ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

Συστήματα Διαχείρισης και Ανάλυσης Δεδομένων (2022-2023)

Διονύσιος Ρηγάτος (P3200262)

1η Εργασία

Ζήτημα 1ο

Query

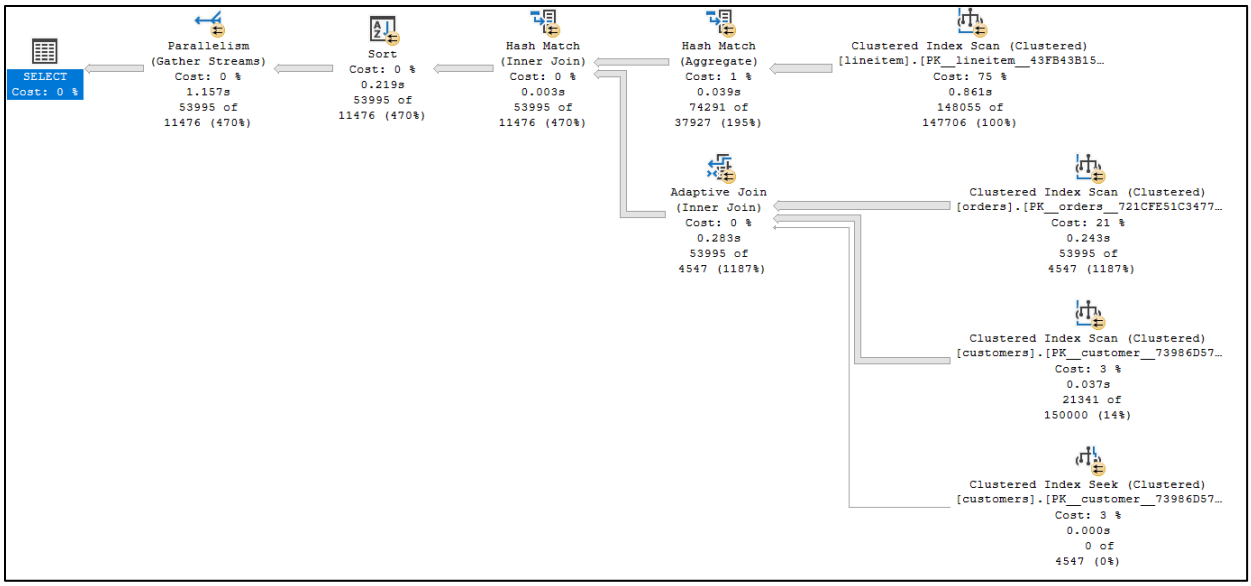
```
select cname, orders.orderkey,sum(price) as total
from customers, orders, lineitem
where
 customers.custkey = orders.custkey and
 lineitem.orderkey = orders.orderkey and
 orderdate <= '1993-12-31' and
 shipdate between '1994-01-01' and '1994-02-28'
group by cname,orders.orderkey
order by total desc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

Αρχικά Στατιστικά

Row Num	Table	Scan Count	Logical Reads	Physical Reads	Read-Ahead Reads
	customers	13	2,685	2	2,562
	orders	13	16,925	1	16,727
	lineitem	13	60,387	1	59,294
	Worktable	0	0	0	0
	Worktable	0	0	0	0

	CPU	Elapsed
SQL Server parse and compile time:	00:00:00.000	00:00:00.015
SQL Server Execution Times:	00:00:00.016	00:00:01.410
Total	**00:00:00.016**	**00:00:01.425**

Αρχικό Πλάνο Εκτέλεσης



Παρατηρήσεις:

- Γίνονται υπερβολικά πολλά logical reads στα lineitem, orders και customers.
- Κοστοβόρο (75%) tablescan πάνω στο lineitem για την aggregate συνάρτηση που καλείται στο lineitem.price. Παρατηρούμε επίσης ότι έχουμε range από ημερομηνίες του lineitem.shipdate στο where του query. Δεν αποτελεί κλειδί και συνεπώς δεν υπάρχει clustered index πάνω σε αυτό.
- Κοστοβόρο (21%) tablescan πάνω στο orders για να γίνει το join μεταξύ orders, customers. Παρατηρούμε πως έχουμε date range περιορισμό βάσει του orders.orderdate, το οποίο δεν αποτελεί κλειδί.
- Group by βάσει του customers.cname, στο οποίο δεν υπάρχει ευρετήριο by default καθώς δεν αποτελεί κλειδί.

Ιδέες

- Ευρετήριο και στα δύο date ranges που βρήκαμε, καθώς θα επωφεληθούμε σημαντικά σε ταχύτητα αφού περιορίζονται οι εγγραφές βάσει ημερομηνίας. Στο orders θέλουμε και το orders.custkey το οποιο χρησιμοποιείται για join, επομένως θα αποτελέσει δεύτερο κλειδί του ευρετηρίου. Στο shipdate χρησιμοποιούμε το price από αυτό το date range για το aggregate function sum, και συνεπώς θα το κανουμε include στο ευρετήριό μας για να το ανακτάμε γρήγορα.
- Ευρετήριο πάνω στο customers.cname αφού χρησιμοποιείται για group by και θέλουμε να αποφύγουμε το tablescan.

Ευρετήρια

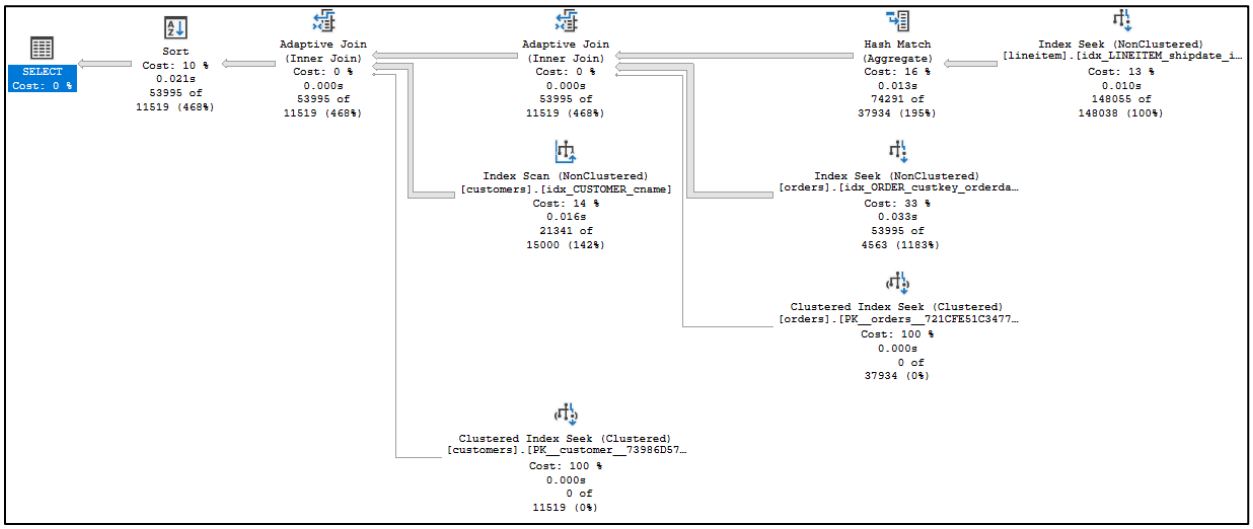
```
create index idx_LINEITEM_shipdate_iprice on lineitem(shipdate) include (price)
create index idx_ORDER_custkey_orderdate on orders(orderdate, custkey)
create index idx_CUSTOMER_cname on customers(cname)
```

Τελικά Στατιστικά

Row Num	Table	Scan Count	Logical Reads	Physical Reads	Read-Ahead Reads
	customers	1	446	3	470
	orders	1	967	3	985
	lineitem	1	407	1	403
	Worktable	0	0	0	0
	Worktable	0	0	0	0

	CPU	Elapsed
SQL Server parse and compile time:	00:00:00.000	00:00:00.011
SQL Server Execution Times:	00:00:00.016	00:00:00.395
Total	**00:00:00.016**	**00:00:00.406**

Τελικό Πλάνο Εκτέλεσης



Συμπεράσματα:

- Αρχικά παρατηρούμε πως σε όλους τους πίνακες τα logical reads που χρειάστηκαν μειώθηκαν εξαιρετικά. Συγκεκριμένα:
    - Customers: Παρατηρούμε πως το tablescan έγινε index scan και συνεπώς τα reads μειώθηκαν από 2.685 σε 446, πιθανώς επειδή κάθε ανάκτηση έγινε στοχευμένα και δεν χρειάστηκε να γινει parse όλος ο πίνακας.
    - Orders: Όπως προαναφέρθηκε, εφόσον περιορίζαμε το query μας βάσει το date range του orders.orderdate, το ευρετήριο αποτέλεσε κλειδί στο να μειωθούνε τα logical reads που έγιναν για την ανάκτηση του orders.custkey. Αυτό προφανώς έγινε καθώς περιορίζοντας τα orderdates, αυτόματα περιορίσαμε και τον αριθμό custkeys σε όσα ακριβώς χρειαζόμαστε να ελέγξουμε. Το αποτέλεσμα ήταν να μειωθούν τα logical reads από 16.925 σε 997.
    - Orderline: Επίσης ένα date range που περιόριζε σημαντικά τις εγγραφές που χρειαζόμαστε για να γίνει το sum aggregation. Είναι εμφανές πως το ευρετήριο με κλειδί το lineitem.shipdate περιέχει μόνο τα lineitem.price που μπορεί να χρειαστούμε για το επερώτημα, και συνεπώς μειώνουμε και το αντίκτυπο που θα είχε ένα aggregation σε έναν τεράστιο πίνακα με περιορισμούς. Το αποτέλεσμα ήταν να μειώσουμε τα 60.387 σε μόλις 407.
- Τα αποτελέσματα είναι αναμενόμενα καθώς στο τελικό πλάνο εκτέλεσης φαίνεται πως τα clustered index scans (πραγματικά tablescans στην προκειμένη περίπτωση) αντικαταστάθηκαν από index scans, χάριν στα ευρετήρια που δημιουργήσαμε.

Ζήτημα 2ο

Query

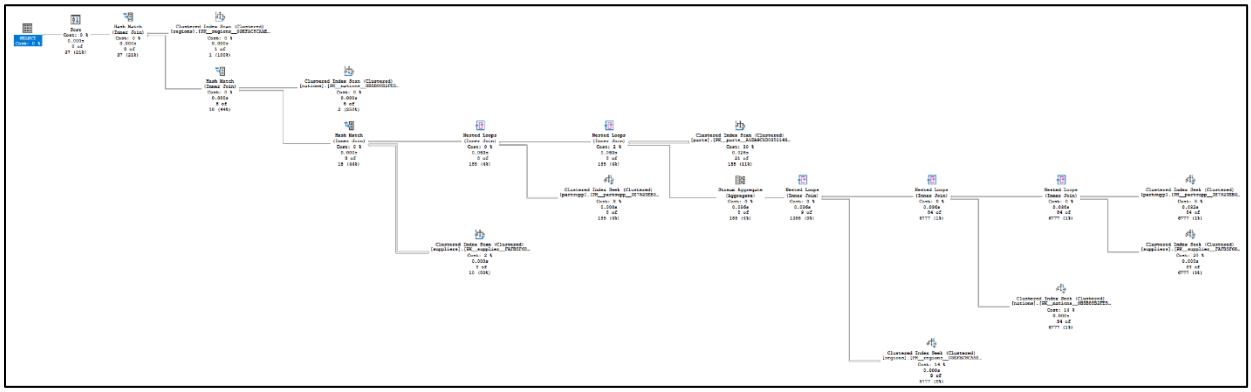
```
select  s_acctbal,sname, nation, parts.partkey, manufacturer, sphone, s_comment
   from parts, suppliers,partsupp, nations,regions
where parts.partkey = partsupp.partkey and
      suppliers.suppkey = partsupp.suppkey and
      suppliers.nationkey=nations.nationkey and
      nations.regionkey=regions.regionkey 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.suppkey = partsupp.suppkey and
                           suppliers.nationkey=nations.nationkey and
                           nations.regionkey=regions.regionkey and
                           region='EUROPE')
order by s_acctbal desc, nation, sname, parts.partkey;
```

Αρχικά Στατιστικά

Row Num	Table	Scan Count	Logical Reads	Physical Reads	Read-Ahead Reads
	parts	1	2,795	2	2,798
	suppliers	1	408	1	168
	nations	1	170	1	0
	regions	1	170	1	0
	partsupp	29	108	29	0

	CPU	Elapsed
SQL Server parse and compile time:	00:00:00.031	00:00:00.035
SQL Server Execution Times:	00:00:00.000	00:00:00.065
Total	**00:00:00.031**	**00:00:00.100**

Αρχικό Πλάνο Εκτέλεσης



Παρατηρήσεις:

- Πιθανώς το query να μπορεί να περιοριστεί σημαντικά μέσω των parts.psize, parts.ptype και regions.region. Όμως ο πίνακας regions περιέχει μόλις 5 εγγραφές και ένα ευρετήριο πάνω σε αυτόν θα ήταν πιθανώς ανούσιο και η δημιουργία του θα κόστιζε περισσότερο από το όφελος που θα είχαμε.
- Το πλάνο εκτέλεσης είναι εκτενές με αρκετά clustered index scans (tablescans), παρ' όλα αυτά θα επικεντρωθούμε σε αυτό που απαιτεί τα περισσότερα logical reads, αυτό του πίνακα parts, στα attributes που προαναφέρθηκαν.

Ιδέες:

- Περιορισμός των εγγραφών που διαβάζονται από τον πίνακα parts με ένα ευρετήριο πάνω στα parts.ptype και parts.psize, που έχουν πολύ συγκεκριμένη τιμή στο query και συνεπώς σίγουρα αποτελούν κόφτη στις εγγραφές που χρειαζόμαστε. Παρατηρούμε επίσης πως χρειαζόμαστε το parts.manufacturer, το οποίο θα το κάνουμε include στο ευρετήριο.
- Πειραματισμός με πίνακες πάνω στους πίνακες suppliers και regions (πάρα τους ενδοιασμούς) για την πιθανή επίτευξη καλύτερων αποτελεσμάτων.

Ευρετήρια:

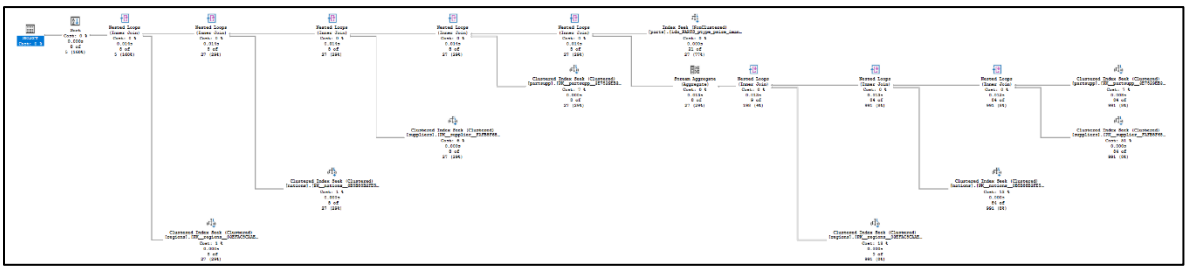
```
create index idx_PARTS_ptype_psize_imanufacturer on parts(ptype, psize) include (manufacturer)
```

Τελικά Στατιστικά

Row Num	Table	Scan Count	Logical Reads	Physical Reads	Read-Ahead Reads
	suppliers	0	272	1	168
	regions	0	184	1	0
	nations	0	184	1	0
	partsupp	29	108	29	0
	parts	1	3	3	0

	CPU	Elapsed
SQL Server parse and compile time:	00:00:00.000	00:00:00.000
SQL Server Execution Times:	00:00:00.000	00:00:00.052
Total	**00:00:00.000**	**00:00:00.052**

Τελικό Πλάνο Εκτέλεσης



Συμπεράσματα:

- Οι πειραματισμοί με ευρετήρια πάνω στους πίνακες suppliers και regions δεν οδήγησε σε βελτίωση και συνεπώς τα ευρετήρια αφαιρέθηκαν.
- Η δημιουργία του ευρετηρίου πάνω στον πίνακα parts με κλειδιά parts.ptype, parts.psize οδήγησε στην μείωση των logical reads πάνω στο parts από 2.795 σε μόλις 3. Αυτό είναι αναμενόμενο καθώς περιορίσαμε σημαντικά το τι είδους εγγραφές χρειαζόμαστε.
- Παρατηρήθηκε επίσης μείωση των logical reads στους πίνακες suppliers από 408 σε 272.
- Παρ' όλα αυτά, τα logical reads των nations και regions αυξήθηκαν ελάχιστα από 170 σε 184. Αυτό μπορεί να οφείλεται στο ότι αφού το ευρετήριο μας δεν περιέχει το κλειδί parts.partkey και να χρειάστηκαν μερικά ακόμα scans για να βρεθεί.

Ζήτημα 3ο

Query

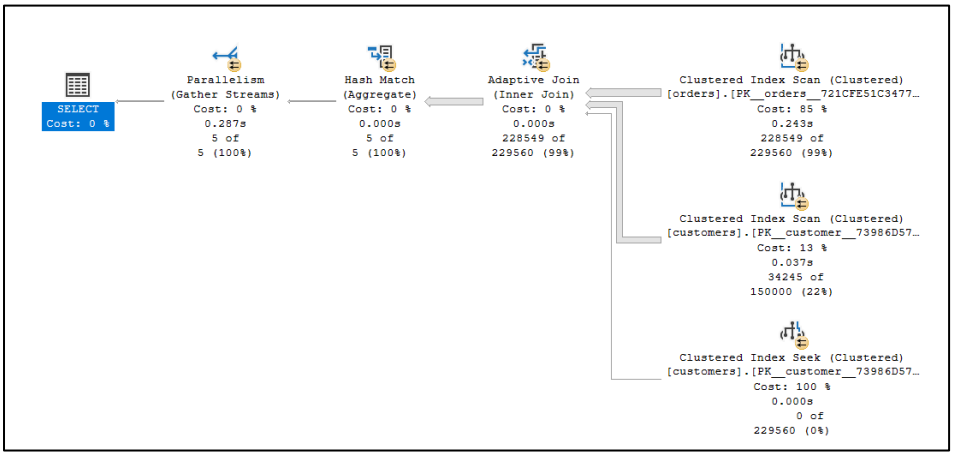
```
select market_segment, sum(totalprice)
   from customers, orders
  where customers.custkey=orders.custkey and YEAR(orderdate) = 1996
  group by market_segment
```

Αρχικά Στατιστικά

Row Num	Table	Scan Count	Logical Reads	Physical Reads	Read-Ahead Reads
	orders	13	16,961	1	16,729
	customers	13	2,685	2	2,562

	CPU	Elapsed
SQL Server parse and compile time:	00:00:00.000	00:00:00.000
SQL Server Execution Times:	00:00:00.000	00:00:00.309
Total	**00:00:00.000**	**00:00:00.309**

Αρχικό Πλάνο Εκτέλεσης



Παρατηρήσεις:

- Το υποερώτημα επιβαρύνεται σημαντικά σε logical reads στους πίνακες orders και customers, κάτι εμφανές από τα tablescans που γίνονται στο πλάνο εκτέλεσης.
- Χρησιμοποιείται η συνάρτηση YEAR() χωρίς ευρετήριο.

Ιδέες:

- Η YEAR() είναι κάτι που θα μπορούσαμε να μετατρέψουμε σε date range σε συνδυασμό με ευρετήριο ώστε να επωφεληθούμε από το πόσες εγγραφές θα κρατηθούν εν τέλει βάσει περιορισμών. Άρα ένα ευρετήριο πάνω στον πίνακα orders με κλειδιά τα orderdate, custkey που περιλαμβάνει το totalprice το οποίο χρησιμοποιείται για το sum aggregation.
- Ευρετήριο στον πίνακα orders πάνω στο custkey (υπάρχει ήδη clustered) που να περιλαμβάνει όμως το market_segment με το οποίο κάνουμε group by.

Ευρετήρια:

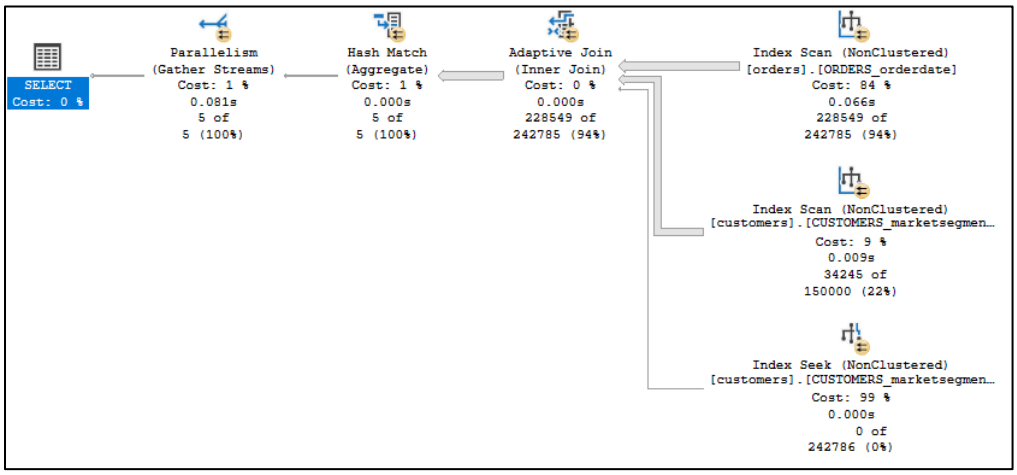
```
create index CUSTOMERS_marketsegment on customers(custkey) include (market_segment)
create index ORDERS_orderdate on orders(orderdate, custkey) include (totalprice)
```

Στατιστικά μόνο με Ευρετήρια

Row Num	Table	Scan Count	Logical Reads	Physical Reads	Read-Ahead Reads
	orders	13	4,146	1	4,176
	customers	13	1,297	1	432

	CPU	Elapsed
SQL Server parse and compile time:	00:00:00.000	00:00:00.006
SQL Server Execution Times:	00:00:00.016	00:00:00.104
Total	**00:00:00.016**	**00:00:00.110**

Πλάνο Εκτέλεσης μόνο με Ευρετήρια



Συμπεράσματα μόνο με ευρετήρια:

- Τα logical reads βελτιώθηκαν σε έναν σημαντικό βαθμό. Παρ' όλα αυτά τα νούμερα είναι ακόμα υψηλά. Συγκεκριμένα στον orders από 16.961 σε 4.146 και στον customers από 2.685 σε 1.297.
- Μήπως μπορούμε να γράψουμε το ερώτημα καλύτερα, λαμβάνοντας υπόψιν την YEAR() και το κόστος της;

New Query

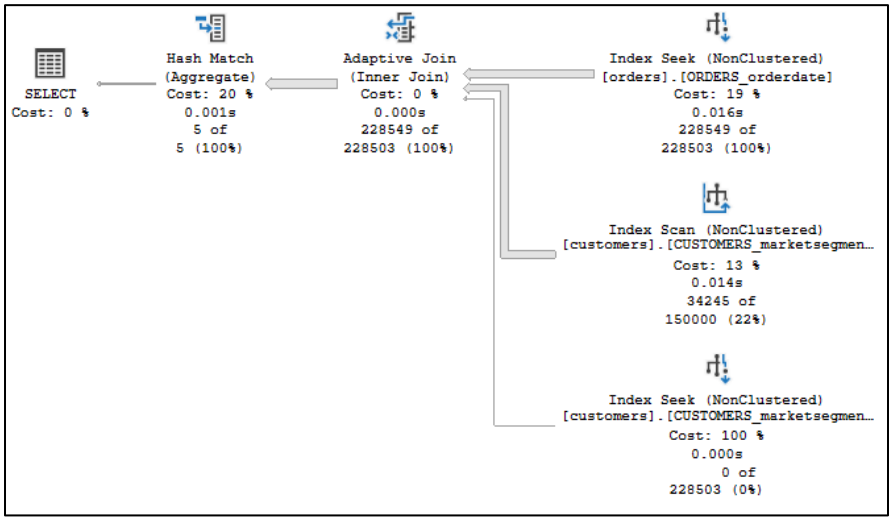
```
SELECT market_segment, SUM(totalprice)
FROM customers
JOIN (
    SELECT custkey, totalprice
    FROM orders
    WHERE orderdate >= '1996-01-01' AND orderdate < '1997-01-01'
) orders ON customers.custkey = orders.custkey
GROUP BY market_segment
```

Στατιστικά με Ευρετήρια + New Query

Row Num	Table	Scan Count	Logical Reads	Physical Reads	Read-Ahead Reads
	orders	1	627	3	632
	customers	1	434	1	432

	CPU	Elapsed
SQL Server parse and compile time:	00:00:00.000	00:00:00.000
SQL Server Execution Times:	00:00:00.000	00:00:00.181
Total	**00:00:00.000**	**00:00:00.181**

Πλάνο Εκτέλεσης με Ευρετήρια + New Query



Συμπεράσματα με ευρετήρια + new query:

- Παρατηρούμε πως τα logical reads έπεσαν ακόμα περισσότερο. Συγκεκριμένα στην orders από 4.146 σε 627 και στην customers από 1.297 σε 434. Τα αποτελέσματα αυτά είναι ικανοποιητικά και αποτελούν απόδειξη πως η επανεγγραφή του query, παρ' ολο που δεν φαίνεται στο πλάνο εκτέλεσης (*) και στο query cost relative to the batch (διήρκεσαν το ίδιο, 50/50%).
- Στο νέο query ο πίνακας orders με τον οποίον γίνεται join είναι σημαντικά μικρότερος αφού περιορίζεται βάσει orderdate, ενός date range. *Αυτό φαίνεται και στο γεγονός ότι πάνω στον πίνακα orders γίνεται index seek αντί για index scan.

Ζήτημα 4ο

Query A

```
select distinct partkey, pname
from parts
WHERE brand='Origin' OR manufacturer='Domkapa'
```

Query B

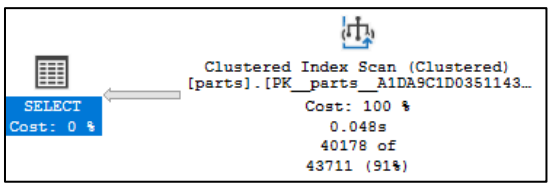
```
select partkey, pname
   from parts
where brand='Origin'

intersect

select partkey, pname
    from parts
where manufacturer='Domkapa'
```

Στατιστικά Query A – No Index No Rewrite

Row Num	Table	Scan Count	Logical Reads	Physical Reads	Read-Ahead Reads
	parts	1	2,795	2	2,798

	CPU	Elapsed
SQL Server parse and compile time:	00:00:00.000	00:00:00.000
SQL Server Execution Times:	00:00:00.000	00:00:00.245
Total	**00:00:00.000**	**00:00:00.245**

Πλάνο Εκτέλεσης Query A – No Index No Rewrite

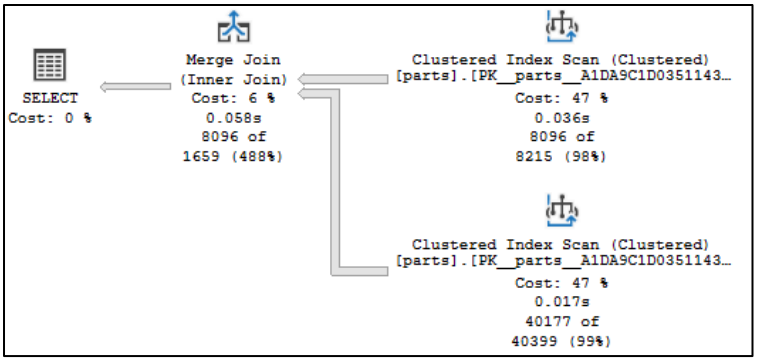


Στατιστικά Query B – No Index No Rewrite

Row Num	Table	Scan Count	Logical Reads	Physical Reads	Read-Ahead Reads
	parts	2	5,590	2	2,798

	CPU	Elapsed
SQL Server parse and compile time:	00:00:00.000	00:00:00.000
SQL Server Execution Times:	00:00:00.000	00:00:00.208
Total	**00:00:00.000**	**00:00:00.208**

Πλάνο Εκτέλεσης Query B – No Index No Rewrite



Ευρετήρια που Προτάθηκαν

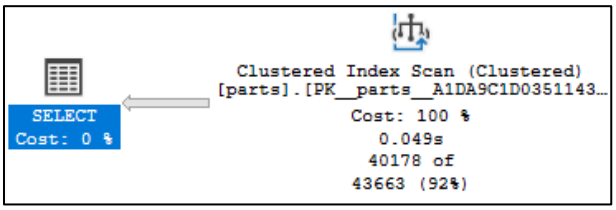
```
create index Q4_idx1 on parts(brand) include (partkey, pname)
create index Q4_idx2 on parts(manufacturer) include (partkey, pname)
```

Στατιστικά Query A – Index 1 No Rewrite

Row Num	Table	Scan Count	Logical Reads	Physical Reads	Read-Ahead Reads
	parts	1	2,795	2	2,798

	CPU	Elapsed
SQL Server parse and compile time:	00:00:00.000	00:00:00.002
SQL Server Execution Times:	00:00:00.000	00:00:00.224
Total	**00:00:00.000**	**00:00:00.226**

Πλάνο Εκτέλεσης Query A – Index 1 No Rewrite



Παρατηρήσεις:

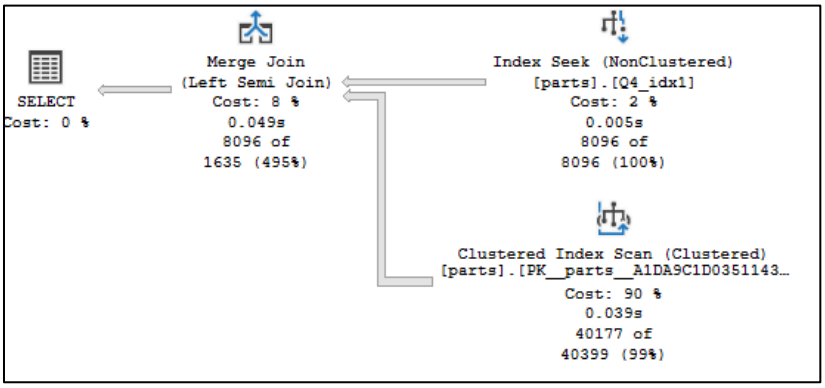
- Είναι εμφανές πως δεν υπάρχει απολύτως καμία διαφορά με την προσθήκη του ευρετηρίου 1. Τα logical reads παραμένουν το ίδιο και το πλάνο εκτέλεσης δείχνει πως έγινε clustered index scan (tablescan) στον πίνακα, άρα δεν χρησιμοποιήθηκε.

Στατιστικά Query B – Index 1 No Rewrite

Row Num	Table	Scan Count	Logical Reads	Physical Reads	Read-Ahead Reads
	parts	2	2,856	6	2,854

	CPU	Elapsed
SQL Server parse and compile time:	00:00:00.000	00:00:00.004
SQL Server Execution Times:	00:00:00.000	00:00:00.205
Total	**00:00:00.000**	**00:00:00.209**

Πλάνο Εκτέλεσης Query B – Index 1 No Rewrite



Παρατηρήσεις:

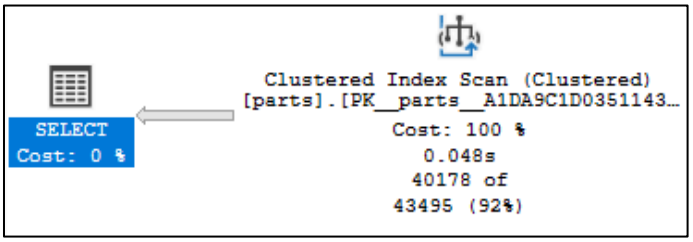
- Το ευρετήριο βοήθησε το query B και αυτό φαίνεται επειδή τα logical reads μειώθηκαν από 5.590 σε 2.856. Επίσης στον πίνακα parts χρησιμοποιήθηκε το ευρετήριο και αντί για index scan έγινε index seek, σε ένα από τα δυο components του join (parts.brand).

Στατιστικά Query A – Index 2 No Rewrite

Row Num	Table	Scan Count	Logical Reads	Physical Reads	Read-Ahead Reads
	parts	1	2,795	2	2,798

	CPU	Elapsed
SQL Server parse and compile time:	00:00:00.000	00:00:00.000
SQL Server Execution Times:	00:00:00.000	00:00:00.245
Total	**00:00:00.000**	**00:00:00.245**

Πλάνο Εκτέλεσης Query A – Index 2 No Rewrite



Παρατηρήσεις:

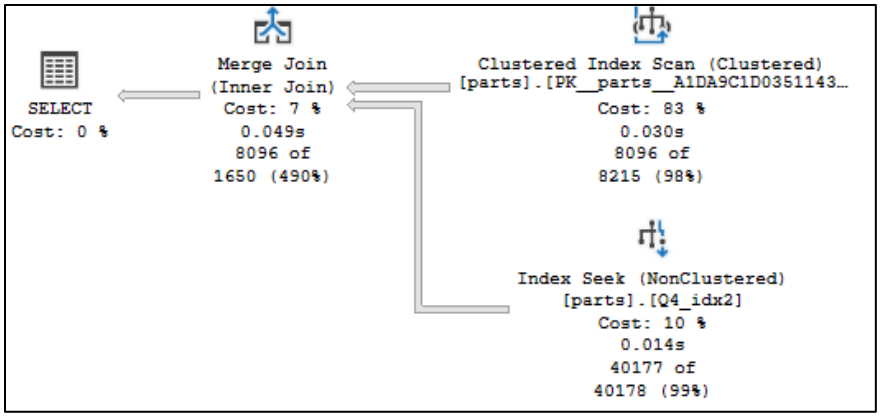
- Είναι εμφανές πως δεν υπάρχει απολύτως καμία διαφορά με την προσθήκη του ευρετηρίου 2. Τα logical reads παραμένουν το ίδιο και το πλάνο εκτέλεσης δείχνει πως έγινε clustered index scan (tablescan) στον πίνακα, άρα δεν χρησιμοποιήθηκε.

Στατιστικά Query B – Index 2 No Rewrite

Row Num	Table	Scan Count	Logical Reads	Physical Reads	Read-Ahead Reads
	parts	2	3,079	4	3,092

	CPU	Elapsed
SQL Server parse and compile time:	00:00:00.000	00:00:00.000
SQL Server Execution Times:	00:00:00.000	00:00:00.196
Total	**00:00:00.000**	**00:00:00.196**

Πλάνο Εκτέλεσης Query B – Index 2 No Rewrite



Παρατηρήσεις:

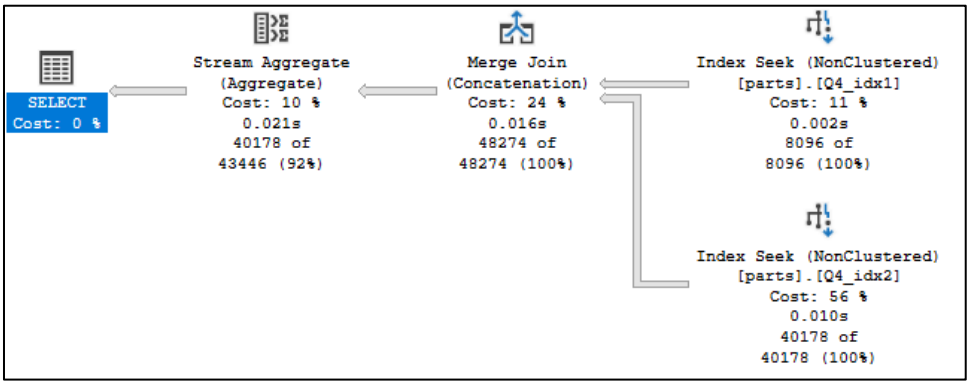
- Το ευρετήριο βοήθησε το query B και αυτό φαίνεται επειδή τα logical reads μειώθηκαν από 5.590 σε 3.079. Επίσης στον πίνακα parts χρησιμοποιήθηκε το ευρετήριο και αντί για index scan έγινε index seek, σε ένα από τα δυο components του join (parts.manufacturer).

Στατιστικά Query A – Index 1 & 2 No Rewrite

Row Num	Table	Scan Count	Logical Reads	Physical Reads	Read-Ahead Reads
	parts	2	346	7	350

	CPU	Elapsed
SQL Server parse and compile time:	00:00:00.000	00:00:00.000
SQL Server Execution Times:	00:00:00.016	00:00:00.216
Total	**00:00:00.016**	**00:00:00.216**

Πλάνο Εκτέλεσης Query A – Index 1 & 2 No Rewrite



Παρατηρήσεις:

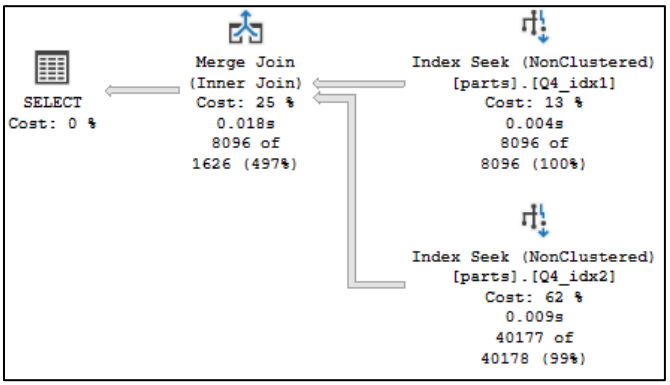
- Παρ' όλο που ξεχωριστά τα ευρετήρια δεν επωφελείσαν το query A, ο συνδυασμός τους φαίνεται να είχε σημαντικό impact καθώς τα logical reads μειώθηκαν από 2.975 σε 346. Αυτό είναι εμφανές και στο πλάνο εκτέλεσης, καθώς αντί για ένα index scan γίνονται δύο index seeks που προφανώς περιέχουν λιγότερες εγγραφές, πιο στοχευμένα.

Στατιστικά Query B – Index 1 & 2 No Rewrite

Row Num	Table	Scan Count	Logical Reads	Physical Reads	Read-Ahead Reads
	parts	2	345	6	350

	CPU	Elapsed
SQL Server parse and compile time:	00:00:00.000	00:00:00.000
SQL Server Execution Times:	00:00:00.000	00:00:00.180
Total	**00:00:00.000**	**00:00:00.180**

Πλάνο Εκτέλεσης Query B – Index 1 & 2 No Rewrite



Παρατηρήσεις:

- Όπως ήταν αναμενόμενο από τα προηγούμενα αποτελέσματα, τα δύο ευρετήρια μαζί βελτίωσαν ακόμα περισσότερο τα logical reads του query B, μειώνοντας τα από 5.590 σε 2.856 (το ελάχιστο που πετύχαμε με το Index 1 μόνο) σε 345. Τώρα φαίνεται πως και για το parts.brand και για το parts.manufacturer χρησιμοποιούντα τα indexes που δημιουργήσαμε.

Συμπεράσματα:

- Τα δύο indexes συνδυαστικά βελτιώνουν τις αποδόσεις των επερωτήσεων σημαντικά και συνεπώς αξιοποιούνται και θα κρατηθούν.
- Ο λόγος που χρειαζόμαστε δύο indexes ξεχωριστά για brand και manufacturer αντί για ένα συνδυαστικό είναι ότι μπορεί ο πίνακας να έχει πολλά δεδομένα για ένα από τα δύο, και λίγα από το άλλο, και συνεπώς να μην επωφεληθούμε από το συνδυαστικό ευρετήριο.
- Θα εξετάσουμε βελτιώσεις πάνω στα queries.

Query A – Index 1 & 2 Rewritten

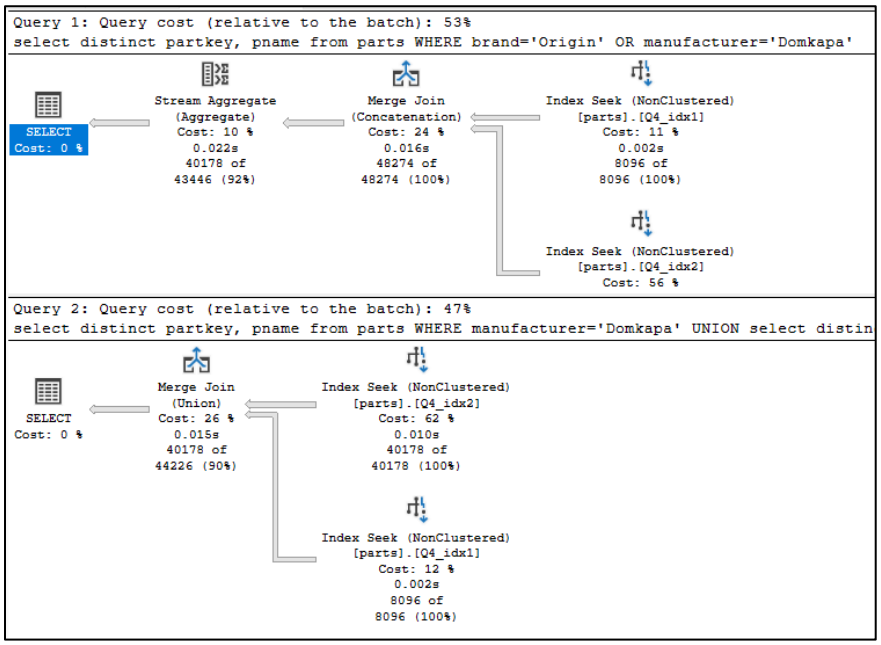
```
select partkey, pname
from parts
where manufacturer='Domkapa'
union
select partkey, pname
from parts
where brand='Origin'
```

Στατιστικά Query A – Index 1 & 2 Rewritten

Row Num	Table	Scan Count	Logical Reads	Physical Reads	Read-Ahead Reads
	parts	2	346	7	350

	CPU	Elapsed
SQL Server parse and compile time:	00:00:00.000	00:00:00.000
SQL Server Execution Times:	00:00:00.000	00:00:00.234
Total	**00:00:00.000**	**00:00:00.234**

Πλάνο Εκτέλεσης Query A – Index 1 & 2 Rewritten



Παρατηρήσεις & Συμπεράσματα:

- Αλλάξαμε το OR με UNION, καθώς το UNION αφαιρεί τα διπλότυπα, και γλυτώνουμε το distinct που απαιτεί έξτρα δουλειά.
- Παρατηρούμε πως ενώ τα logical reads δεν βελτιώθηκαν, στο πλάνο εκτέλεσης γλυτώσαμε ένα aggregation.

Query B – Index 1 & 2 Rewritten

Στατιστικά Query B – Index 1 & 2 Rewritten

Πλάνο Εκτέλεσης Query B – Index 1 & 2 Rewritten

Παρατηρήσεις & Συμπεράσματα:

-