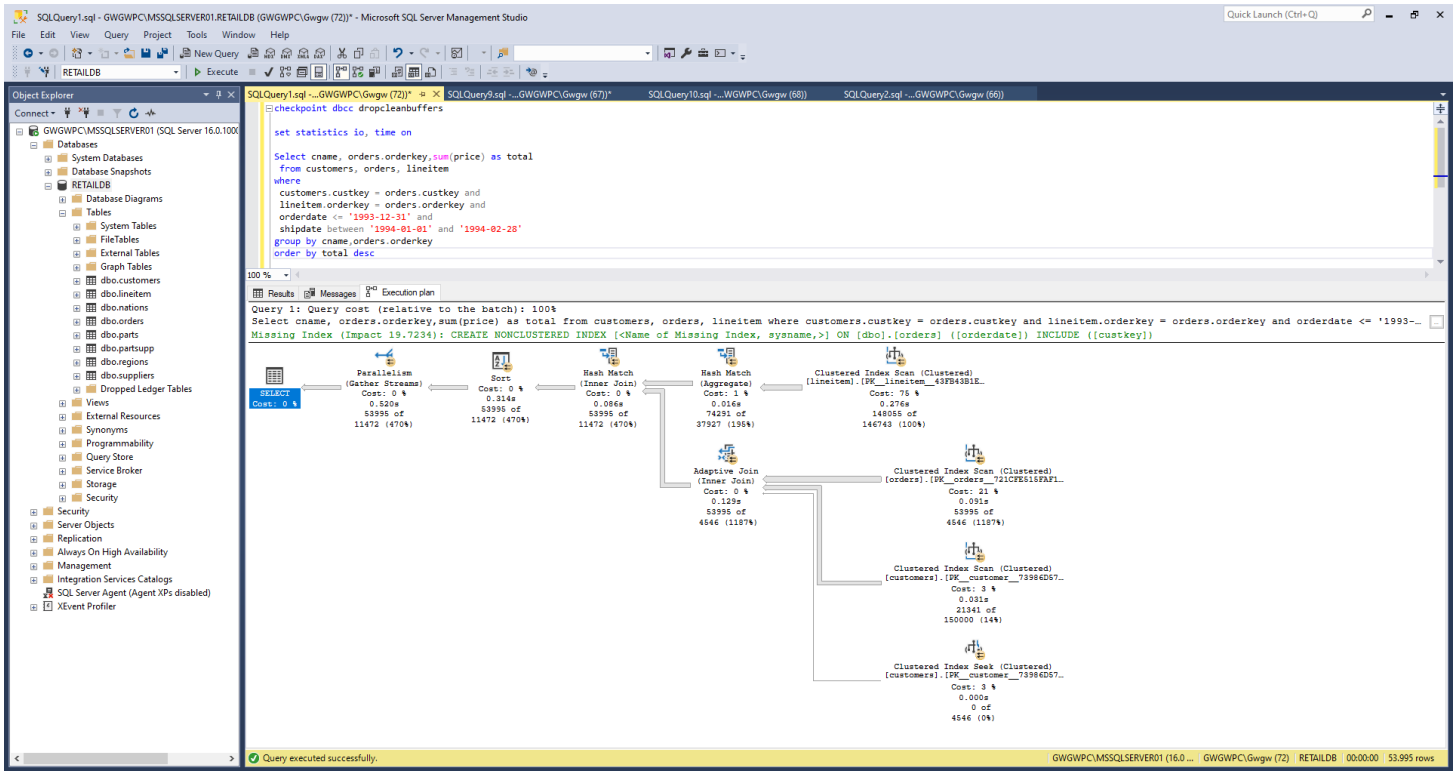
Ζήτημα 1ο

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





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 μπορούμενα 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.(Άρα 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 βελτιστοποιηση 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.

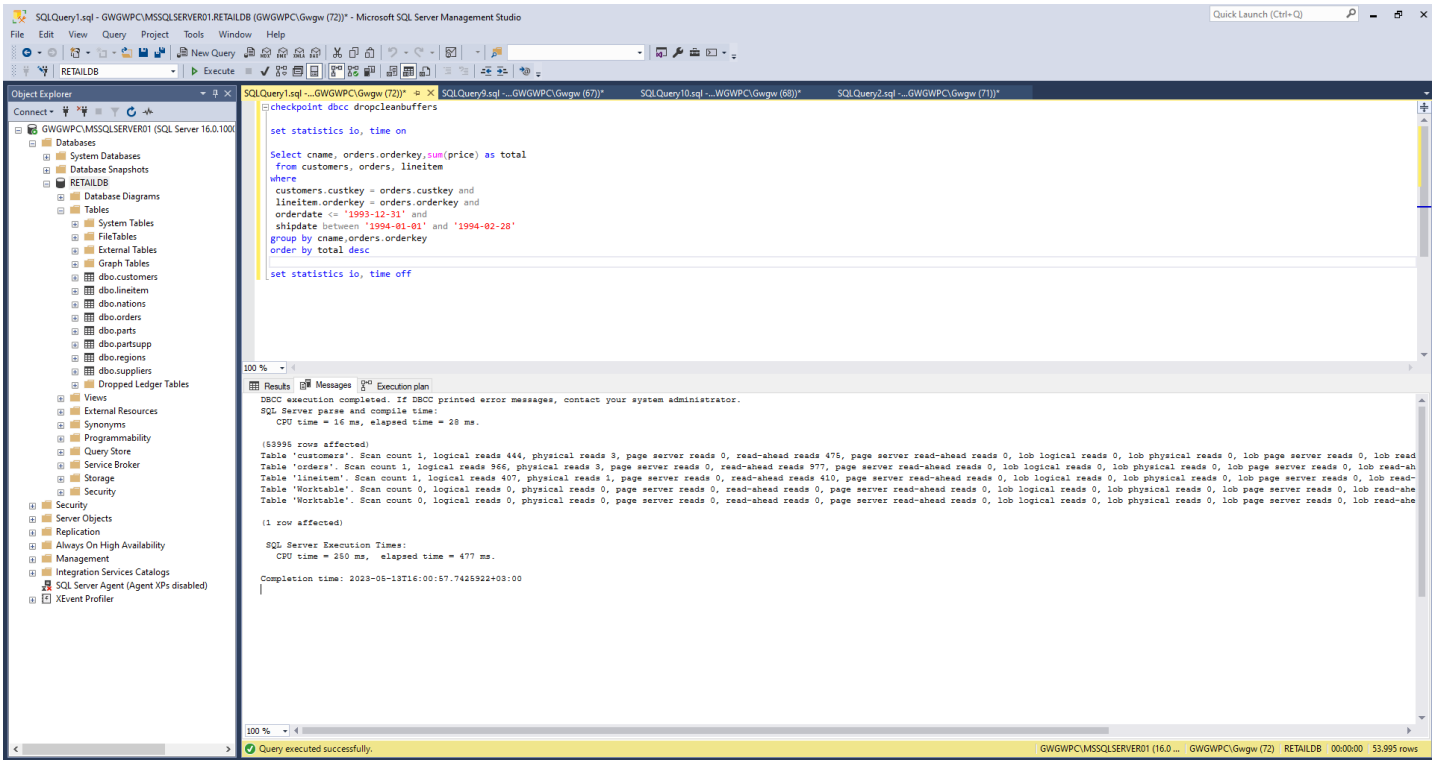
> *Αυτό το index βελτιώνει το join με τον πίνακα orders.*

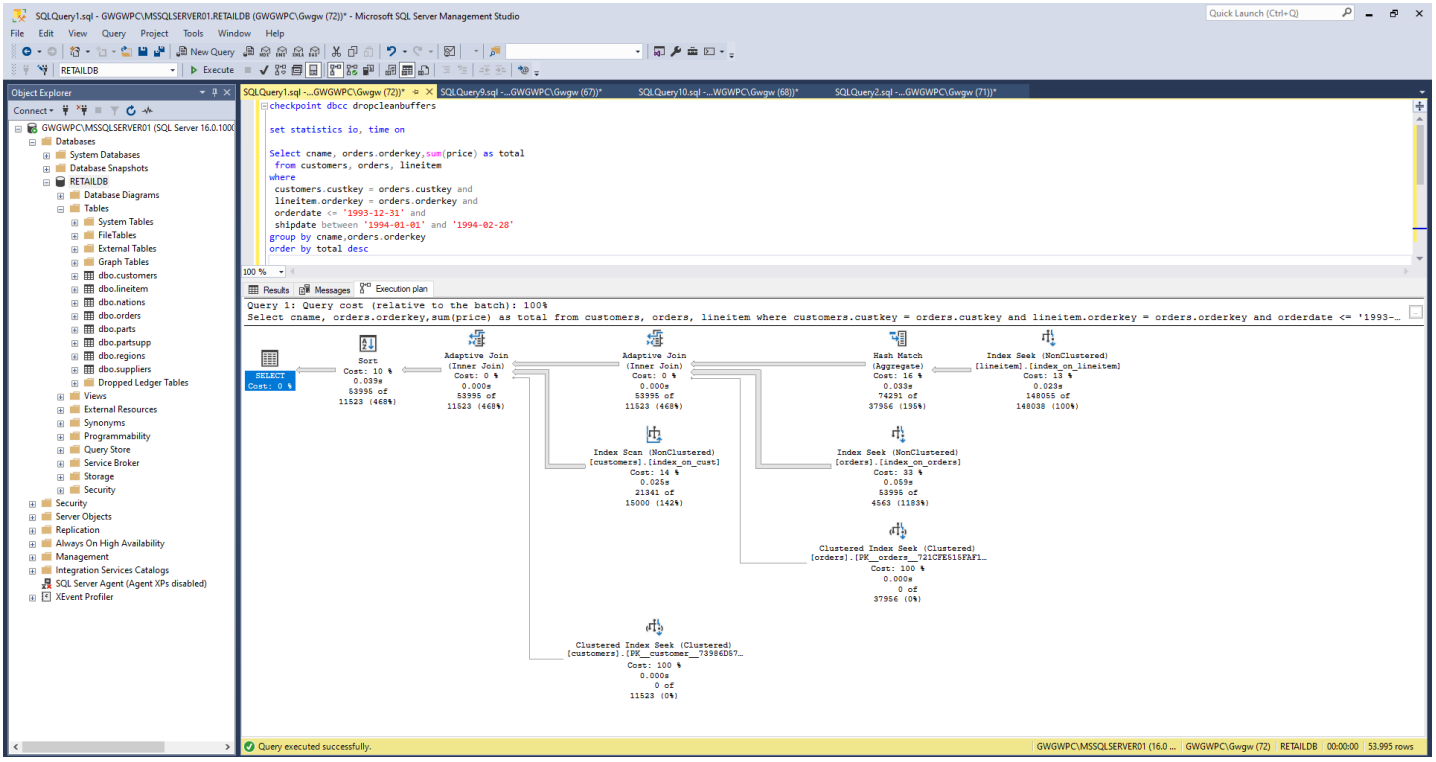
Ωστόσο επειδή θέλουμε να γίνει φιλτράρισμα ως προς το shipdate (και να μπορούμε να ανακτήσουμε το price γρηγορότερα) θα δημιουργήσουμε ένα index στον πίνακα lineitem στο shipdate το οποίο θα κάνει include το price.

```
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:

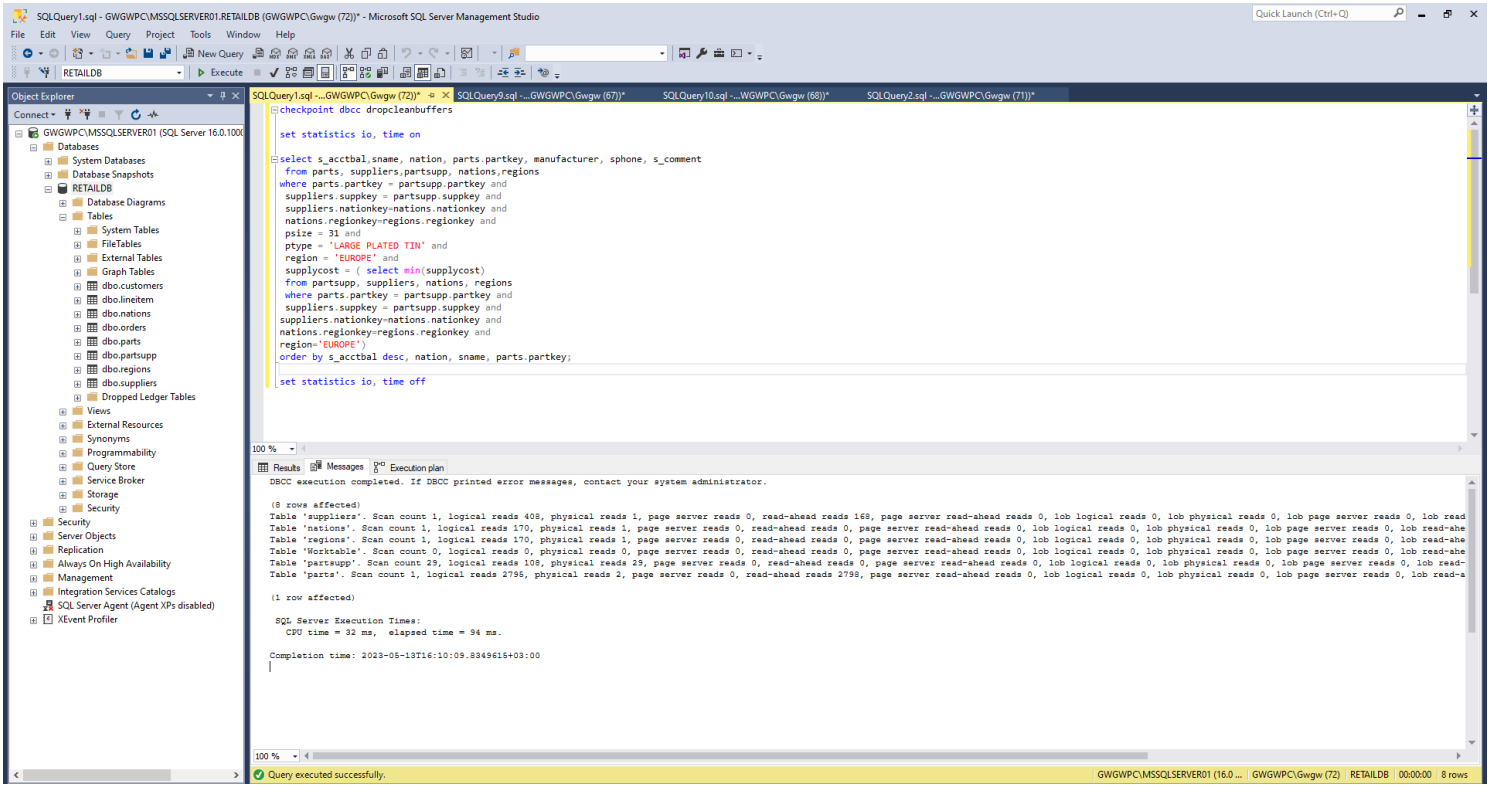
```
Table 'customers'. Scan count 1, logical reads 444,

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

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


SQLQuery1.sql - GWGWPC\MSSQLSERVER01.RETAILDB (GWGWPC\Gwgw (72))* - Microsoft SQL Server Management Studio
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
DBCC execution completed. If DBCC printed error messages, contact your system administrator.
SQL Server parse and compile time:
CPU time = 16 ms, elapsed time = 28 ms.
(53995 rows affected)
(1 row affected)
SQL Server Execution Times:
CPU time = 260 ms, elapsed time = 477 ms.
Completion time: 2023-05-13T16:00:57.7425922+03:00
Query executed successfully.

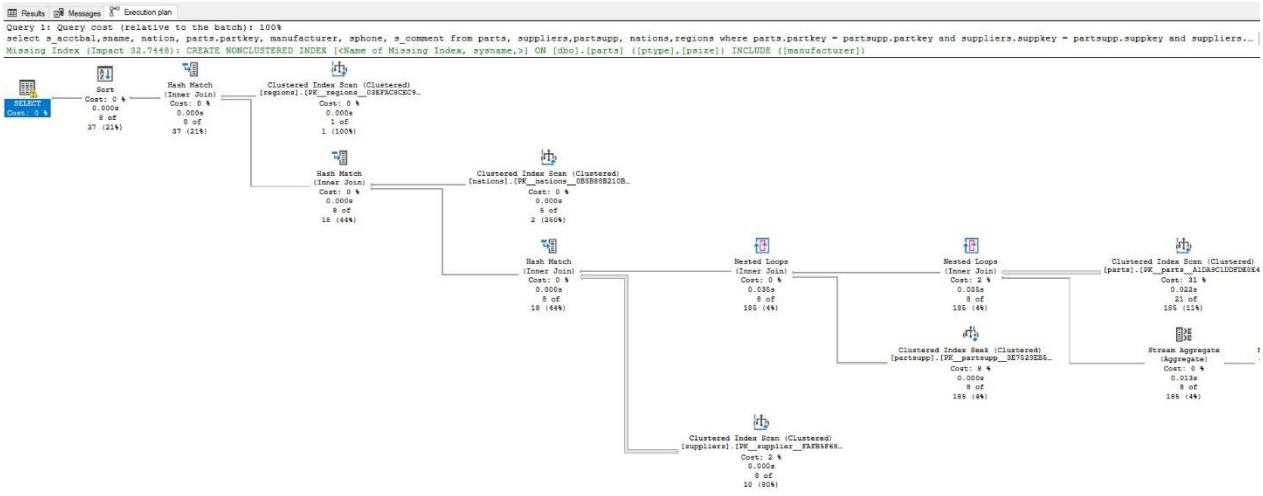

SQLQuery1.sql - GWGWPC\MSSQLSERVER01.RETAILDB (GWGWPC\Gwgw (72))* - Microsoft SQL Server Management Studio
Query 1: Query cost (relative to the batch): 100%
Select cname, orders.orderkey,sum(price) as total from customers, orders, lineitem where customers.custkey = orders.custkey and lineitem.orderkey = orders.orderkey and orderdate <= '1993-...
SELECT
Cost: 0 %
Sort
Cost: 10 %
0.039s
53995 of
11523 (468%)
Adaptive Join
(Inner Join)
Cost: 0 %
0.000s
53995 of
11523 (468%)
Adaptive Join
(Inner Join)
Cost: 0 %
0.000s
53995 of
11523 (468%)
Hash Match
(Aggregate)
Cost: 16 %
0.033s
74291 of
37956 (195%)
Index Seek (NonClustered)
[lineitem].[index_on_lineitem]
Cost: 13 %
0.023s
148055 of
148038 (100%)
Index Scan (NonClustered)
[customers].[index_on_cust]
Cost: 14 %
0.025s
21341 of
15000 (142%)
Index Seek (NonClustered)
[orders].[index_on_orders]
Cost: 33 %
0.059s
53995 of
4563 (1183%)
Clustered Index Seek (Clustered)
[orders].[PK__orders__721CFE518FAF1...
Cost: 100 %
0.000s
0 of
37956 (0%)
Clustered Index Seek (Clustered)
[customers].[PK__customer__7396D67...
Cost: 100 %
0.000s
0 of
11523 (0%)
Query executed successfully.

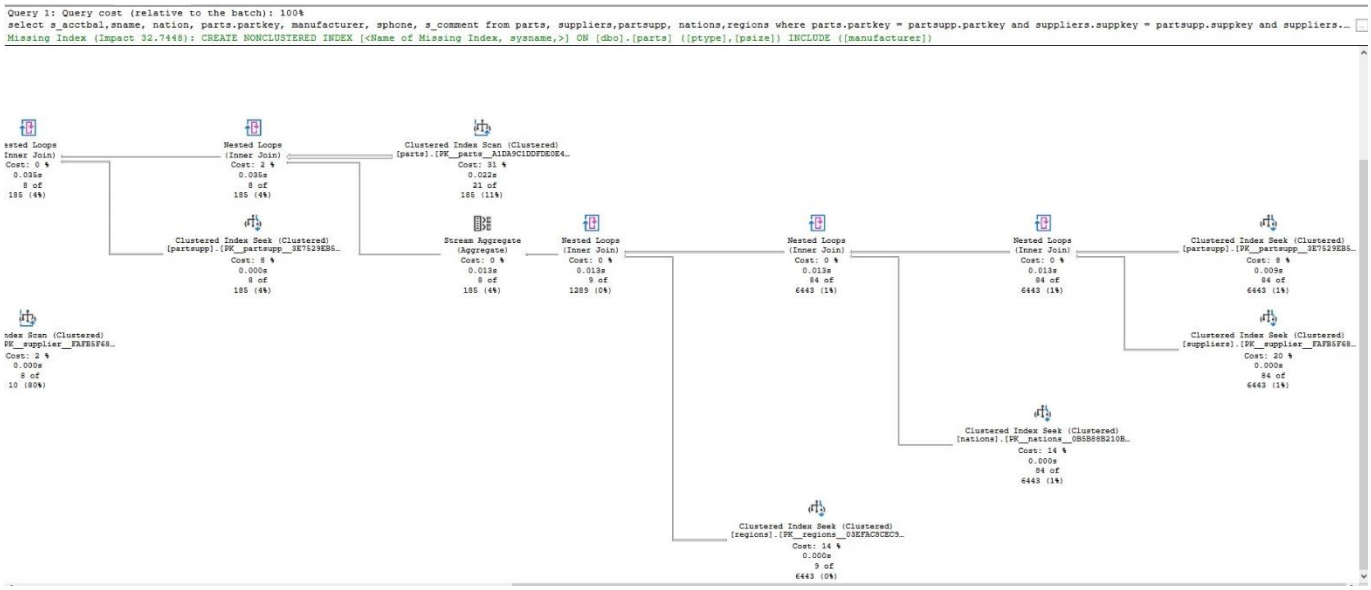
Ζήτημα 2ο

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 επειτα το psize. Επίσης θα κάνει include τους manufacturers (βρίσκονται στο select οπότε είναι καλή ιδέα να μπορούμε να τους ανακτάμε γρήγορα).

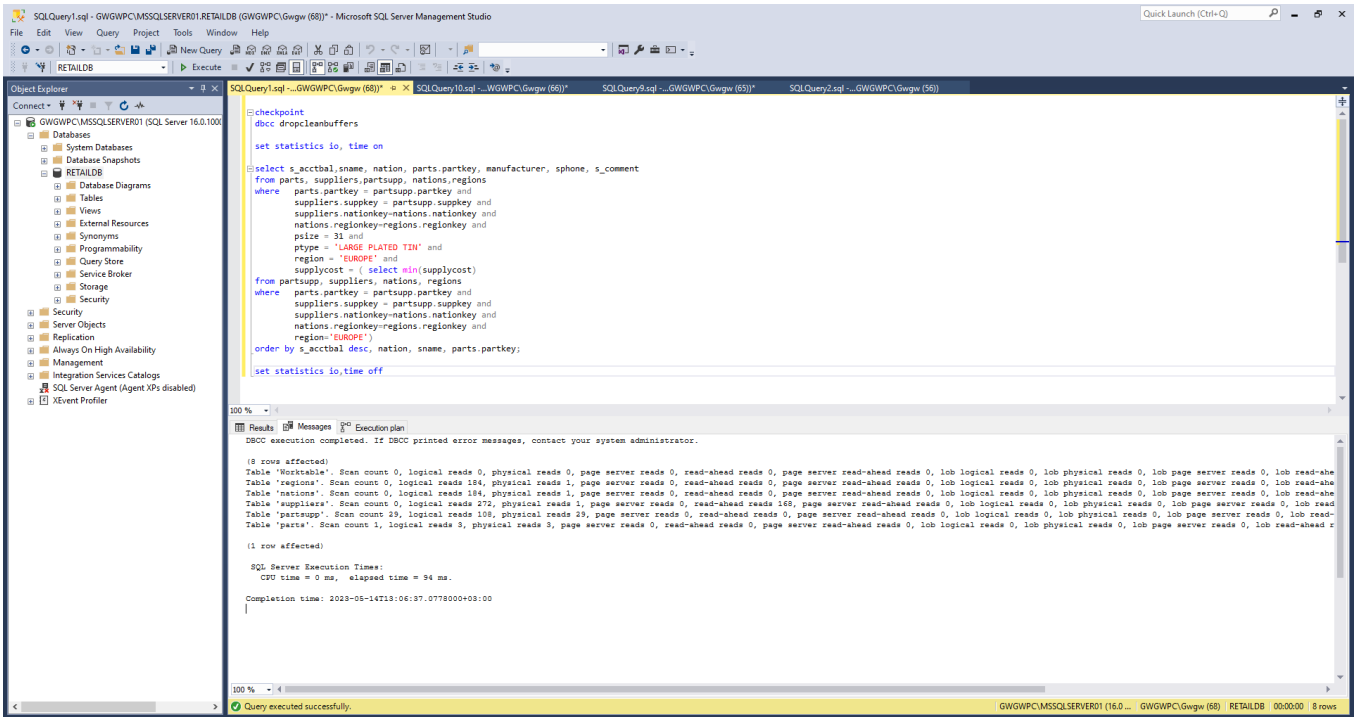
Άρα έχουμε:

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

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

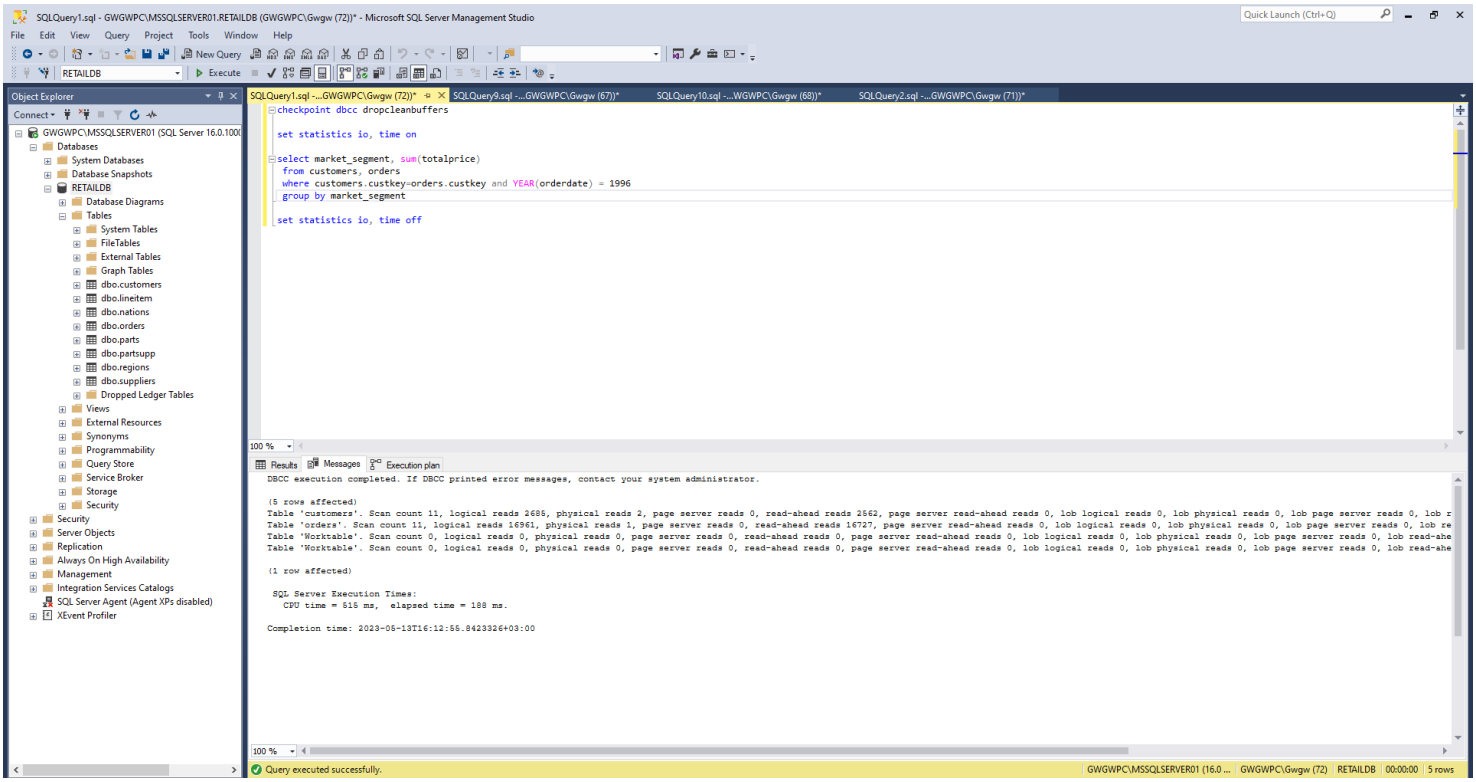
```
Table 'regions'. Scan count 0, logical reads 184, physical reads 1,
Table 'nations'. Scan count 0, logical reads 184, physical reads 1
Table 'suppliers'. Scan count 0, logical reads 272, physical reads 1,
Table 'partsupp'. Scan count 29, logical reads 108, physical reads 29,
Table 'parts'. Scan count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

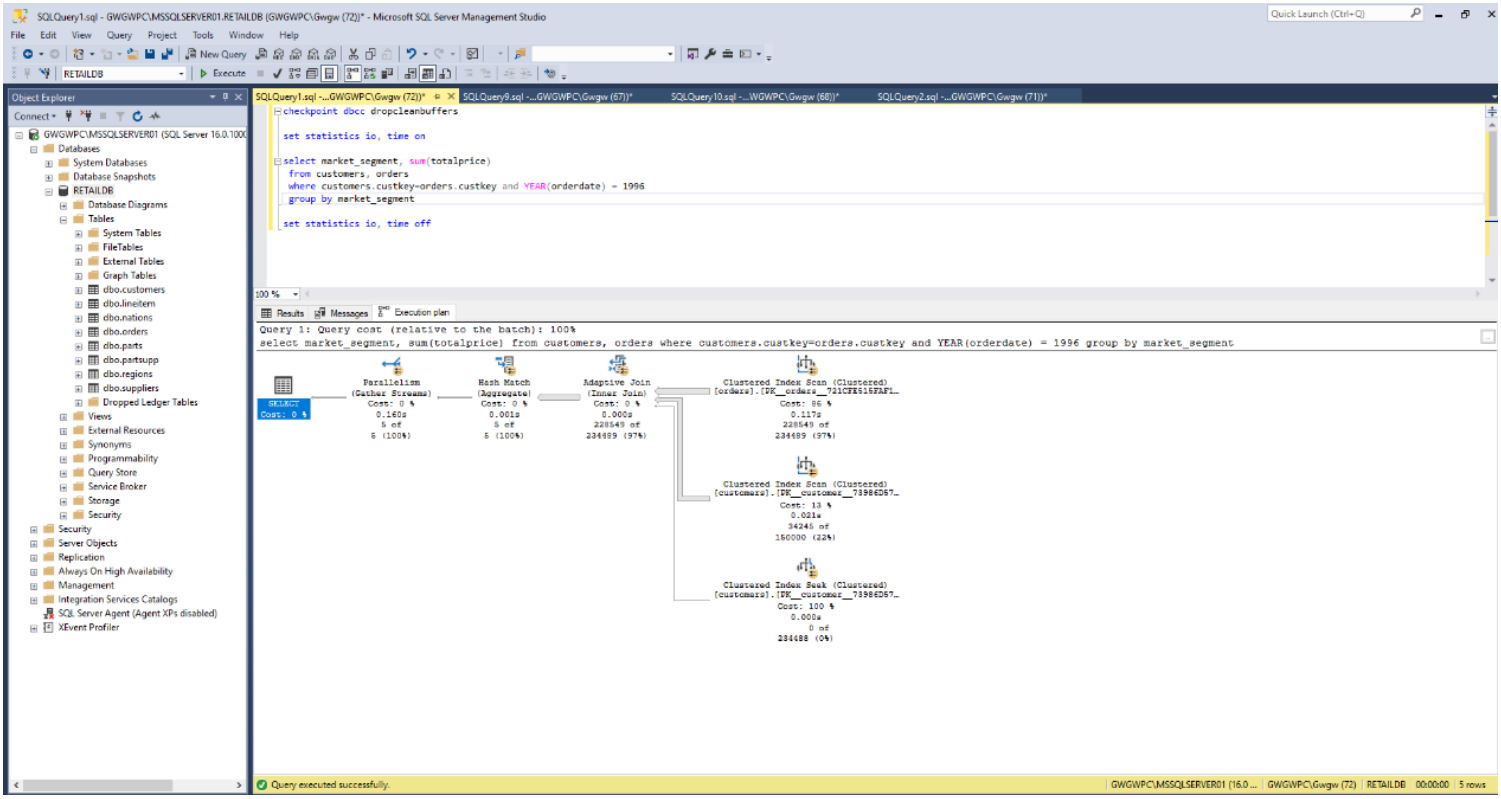
Έχουμε δηλαδή μεγάλη μείωση στα logical reads στον πίνακα parts



Ζήτημα 3ο

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






```
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 ισότητας 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

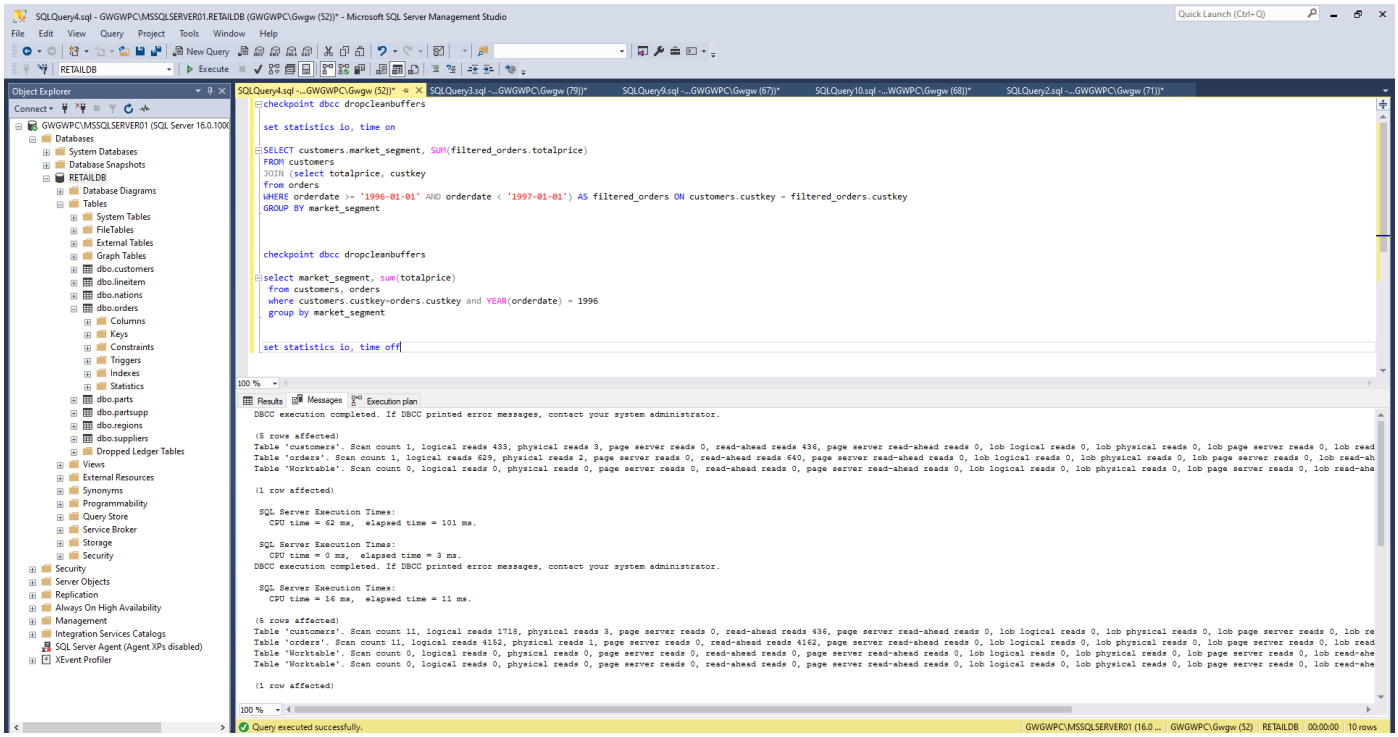
(όταν χρησιμοποιούμε τη συνάρτηση year() σε where clause η βάση θα χρειαστεί να κάνει full table scan για να υπολογίσει το year() σε κάθε row).

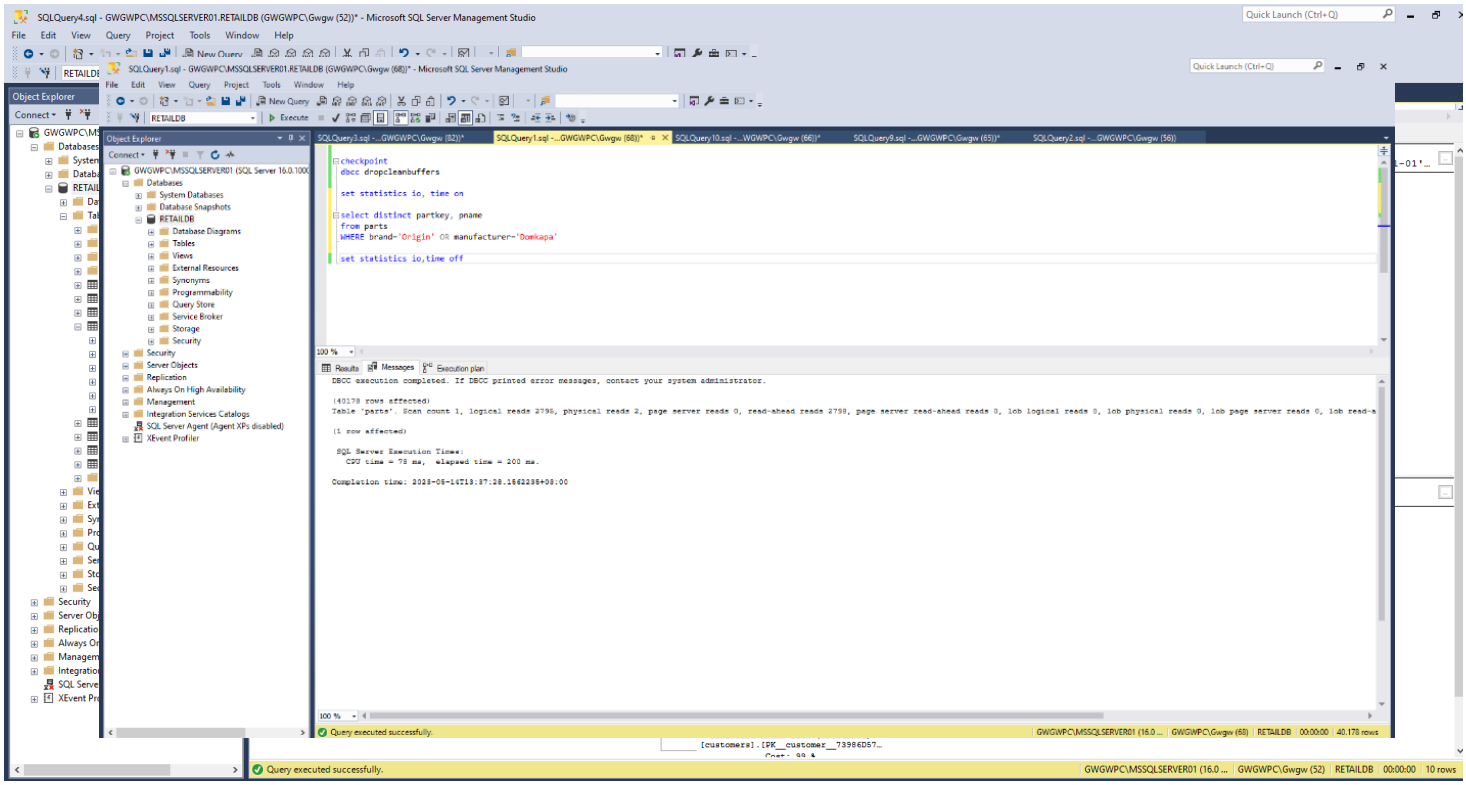
Έπειτα κάνουμε το join στους 2 πίνακες ως προς το custkey, έπειτα την ομαδοποίηση ανά market_segment και τον υπολογισμός του sum(totalprice)

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:





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,

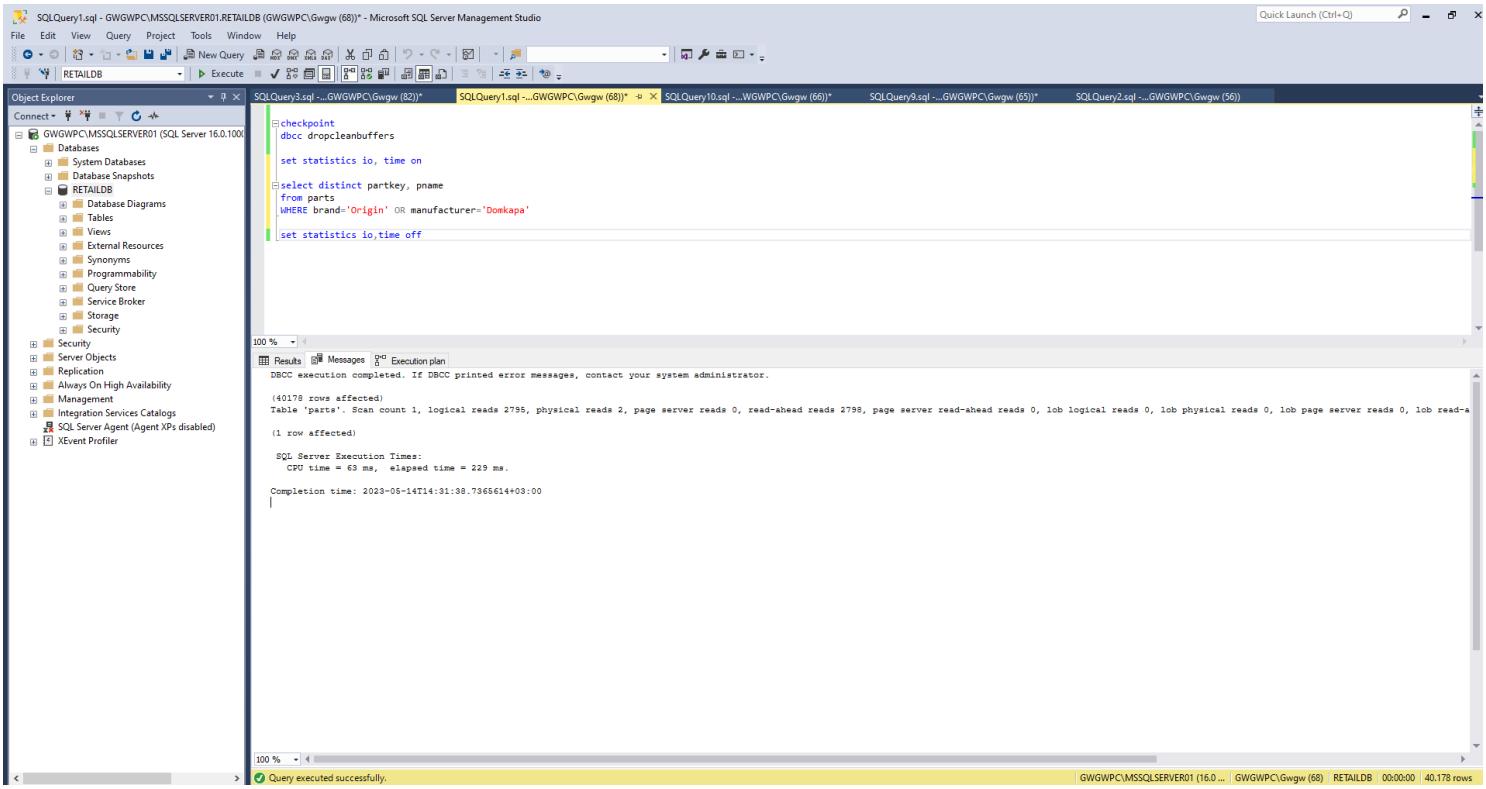
Table 'orders'. Scan count 11, logical reads 4152,

(Όπως φαίνεται και από την παραπάνω εικόνα)

Ζήτημα 4ο

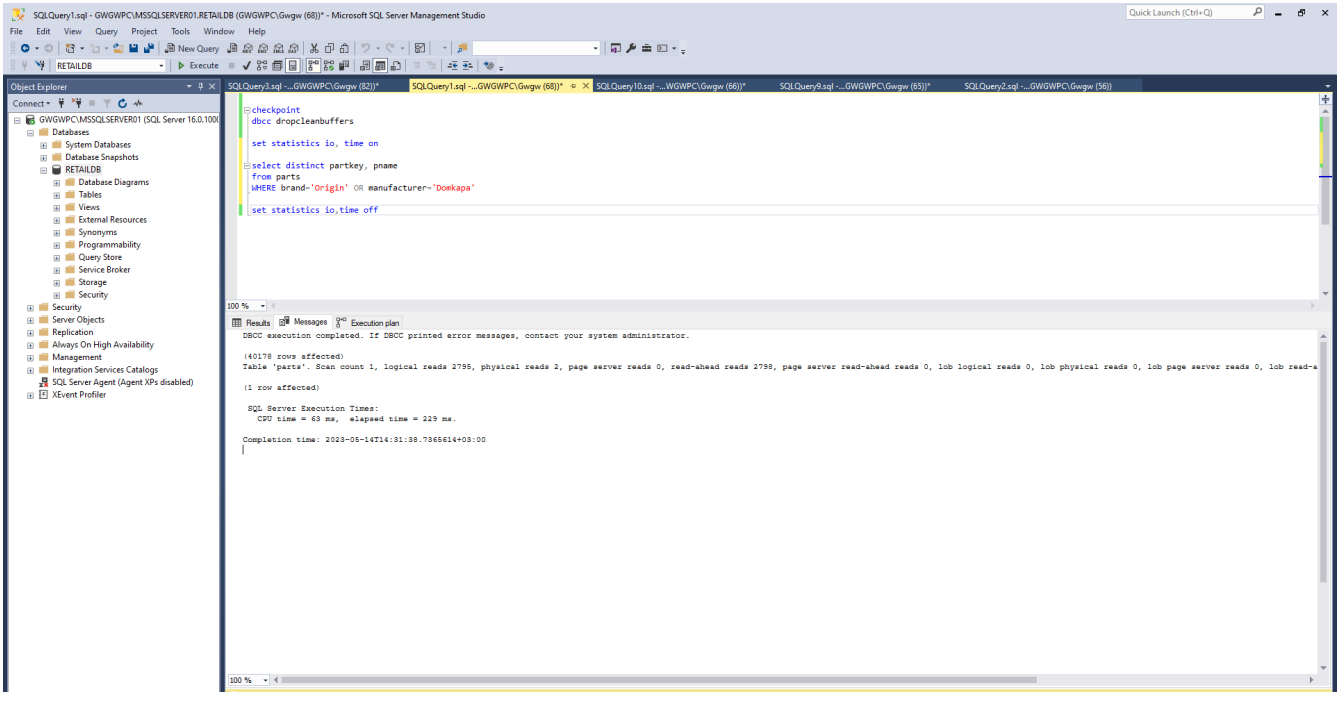
Α1) Για το πρώτο επερώτημα έχουμε:

- **Πριν την δημιουργία των ευρετηρίων:**
  Παρατηρούμε ότι συνολικά έχουμε:
  Table 'parts'. Scan count 1, logical reads 2795.



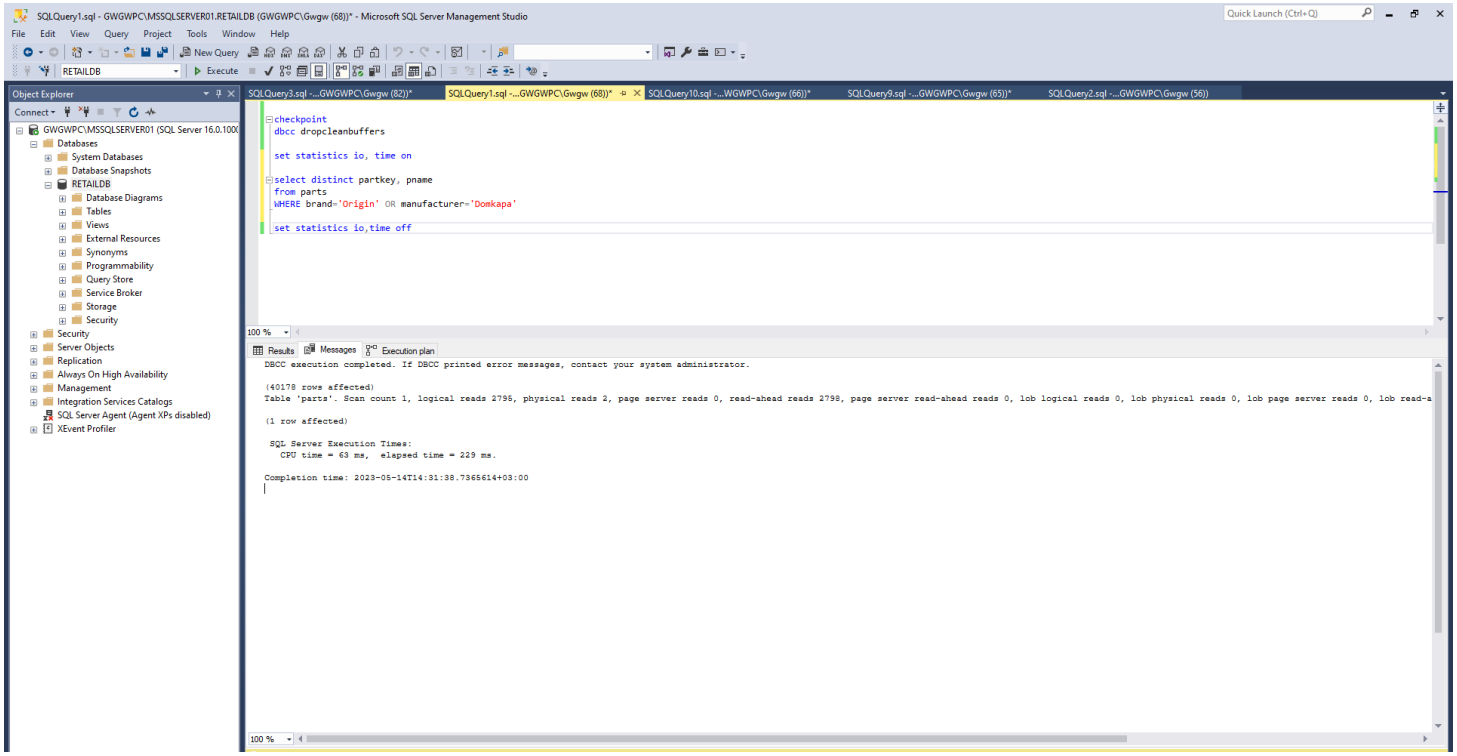
- Δημιουργία idx1 (only)

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)

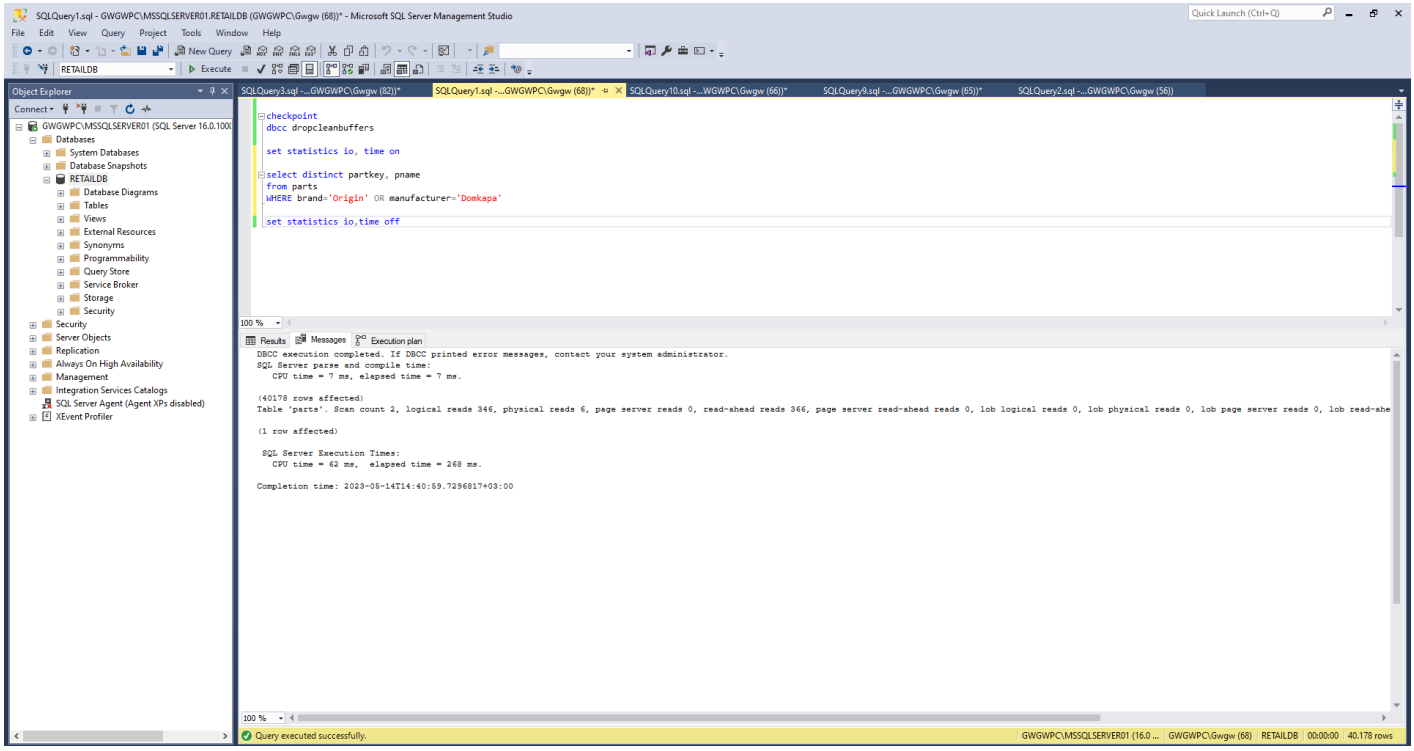
  Το ίδιο ισχύει και για το δεύτερο ευρετήριο. Δηλαδή παρατηρούμε ότι εφόσον υπάρχει τελεστής OR στο επερώτημα, παρότι μπορεί να ελέγξει ως προς αυτό το ευρετήριο δεν συμφέρει γιατί δεν ελέγχει ως προς το άλλο. Οπότε είναι πιο αποδοτικό να κάνει ένα scan σε όλο μαζί. Αυτό επιβεβαιώνεται και με την εκτέλεση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



- Δημιουργία idx1, idx2.

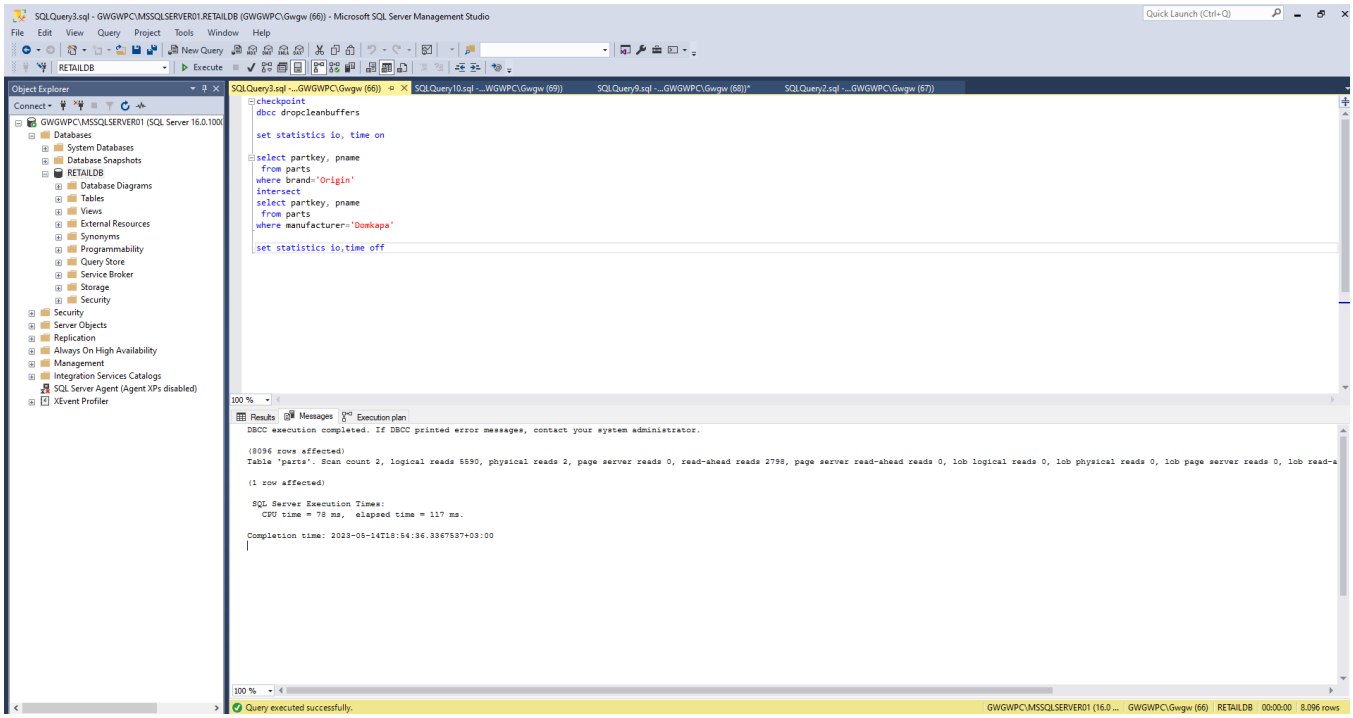
  Εδώ παρατηρούμε ότι εφόσον έχουμε ευρετήρια και στα μέρη του OR, αυτά θα χρησιμοποιηθούν για την εκτέλεση του επερωτήματος. Αυτό επιβεβαιώνεται και από τα logical reads που εκτελούνται μετά τη δημιουργία των ευρετηρίων.

(Τα logical reads είναι 346)



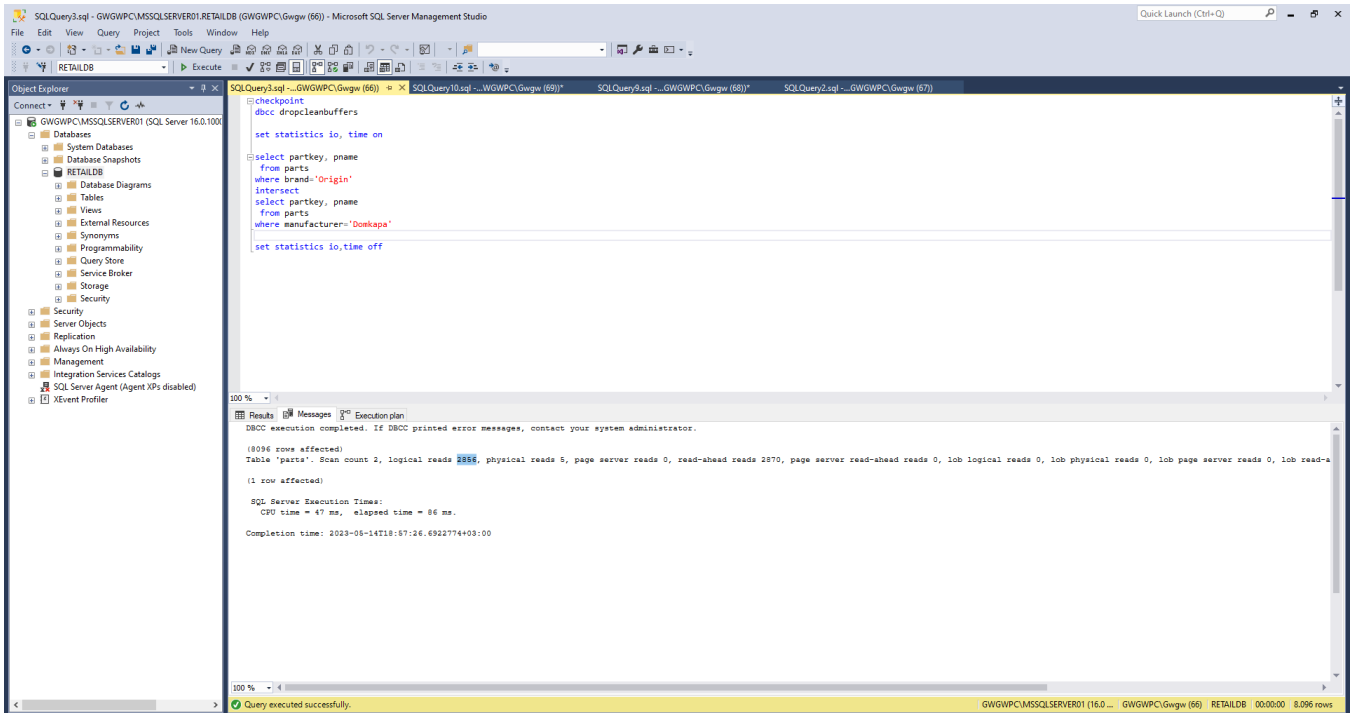
Α2) Για το δεύτερο επερώτημα έχουμε:

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



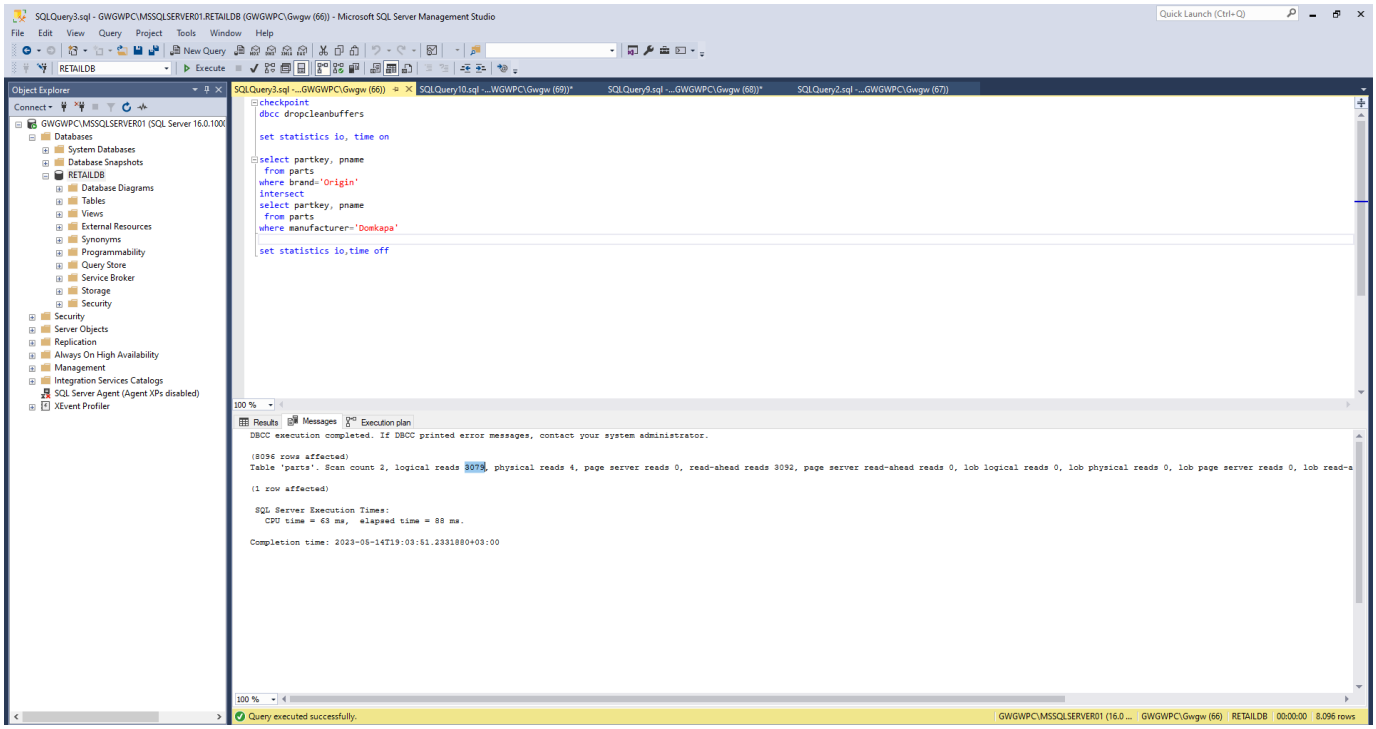
- Δημιουργία idx1 (only)

Εδώ παρατηρούμε ότι υπάρχει INTERSECT στο επερώτημα, 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)



- Δημιουργία idx2 (only)

Κι εδώ παρατηρούμε ότι υπάρχει INTERSECT στο επερώτημα, 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 idx2. (Τα logical reads έγιναν 3079)

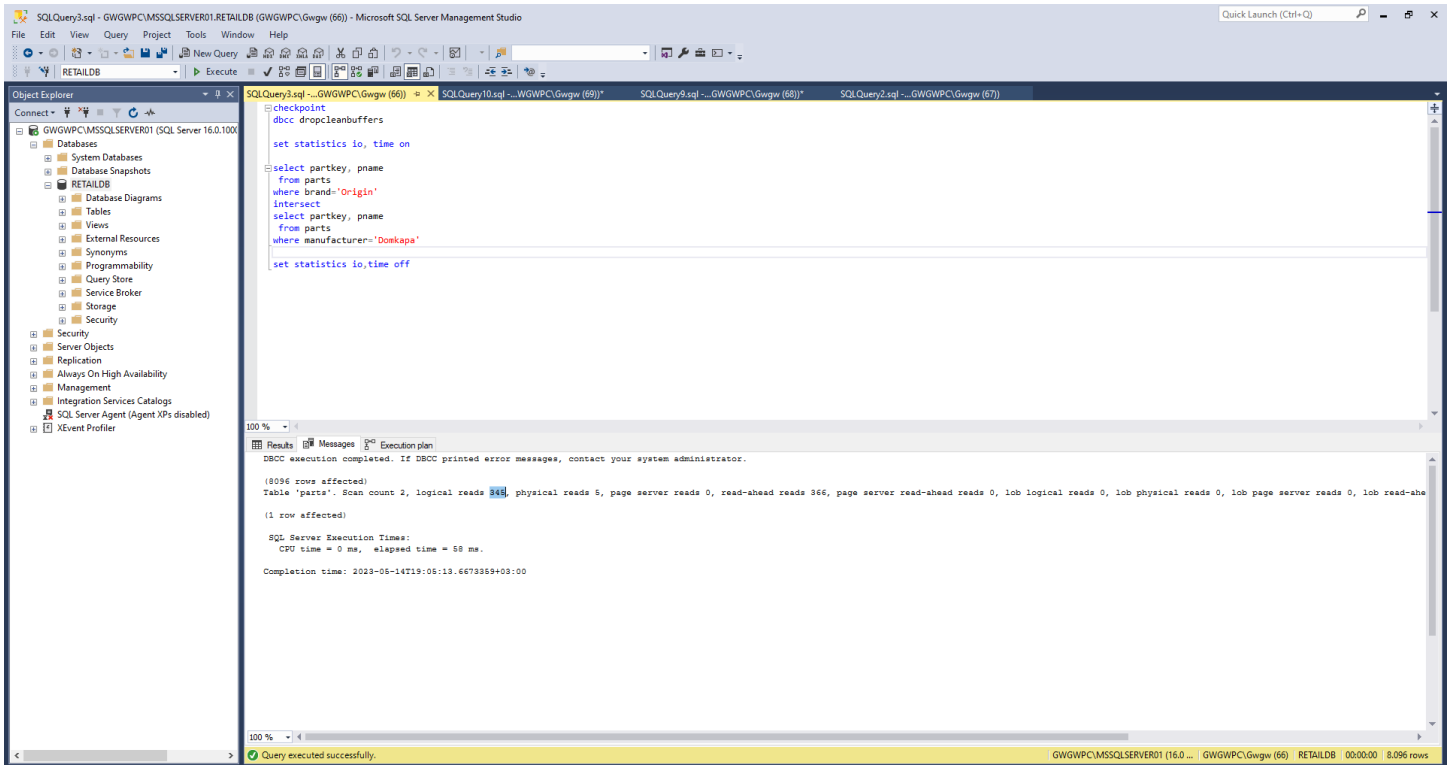


- Δημιουργία idx1, idx2.

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 επερώτημα έχουμε:

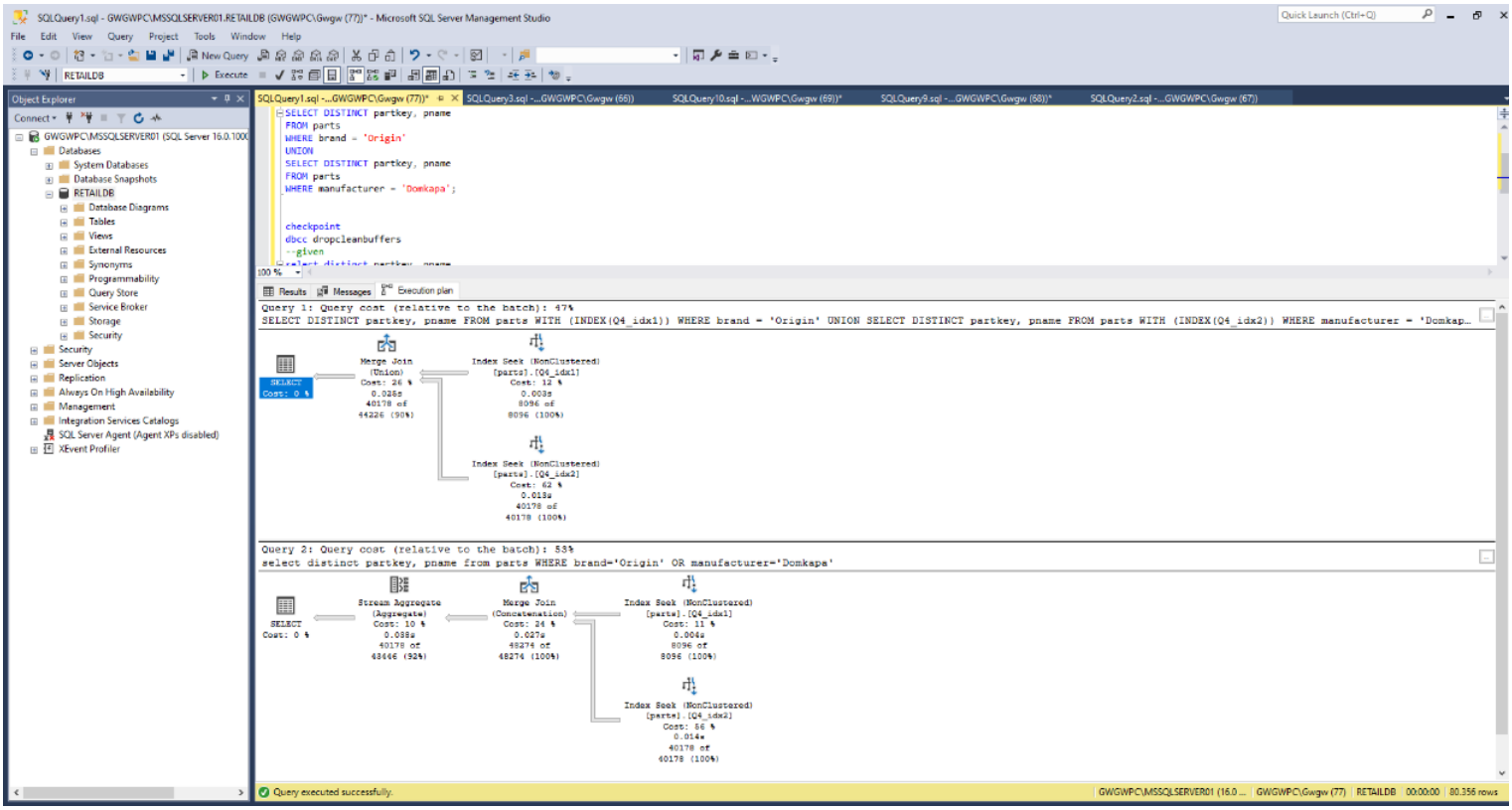
Τα indexes που υπάρχουν δεν θεωρούμε ότι χρειάζονται βελτίσωση, ούτε ότι μπορεί να χρησιμοποιηθεί κάποιο άλλο εφόσον από αυτά παίρνουμε τις πληροφορίες που χρειαζόστε για να βελτιστοποιήσουμε τα logical reads. Ωστόσο, το επερώτημα μπορεί να βελτιωθεί ως εξής:

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

Με αυτό το επερώτημα 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 επερώτημα, 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 επερωτήσεων, 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 επερώτημα έχουμε:

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

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

Και να δημιουργηθεί ένα μόνο index

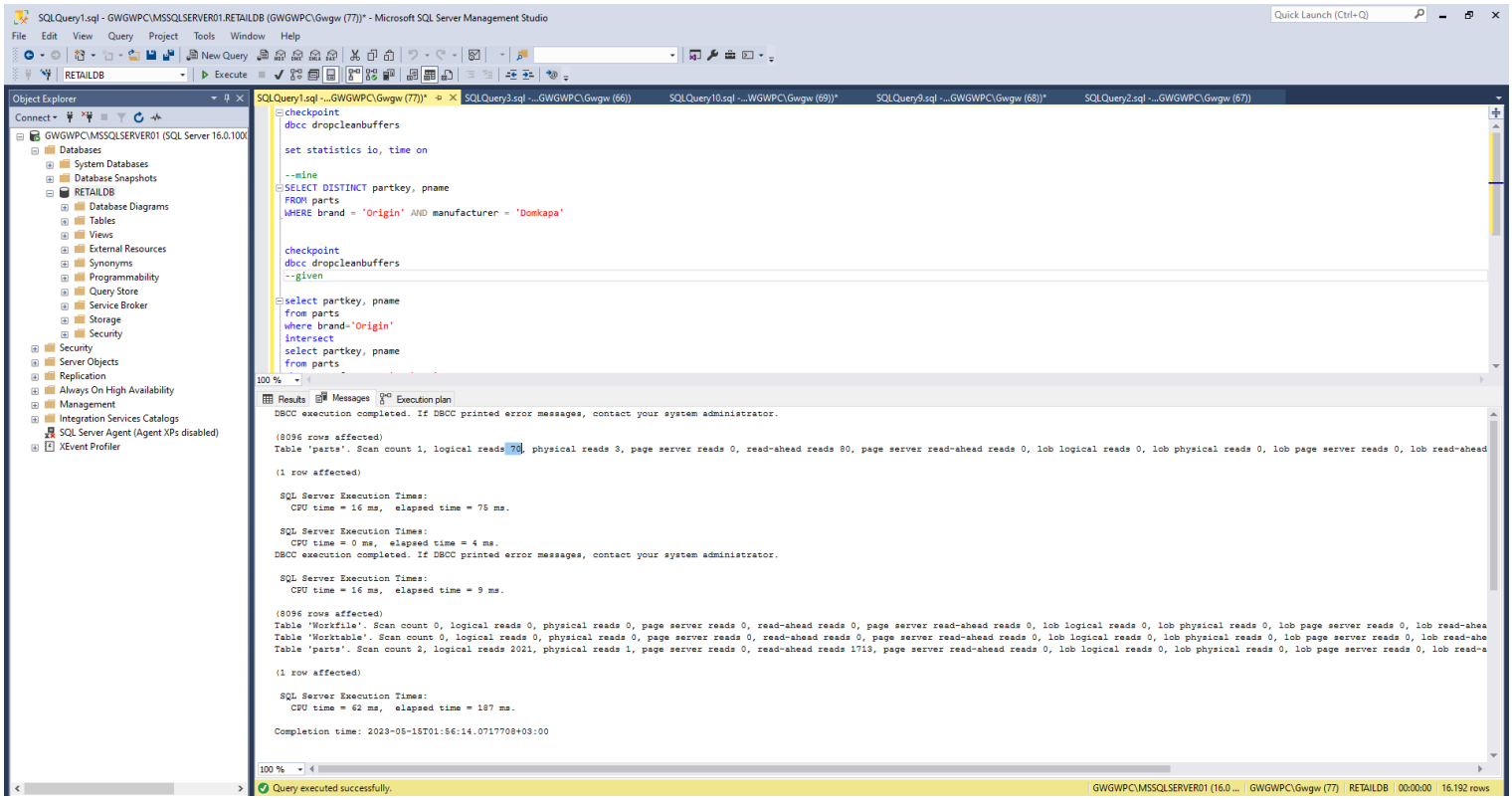
```
CREATE INDEX index_on_parts ON parts (manufacturer, brand, partkey) INCLUDE
(pname)
```

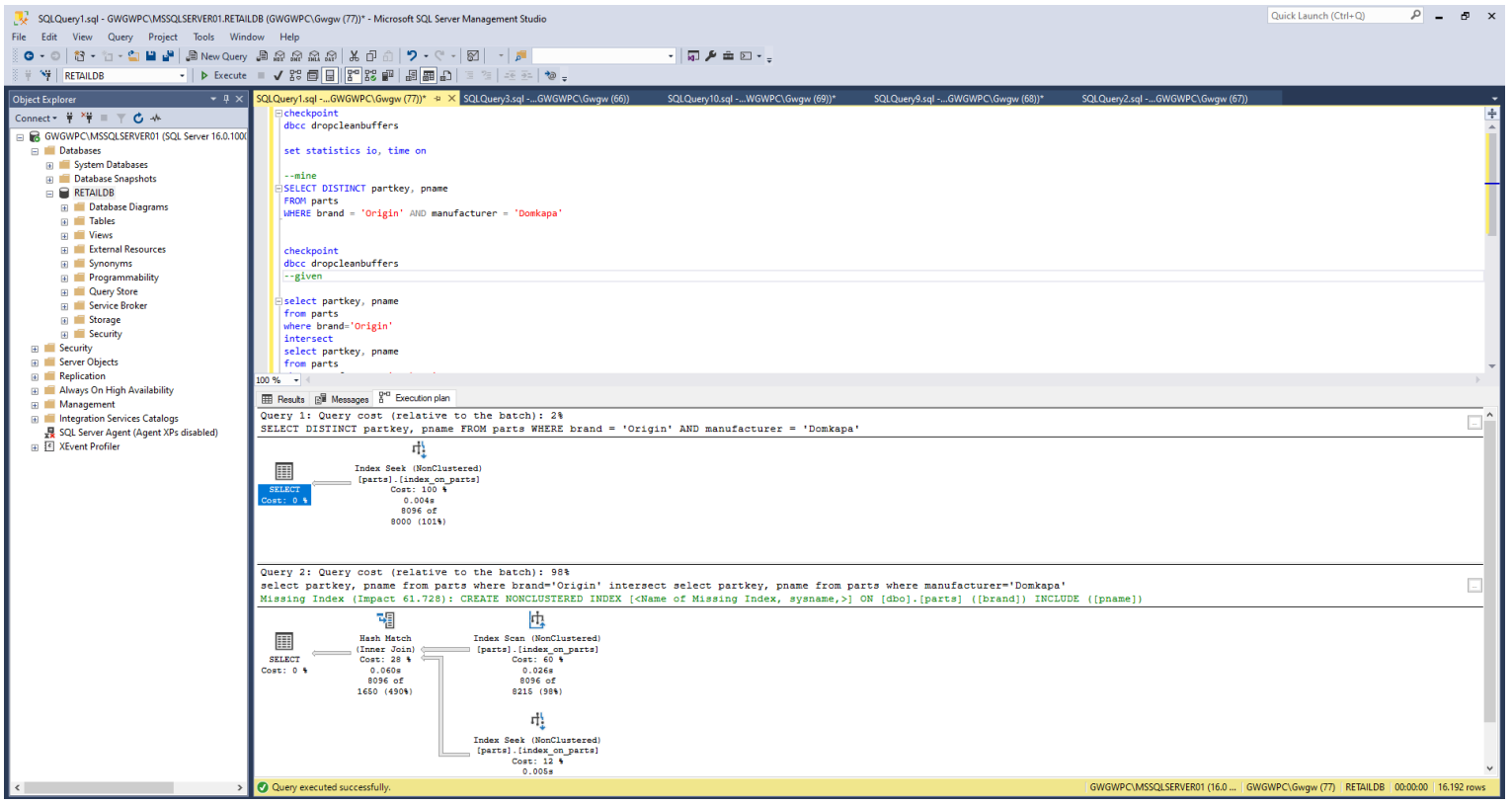
Έτσι, αρχικά δημιουργούμε μόνο ένα ευρετήριο (και όχι 2 όπως προτείνεται από την εκφώνηση), το οποίο ταξινομείται κατά (manufacturer, brand, partkey). Επομένως, η βάση μπορεί να πραγματοποιήσει index seek στο σχετικό τμήμα του ευρετηρίου και να ανακτήσει γρήγορα μόνο τα απαραίτητα rows.

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 επερωτήματος 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 επερωτημάτων είναι 2%-98% (αρκετά υψηλή διαφορά!) και ότι τα logical reads με το προτεινόμενο επερώτημα είναι 70 ενώ με το δωσμνένο 2021.





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 Α2) .

Ζήτημα 5ο

1ο 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:

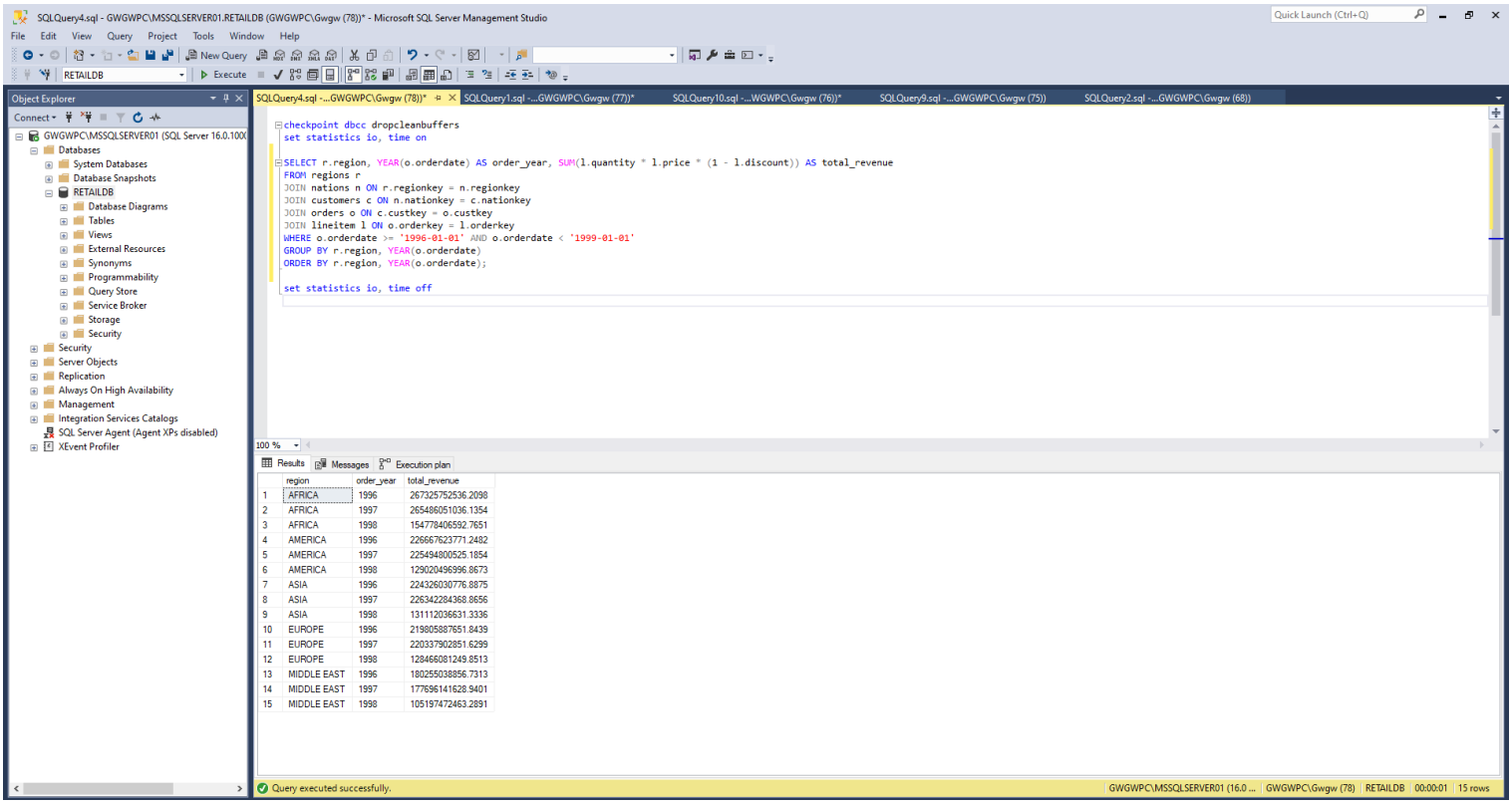
```
SELECT r.region, YEAR(o.orderdate) AS order_year, SUM(l.quantity * l.price * (1
- l.discount)) AS total_revenue
FROM regions r
JOIN nations n ON r.regionkey = n.regionkey
JOIN customers c ON n.nationkey = c.nationkey
JOIN orders o ON c.custkey = o.custkey
JOIN lineitem l ON o.orderkey = l.orderkey
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);
```

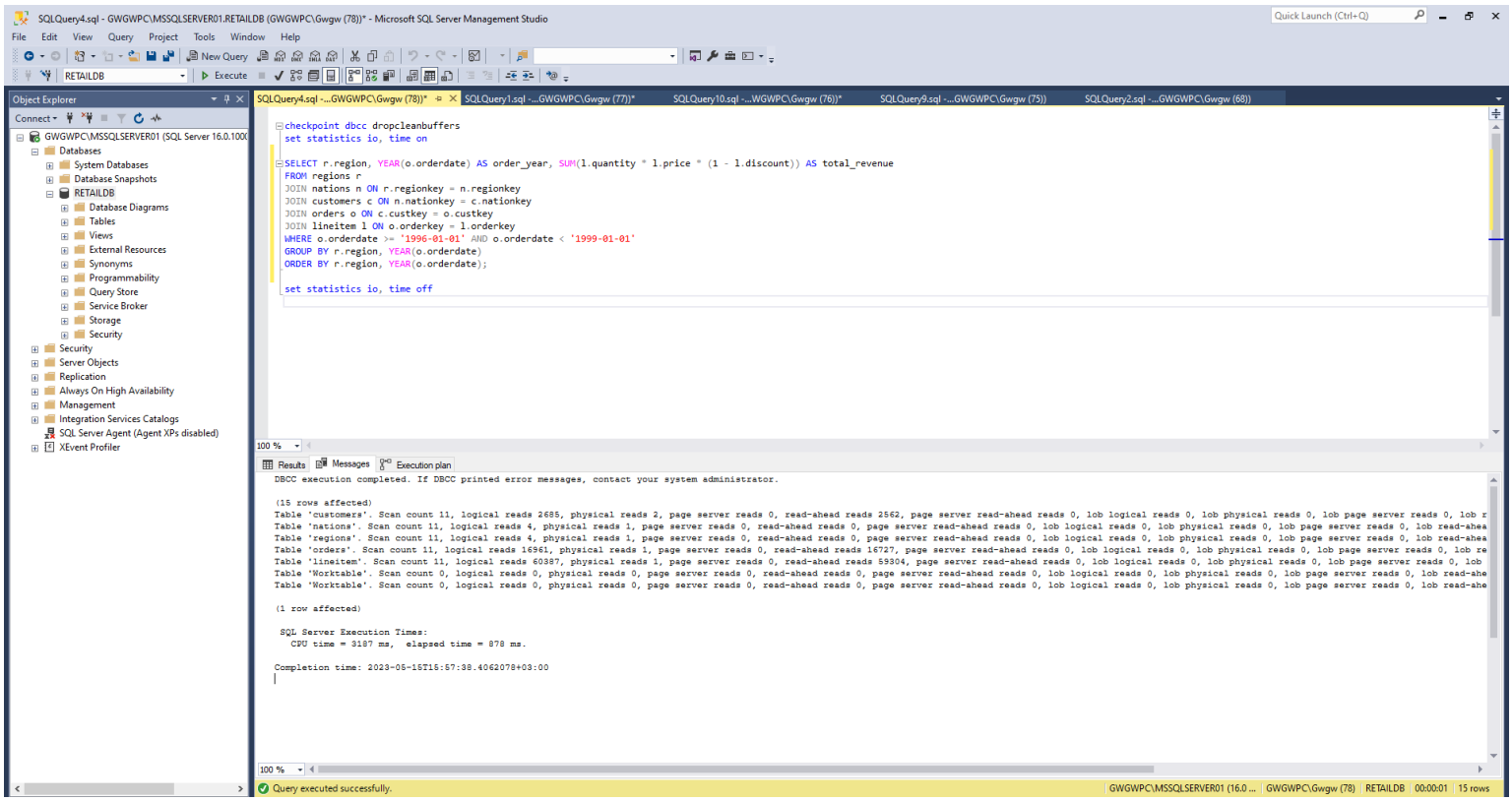
1ο επερώτημα, indexes)

Τα indexes που δημιοργούμε είναι τα εξής:

```
create INDEX index_on_orders ON orders (orderdate) 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 (orderkey) INCLUDE (discount, price,
quantity);
```

Πριν από την δημιουργία των indexes έχουμε





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

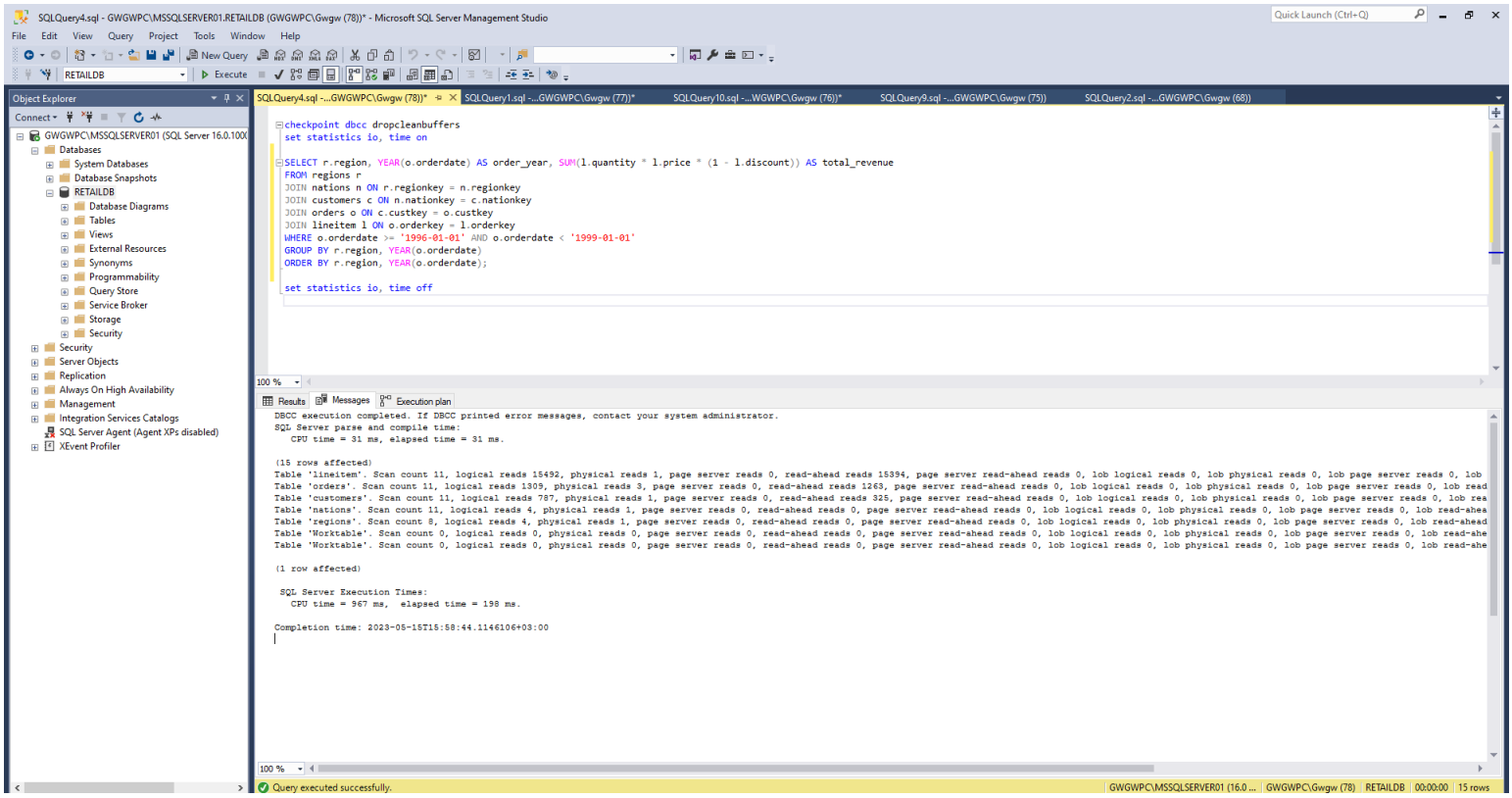
Table 'lineitem'. Scan count 11, logical reads 15492,

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



2ο 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 αρκιβώς 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;
```

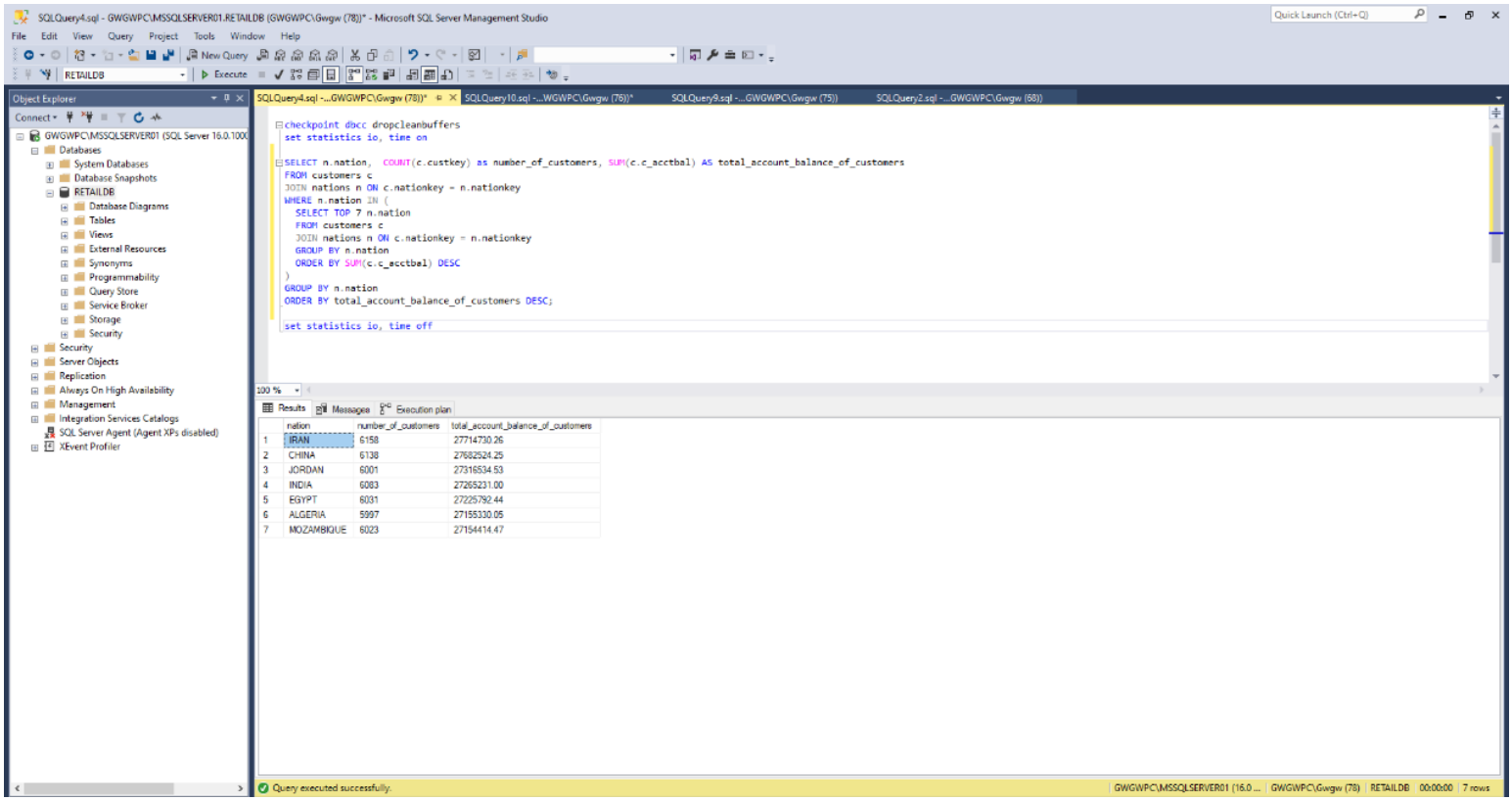
2ο επερώτημα, indexes)

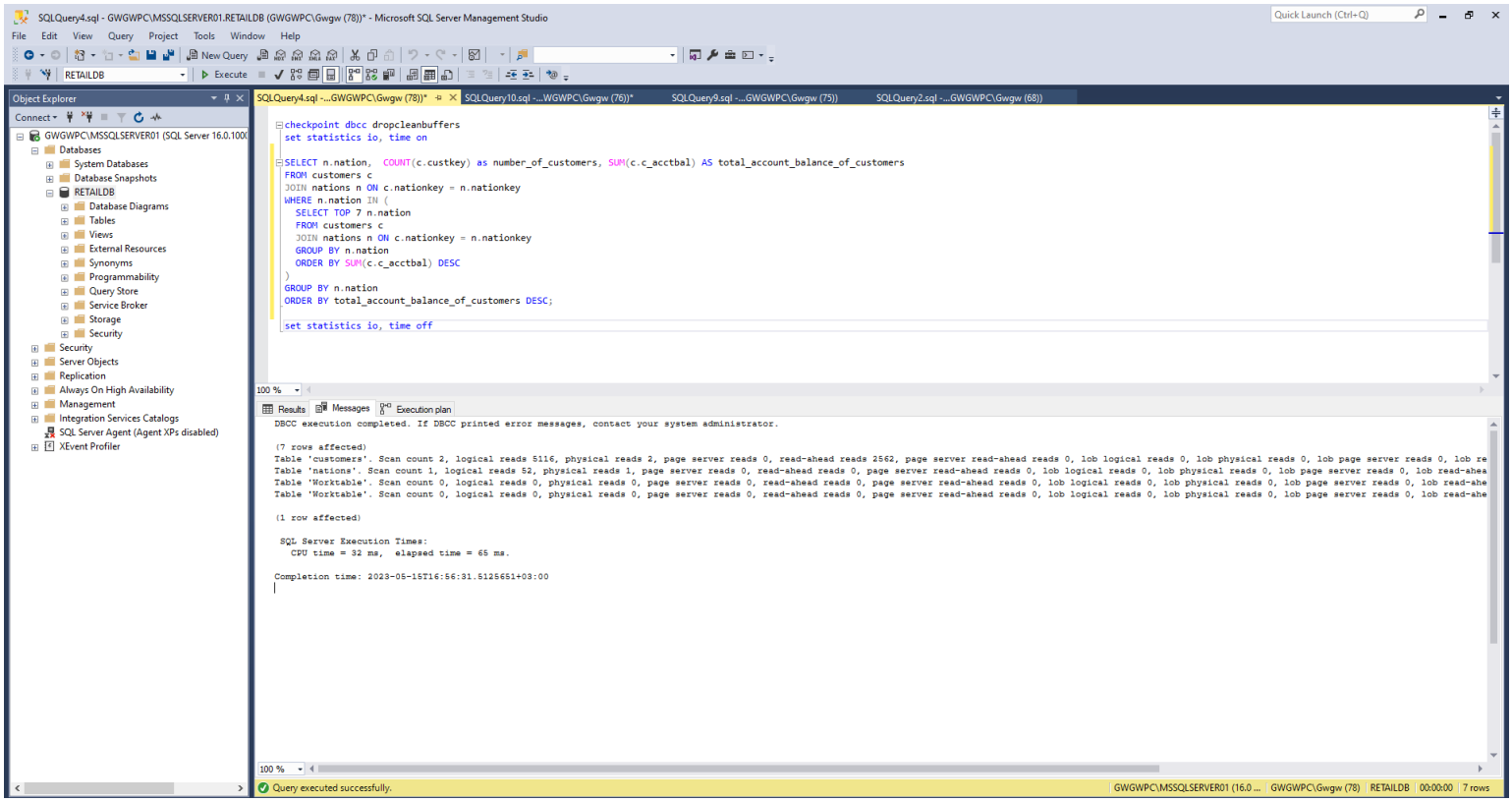
Το index που μπορούμε να δημιουργήσουμε για την βελτιστοποίηση αυτού του επερωτήματος είναι το εξής:

```
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:





Πιο συγκεκριμένα:

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

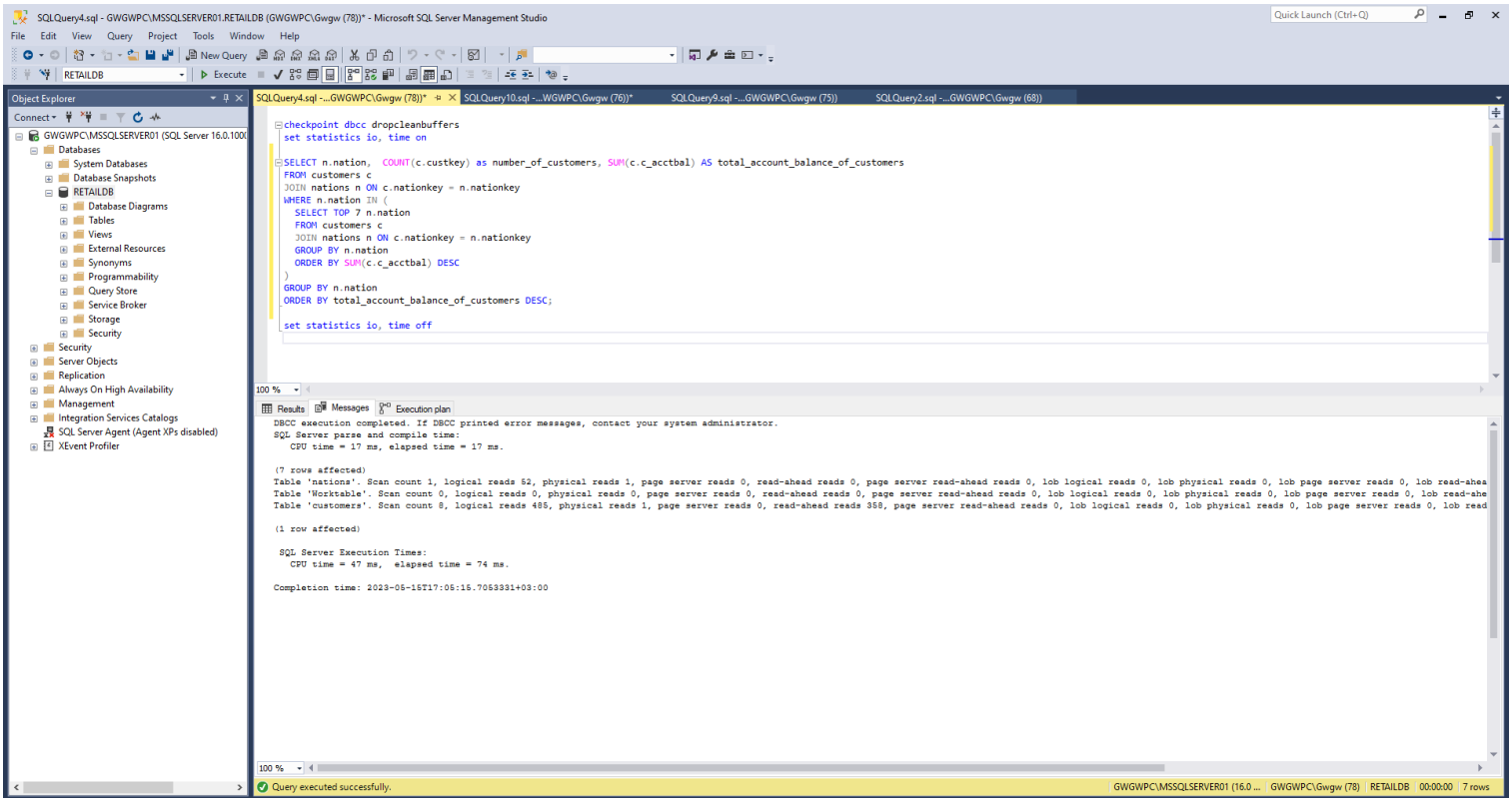
Table 'customers'. Scan count 8, logical reads 5116

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 επιβεβαίωνα 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`