

ΣΥΣΤΗΜΑΤΑ ΔΙΑΧΕΙΡΗΣΗΣ & ΑΝΑΛΥΣΗΣ ΔΕΔΟΜΕΝΩΝ

1η Εργασία

Αλβιόνα Μάντσο
3200098



Σχετικά με όσα ακολουθούν, σημειώνονται τα εξής:

1. Για την επίδειξη σε μορφή πινάκων των στατιστικών στοιχείων που αφορούν κάθε επερώτημα πριν και μετά από την κατασκευή των ευρετηρίων που προτείνονται έχει χρησιμοποιηθεί online parser: [StatisticsParser](#)
2. Δεν έχει γίνει ρητή ενημέρωση των στατιστικών που διατηρούνται (μέσω της εντολής `update statistics`).
3. Η γενική μεθοδολογία που ακολουθείται για την κατασκευή των ευρετηρίων παρακάτω είναι ότι τα ευρετήρια ορίζονται επί των γνωρισμάτων του πίνακα που συμμετέχουν στον έλεγχο κάποιας συνθήκης (στο `where`) ή στη λίστα των `group by` και `order by`, ενώ τυχόν άλλα γνωρίσματα που εμφανίζονται στο επερώτημα συμπεριλαμβάνονται απλώς (μέσω της εντολής `include`) για την επίτευξη ευρετηρίου επικάλυψης (`covering index`). Πέραν αυτής της γενικής ιδέας, πρόσθετες ή πιο εξειδικευμένες -ως προς το εκάστοτε επερώτημα- παρατηρήσεις θα καταγράφονται και θα επεξηγούνται στα αντίστοιχα σημεία.
4. Τα ευρετήρια ονοματοδοτούνται με την ακόλουθη μορφή:
`attr1_attr2_..._attrkTableName ↔ create index on TableName(attr1, attr2, ... ,attrk) include (...)`
5. Οι βελτιστοποιήσεις και οι συγκρίσεις που γίνονται βασίζονται στην παρατήρηση των πλάνων εκτέλεσης ενώ από τα στατιστικά λαμβάνουν υπόψιν κυρίως τα `logical reads`. Το στατιστικό του χρόνου (`time`) παρατίθεται χωρίς να λαμβάνεται ιδιαίτερα υπόψιν για την κατεύθυνση βελτιστοποίησης των επερωτημάτων και στη συνέχεια την τεκμηρίωση των προτάσεων, και αυτό γιατί αποτελεί παράγοντα που μεταβάλλεται από εκτέλεση σε εκτέλεση. Ωστόσο σε ενδεχόμενη ακραία τιμή του χρόνου δίνεται προσοχή και σε αυτόν.

Ζήτημα 1^ο

Για το επερώτημα όπως δίνεται κατασκευάζεται το ακόλουθο πλάνο εκτέλεσης (actual execution plan) όπου φαίνεται ότι για τους πίνακες customers, orders και lineitem γίνεται Clustered Index Scan (αντίστοιχη λειτουργία με το Table Scan, απλώς αξιοποιείται το ήδη υπάρχον ευρετήριο στα κλειδιά PK_customers..., PK_orders... και PK_lineitem..., αντίστοιχα):

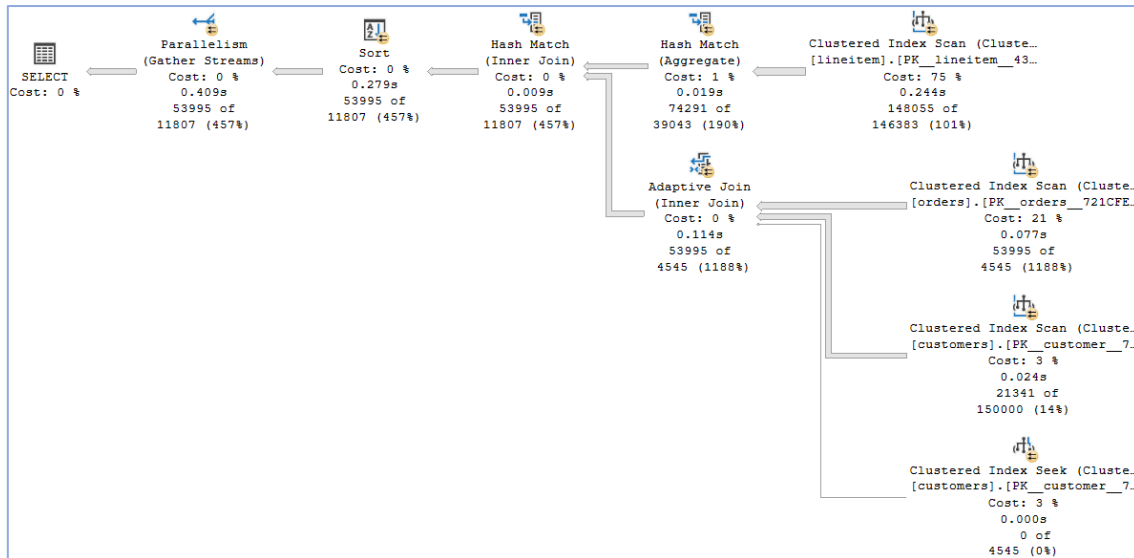


Table	Scan Count	Logical Reads	Physical Reads
customers	7	2,685	2
lineitem	7	60,387	1
orders	7	16,961	1
Worktable	0	0	0
Total	21	80,033	4

	CPU	Elapsed
SQL Server parse and compile time:	00:00:00.000	00:00:00.000
SQL Server Execution Times:	00:00:00.359	00:00:00.697
Total	00:00:00.359	00:00:00.697

Παρατηρούμε ότι οι πράξεις Clustered Index Scan (ουσιαστικά Table Scan) είναι πολύ κοστοβόρες και χρονοβόρες οπότε θα προσπαθήσουμε να τις αποφύγουμε μέσω της δημιουργίας κατάλληλων ευρετηρίων στους τρεις πίνακες.

```
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
```

```
create index orderdateOrders on
orders(orderdate, custkey)
```

```
create index shipdateLineitem on
lineitem(shipdate) include (price)
```

```
create index cnameCustomers on
customers(cname)
```

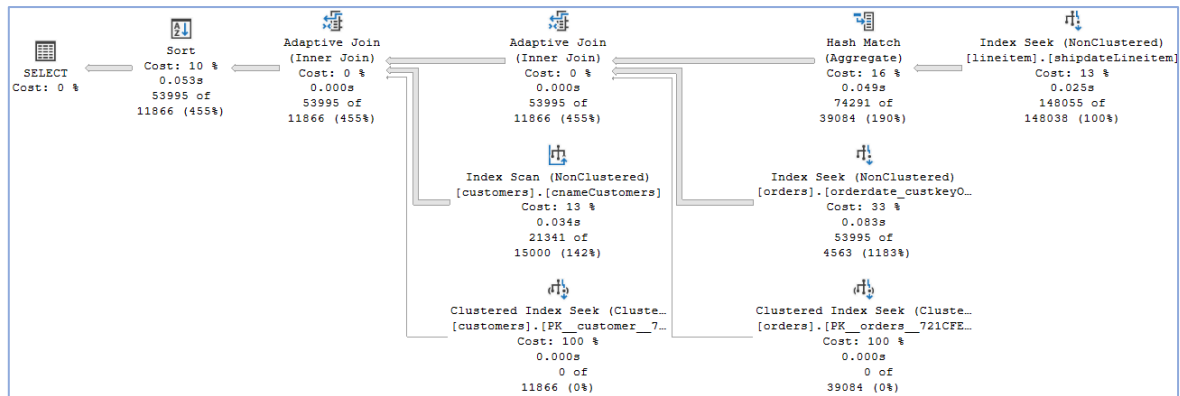


Table	Scan Count	Logical Reads	Physical Reads
customers	1	440	3
lineitem	1	407	1
orders	1	964	1
Worktable	0	0	0
Total	3	1,811	5

	CPU	Elapsed
SQL Server parse and compile time:	00:00:00.000	00:00:00.000
SQL Server Execution Times:	00:00:00.140	00:00:00.570
Total	00:00:00.140	00:00:00.570

Παρατηρούμε ότι και τα τρία ευρετήρια που κατασκευάστηκαν αξιοποιούνται.

Ως προς τον πίνακα lineitem γίνεται Index Seek πάνω στο ευρετήριο shipdateLineitem για να περιοριστούν οι εγγραφές σε αυτές που ικανοποιούν τη συνθήκη shipdate between '1994-01-01' and '1994-02-28', ενώ λόγω της συμπερίληψης (include) του γνωρίσματος price δεν απαιτούνται πρόσθετες ανακτήσεις για τον πίνακα αυτό. Έτσι τα logical reads μειώνονται ραγδαία (60387→407).

Στη συνέχεια για τον πίνακα orders γίνεται Index Seek πάνω στο ευρετήριο orderdateOrders για να περιοριστούν οι εγγραφές σε αυτές που ικανοποιούν τη συνθήκη orderdate <= '1993-12-31' και να βοηθηθεί έπειτα ο έλεγχος της συνθήκης customers.custkey = orders.custkey της σύζευξης (το γνώρισμα custkey δεν είναι κλειδί στον πίνακα αυτό). Και εδώ τα logical reads μειώνονται πολύ (16961→964).

Τέλος, για τον πίνακα customers γίνεται πλέον Index Scan πάνω στο ευρετήριο cnameCustomers αντί Clustered Index Scan. Το ευρετήριο δεν χρησιμοποιείται για seek εφόσον δεν υπάρχει κάποια συνθήκη που αφορά το γνώρισμα cname αλλά παρ' όλα αυτά τα logical reads που αφορούν τον πίνακα customers μειώνονται σημαντικά (2865→440), εφόσον το ευρετήριο είναι πολύ μικρότερο από τον ίδιο τον πίνακα και άρα η ανάκτηση αυτού επιφέρει διάβασμα λιγότερων σελίδων (logical/physical reads).

Ζήτημα 2^ο


Η εκτέλεση του επερωτήματος όπως δίνεται οδηγεί στα ακόλουθα στατιστικά:

Table	Scan Count	Logical Reads	Physical Reads
nations	1	170	1
parts	1	2,795	2
partsupp	29	108	29
regions	1	170	1
suppliers	1	408	1
Worktable	0	0	0
Total	33	3,651	34

	CPU	Elapsed
SQL Server parse and compile time:	00:00:00.000	00:00:00.000
SQL Server Execution Times:	00:00:00.016	00:00:00.113
Total	00:00:00.016	00:00:00.113

Το πλάνο εκτέλεσης (actual execution plan) που κατασκευάζεται για το ερώτημα αυτό είναι ιδιαίτερα μεγάλο οπότε δεν συμπεριλαμβάνεται (δεν θα ήταν ευανάγνωστο). Ωστόσο παρατίθενται τα βασικά του σημεία:

Τα Clustered Index Scans (αντίστοιχη λειτουργία με τα Table Scans απλώς αξιοποιούνται τα ήδη υπάρχοντα ευρετήρια στα κλειδιά) που πραγματοποιούνται και που (συχνά) είναι τα πιο κοστοβόρα και χρονοβόρα είναι τα εξής:


Clustered Index Scan (Cluste... [regions].[PK_regions__03EF... Cost: 0 % 0.000s 1 of 1 (100%)

Εξαρχής δεν υπάρχει λόγος να γίνει προσπάθεια βελτίωσης εφόσον το σχετικό κόστος είναι 0%.


Και χωρίς την απεικόνιση του κόστους, ωστόσο, παρατηρούμε:

```
select count(*)  
from regions
```

	regionkey	region
1	0	AFRICA
2	1	AMERICA
3	2	ASIA
4	3	EUROPE
5	4	MIDDLE EAST

Επιστρέφονται μόλις 5 αποτελέσματα.

Άρα, η δημιουργία οποιουδήποτε ευρετηρίου θα ήταν εντελώς περιττή και δε θα αντιστάθμιζε καθόλου το κόστος δημιουργίας ενώ και η χρήση του μάλλον δεν θα ήταν συμφέρουσα (random I/Os).


Clustered Index Scan (Cluste... [nations].[PK_nations__0B5B... Cost: 0 % 0.000s 5 of 2 (250%)


Ομοίως, δεν υπάρχει λόγος να γίνει προσπάθεια βελτίωσης εφόσον το σχετικό κόστος είναι 0%.

Και χωρίς την απεικόνιση του κόστους, ωστόσο παρατηρούμε το εξής:

```
select count(*)  
from nations
```

Επιστρέφονται μόλις 25 αποτελέσματα.


Η δημιουργία οποιουδήποτε ευρετηρίου θα ήταν εντελώς περιττή και εδώ.


Clustered Index Scan (Cluste... [suppliers].[PK_supplier_F... Cost: 2 % 0.000s 8 of 10 (80%)

Ο πίνακας suppliers συμμετέχει στον έλεγχο των συνθηκών:

suppliers.supkey = partsupp.supkey και
suppliers.nationkey=nations.nationkey

(που αφορούν τις συζεύξεις). Ως εκ τούτου επιχειρήθηκε πειραματικά να κατασκευαστεί ευρετήριο επί των γνωρισμάτων που ελέγχονται στις συνθήκες αυτές, το οποίο ωστόσο δεν είχε καμία επίδραση. Όπως μπορεί να διαπιστωθεί από το επερώτημα, ένα αποδοτικό ευρετήριο θα έπρεπε να συμπεριλαμβάνει (include) όλα τα γνωρίσματα που εμφανίζονται στο επερώτημα (όχι μόνο στις συνθήκες του, αλλά και στη λίστα του order, στο select κλπ). Για το συγκεκριμένο επερώτημα αυτό θα σήμαινε το ευρετήριο να είναι απλώς ένα αντίγραφο του πίνακα, καθώς όλα τα γνωρίσματα (ακόμη και το s_comment) συμμετέχουν στο επερώτημα. Προφανώς αυτό δεν θα είχε νόημα και συνεπώς δεν προτείνεται τελικά κάποιο ευρετήριο ούτε σε αυτόν τον πίνακα.


Clustered Index Scan (Cluste... [parts].[PK_parts__A1DA9C1D... Cost: 31 % 0.051s 21 of 171 (12%)

(στην επόμενη σελίδα)

Clustered Index Scan (Cluste...
[parts].[PK_parts_A1DA9C1D...
Cost: 31 %
0.051s
21 of
171 (12%)

Το εξωτερικό υπο-επερώτημα περιλαμβάνει τον έλεγχο της εξής συνθήκης για τις εγγραφές του πίνακα parts: `psize = 31 and region = 'EUROPE'`

Συνεπώς μπορούμε να αποφύγουμε το Clustered Index Scan εάν κατασκευάσουμε το ευρετήριο: `create index ptype_psizeParts on parts(ptype,psize) include (manufacturer)`

Το ευρετήριο έχει κατασκευαστεί στα γνωρίσματα ptype, psize (με αυτή τη σειρά) ώστε πρωτίστως να περιοριστούν οι εγγραφές ως προς το ptype = 'LARGE PLATED TIN' (μόνο 1281 εγγραφές θα ικανοποιούν αυτή τη συνθήκη) και στη συνέχεια από αυτές να επιλεχθούν μόνον όσες ικανοποιούν και τη συνθήκη psize = 31. Ο λόγος που επιλέγεται αυτή η σειρά είναι ότι το γνώρισμα ptype έχει μεγαλύτερη επιλεξιμότητα (selectivity *) σε σχέση με το psize και άρα το ευρετήριο αυτό θα εξυπηρετεί εν γένει καλύτερα τέτοιου είδους επερωτήματα (όχι δηλ. μόνο για το συγκεκριμένο ptype και psize). Τέλος, έχει οριστεί έτσι ώστε να είναι ευρετήριο επικάλυψης για το επερώτημα, αφού συμπεριλαμβάνει (include) και το γνώρισμα manufacturer.

* selectivity:

<code>select count(distinct ptype)</code>	→ 150 / 200000	← <code>select count(*)</code> from parts
<code>from parts</code>		
<code>select count(distinct psize)</code>	→ 50 / 200000	
<code>from parts</code>		

Table	Scan Count	Logical Reads	Physical Reads
nations	0	184	1
parts	1	3	3
partsupp	29	108	29
regions	0	184	1
suppliers	0	272	1
Worktable	0	0	0
Total	30	751	35

	CPU	Elapsed
SQL Server parse and compile time:	00:00:00.000	00:00:00.000
SQL Server Execution Times:	00:00:00.000	00:00:00.074
Total	00:00:00.000	00:00:00.074

Παρατηρούμε μια ραγδαία μείωση των logical reads που αφορούν τον πίνακα parts (2795→3), ενώ το ευρετήριο αυτό φαίνεται πως οδήγησε τον optimizer να κάνει διαφορετικές επιλογές στο γενικό πλάνο εκτέλεσης και όχι μόνο τοπικά. Με μια ματιά στο πλάνο εκτέλεσης (που και πάλι δεν συμπεριλαμβάνεται εδώ) φαίνεται να γίνονται αλληπάλληλα Nested Loops (που αντιστοιχούν στις συζεύξεις-joins του εξωτερικού και εσωτερικού υπο-επερωτήματος) τα οποία τροφοδοτούνται από ενδιάμεσα αποτελέσματα που προκύπτουν από Clustered Index Seek (και όχι scan πλεον). Έτσι τα logical reads του πίνακα suppliers παρουσιάζουν μείωση (408→272) παρότι τα αντίστοιχα logical reads των nations και regions παρουσιάζουν μια μικρή αύξηση (αυτό μάλλον οφείλεται στο μικρό τους μέγεθος που καθιστά πιο συμφέρον ένα Scan).

Clustered Index Seek (Cluste...
[regions].[PK_regions__03EF...
Cost: 1 %
0.000s
8 of
27 (29%)

Clustered Index Seek (Cluste...
[nations].[PK_nations__0B5B...
Cost: 1 %
0.000s
8 of
27 (29%)

Clustered Index Seek (Cluste...
[suppliers].[PK_supplier_F...
Cost: 6 %
0.000s
8 of
27 (29%)

Index Seek (NonClustered)
[parts].[ptype_psizeParts]
Cost: 0 %
0.001s
21 of
27 (77%)

Ζήτηση 3^ο

Για το ερωτήμα όπως δίνεται από τον προγραμματιστή κατασκευάζεται το ακόλουθο πλάνο εκτέλεσης (actual execution plan) όπου φαίνεται ότι για τους πίνακες customers και orders γίνεται Clustered Index Scan (αντίστοιχη λειτουργία με το Table Scan, απλώς αξιοποιείται το ήδη υπάρχον ευρετήριο στα κλειδιά PK_customers_... και PK_orders_..., αντίστοιχα):

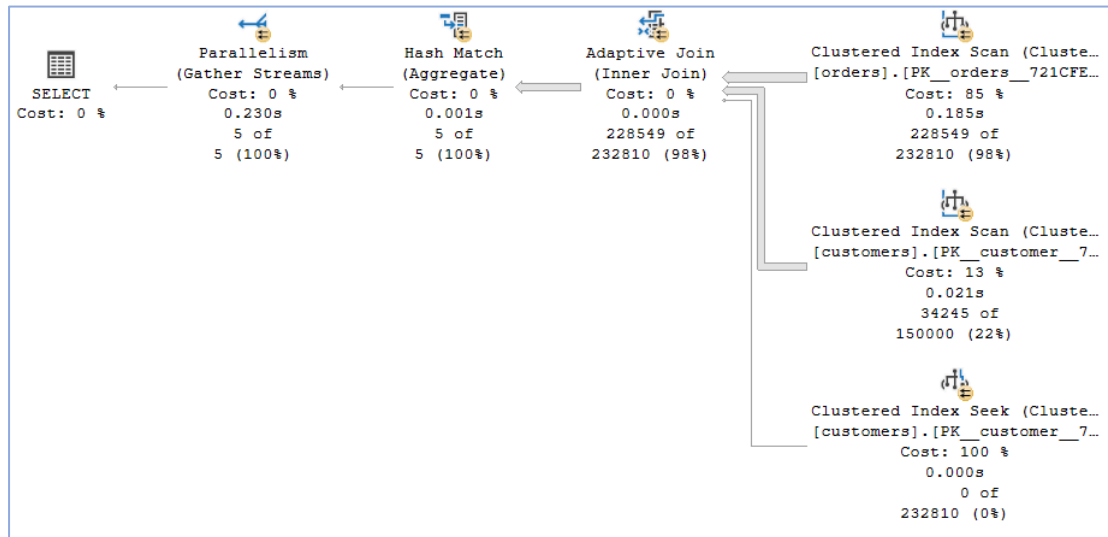


Table	Scan Count	Logical Reads	Physical Reads
customers	7	2,685	2
orders	7	16,961	1
Worktable	0	0	0
Total	14	19,646	3

	CPU	Elapsed
SQL Server parse and compile time:	00:00:00.000	00:00:00.000
SQL Server Execution Times:	00:00:00.530	00:00:00.266
Total	00:00:00.530	00:00:00.266

Λόγω της συνθήκης **YEAR**(orderdate) = 1996 προκύπτει ως πρόταση βελτίωσης η δημιουργία ευρετηρίου:

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

όπου οι στήλες (custkey, totalprice) συμπεριλαμβάνονται επειδή συμμετέχουν στο επερώτημα, ώστε το ευρετήριο να είναι επικάλυψης (covering index) και άρα επαρκές για την εκτέλεση του επερωτήματος χωρίς να χρειάζονται επιπλέον ανακτήσεις.

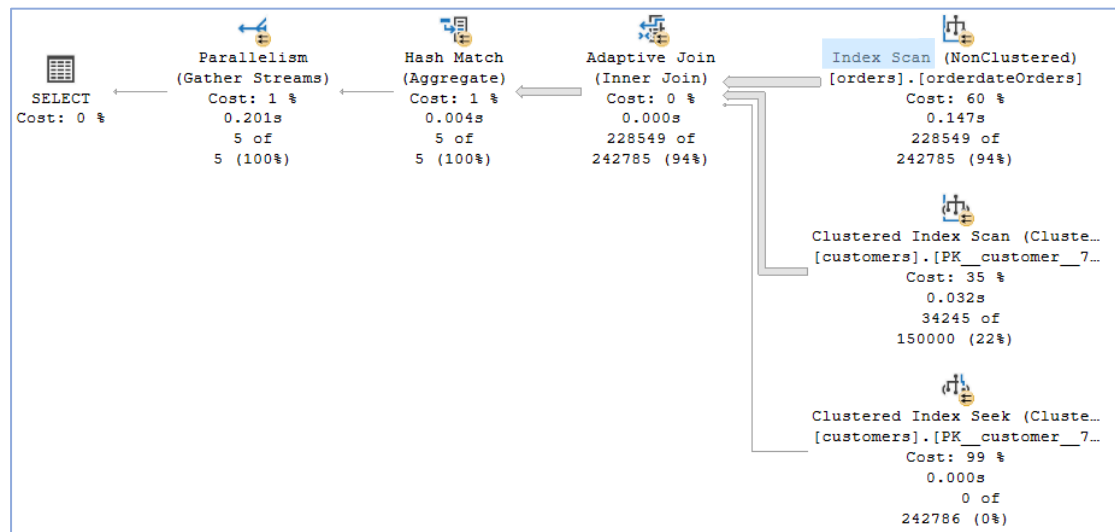


Table	Scan Count	Logical Reads	Physical Reads
customers	7	2,685	2
orders	7	4,139	1
Worktable	0	0	0
Total	14	6,824	3

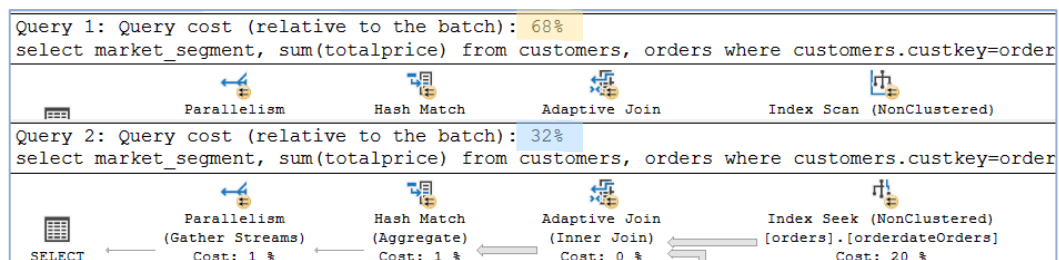
Παρατηρούμε ότι παρότι το ευρετήριο αυτό αξιοποιείται (γίνεται Index Scan) και κατά συνέπεια τα logical reads που αφορούν τον πίνακα orders μειώνονται σημαντικά (16,961 → 4,139), δεν γίνεται Index Seek όπως θα θέλαμε! Αυτό οφείλεται στην χρήση της συνάρτησης **YEAR()** που δεν επιτρέπει στον optimizer να χρησιμοποιήσει το ευρετήριο για seek.

Επαναδιατυπώνουμε λοιπόν το επερώτημα χωρίς τη χρήση της συνάρτησης YEAR():

```
select market_segment, sum(totalprice)
from customers, orders
where customers.custkey=orders.custkey and
orderdate between '1996-01-01' and '1996-12-31'
group by market_segment
```

Table	Scan Count	Logical Reads	Physical Reads
customers	7	2,685	2
orders	7	659	2
Worktable	0	0	0
Total	14	3,344	4

	CPU	Elapsed
SQL Server parse and compile time:	00:00:00.000	00:00:00.000
SQL Server Execution Times:	00:00:00.109	00:00:00.170
Total	00:00:00.109	00:00:00.170



Συγκριτικά για τα δύο ερωτήματα, παρατηρούμε ότι το δεύτερο (χωρίς τη χρήση της συνάρτησης YEAR()) επιφέρει πολύ λιγότερα logical reads (659) σε σχέση με το αρχικό (4139) και είναι πιο αποδοτικό όπως φαίνεται από τα ποσοστά σχετικού κόστους.

Τέλος μπορούν να βελτιωθούν τα logical reads στον πίνακα customers (που μέχρι στιγμής δεν έχουν μεταβληθεί) με τη δημιουργία του ευρετηρίου:

```
create index market_segmentCustomers on customers(market_segment)
include (custkey)
```

εφόσον το group by (που έμμεσα απαιτεί ταξινόμηση για να εκτελεστεί) γίνεται επί του γνωρίσματος market_segment του πίνακα customers. Συμπεριλαμβάνεται και το custkey που εμφανίζεται στο επερώτημα, ώστε να έχουμε ευρετήριο επικάλυψης (covering index).

Table	Scan Count	Logical Reads	Physical Reads
customers	1	433	3
orders	1	628	2
Worktable	0	0	0
Total	2	1,061	5

	CPU	Elapsed
SQL Server parse and compile time:	00:00:00.000	00:00:00.000
SQL Server Execution Times:	00:00:00.032	00:00:00.217
Total	00:00:00.032	00:00:00.217

Ζήτηση 4^ο

(1)

- Χωρίς τη χρήση ευρετηρίου:

1^ο επερώτημα

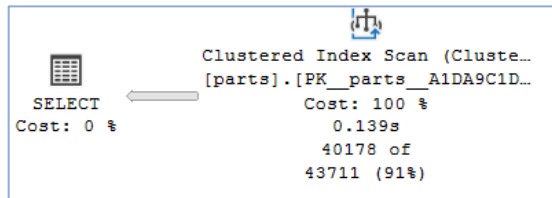


Table	Scan Count	Logical Reads	Physical Reads
parts	1	2,795	2
Total	1	2,795	2

CPU		Elapsed
SQL Server parse and compile time:		00:00:00.000
SQL Server Execution Times:		00:00:00.250
Total	00:00:00.031	00:00:00.250

2^ο επερώτημα

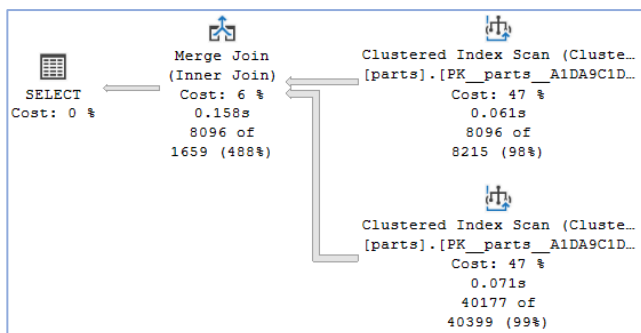


Table	Scan Count	Logical Reads	Physical Reads
parts	2	5,590	0
Total	2	5,590	0

CPU		Elapsed
SQL Server parse and compile time:		00:00:00.000
SQL Server Execution Times:		00:00:00.293
Total	00:00:00.063	00:00:00.293

- Με τη χρήση του ευρετηρίου Q4_idx1 (*on parts (brand) include (pname)*)

1^ο επερώτημα

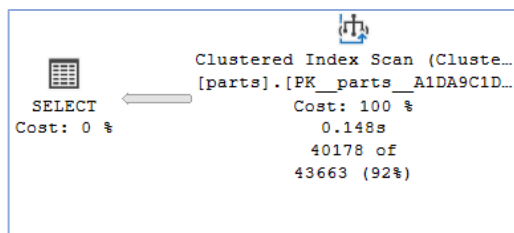


Table	Scan Count	Logical Reads	Physical Reads
parts	1	2,795	2
Total	1	2,795	2

CPU		Elapsed
SQL Server parse and compile time:		00:00:00.000
SQL Server Execution Times:		00:00:00.272
Total	00:00:00.016	00:00:00.272

Δεν υπάρχει κάποια αλλαγή. Το ευρετήριο δεν αξιοποιείται παρότι κατασκευάστηκε, εφόσον αφορά μόνο το γνώρισμα brand τη στιγμή που η συνθήκη του επερωτήματος περιέχει λογική διάζευξη (OR). Ελέγχονται, λοιπόν, όλες οι εγγραφές για την ικανοποίηση της (εξ' ου γίνεται Clustered Index Scan μέσω του υπάρχοντος ευρετηρίου στο κλειδί PK_parts_...).

2^ο επερώτημα

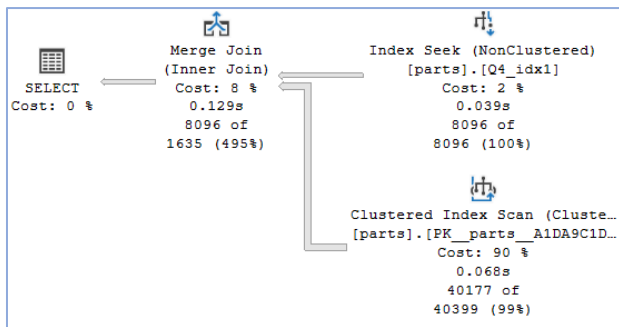


Table	Scan Count	Logical Reads	Physical Reads
parts	2	2,856	4
Total	2	2,856	4

	CPU	Elapsed
SQL Server parse and compile time:	00:00:00.000	00:00:00.000
SQL Server Execution Times:	00:00:00.047	00:00:00.205
Total	00:00:00.047	00:00:00.205

Το ευρετήριο αξιοποιείται για seek (γίνεται Index Seek) για το σκέλος του intersect που ελέγχει τη συνθήκη `where brand='Origin'` (εφόσον το ευρετήριο έχει κατασκευαστεί πάνω στο γνώρισμα brand). Έτσι, μειώνονται τα απαιτούμενα logical reads (5590→2856). Για το άλλο σκέλος γίνεται και πάλι Clustered Index Scan όπως πριν.

- Με τη χρήση του ευρετηρίου Q4_idx2 (`on parts(manufacturer) include (pname)`)

1^ο επερώτημα

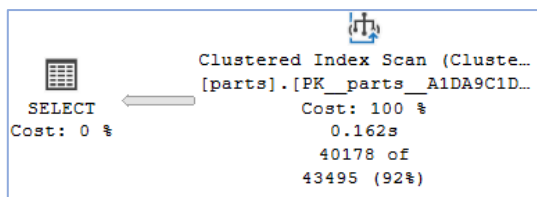


Table	Scan Count	Logical Reads	Physical Reads
parts	1	2,795	2
Total	1	2,795	2

	CPU	Elapsed
SQL Server parse and compile time:	00:00:00.000	00:00:00.000
SQL Server Execution Times:	00:00:00.015	00:00:00.234
Total	00:00:00.015	00:00:00.234

Δεν υπάρχει κάποια αλλαγή. Το ευρετήριο δεν αξιοποιείται παρότι κατασκευάστηκε, εφόσον αφορά μόνο το γνώρισμα manufacturer τη στιγμή που η συνθήκη του επερωτήματος περιέχει λογική διάζευξη (OR). Ελέγχονται, λοιπόν, όλες οι εγγραφές για την ικανοποίηση της (εξ' ου γίνεται Clustered Index Scan μέσω του υπάρχοντος ευρετηρίου στο κλειδί PK_parts...).

2^ο επερώτημα

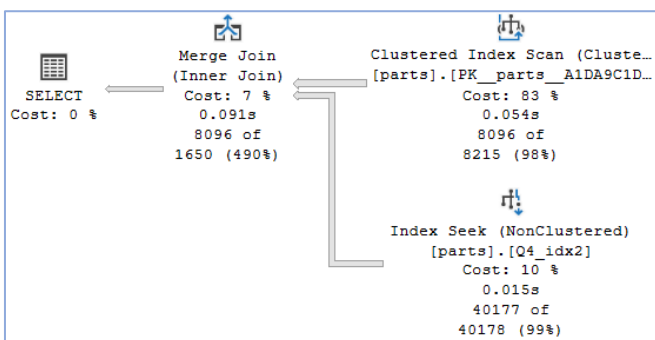


Table	Scan Count	Logical Reads	Physical Reads
parts	2	3,079	2
Total	2	3,079	2

	CPU	Elapsed
SQL Server parse and compile time:	00:00:00.000	00:00:00.000
SQL Server Execution Times:	00:00:00.016	00:00:00.226
Total	00:00:00.016	00:00:00.226

Το ευρετήριο αξιοποιείται για seek (γίνεται Index Seek) για το σκέλος του intersect που ελέγχει τη συνθήκη `where manufacturer='Domkapa'` (εφόσον το ευρετήριο έχει κατασκευαστεί πάνω στο γνώρισμα `manufacturer`). Έτσι, μειώνονται τα απαιτούμενα logical reads (5590→3079). Για το πρώτο σκέλος γίνεται και πάλι Clustered Index Scan όπως στο αρχικό ερώτημα.

Παρατήρηση: Το ευρετήριο `Q4_idx1` (στο γνώρισμα `brand`) μειώνει περισσότερο τα logical reads που απαιτούνται για το 2^ο επερώτημα (5590 →2856) σε σχέση με το `Q4_idx2` (στο γνώρισμα `manufacturer`) (όπου η μείωση είναι 5590 →3079). Αυτό είναι λογικό εφόσον υπάρχουν 8096 εγγραφές όπου `brand='Origin'` αλλά 40178 (δηλ. αρκετά περισσότερες) εγγραφές όπου `manufacturer='Domkapa'`. Άρα το `Q4_idx1` περιορίζει περισσότερο τα αποτελέσματα του σκέλους στο οποίο αξιοποιείται.

```
select count(*)
from parts
where brand='Origin'
```

—————> 8096

```
select count(*)
from parts
where manufacturer='Domkapa'
```

—————> 40178

(2) 1^ο επερώτημα

Προτείνεται ο ορισμός και των δύο ευρετηρίων `Q4_idx1`, `Q4_idx2`.

Επίσης εξετάζεται το ερώτημα επαναδιατυπωμένο με τη χρήση union ως εξής:

I.

Αρχικό ερώτημα

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

II.

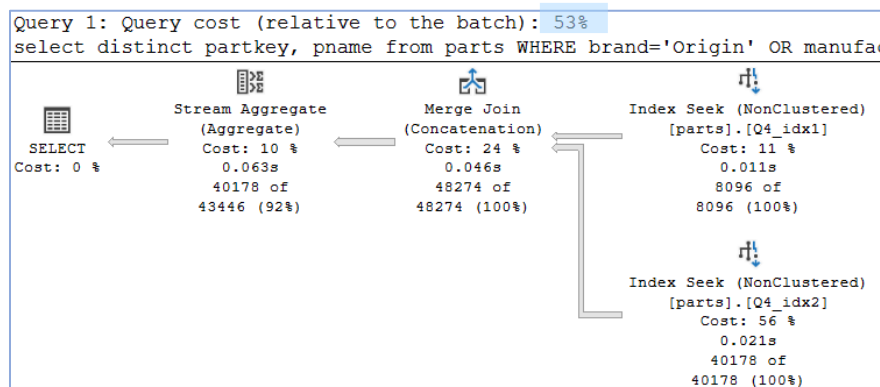
```
select partkey, pname
from parts
WHERE brand='Origin'

union

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

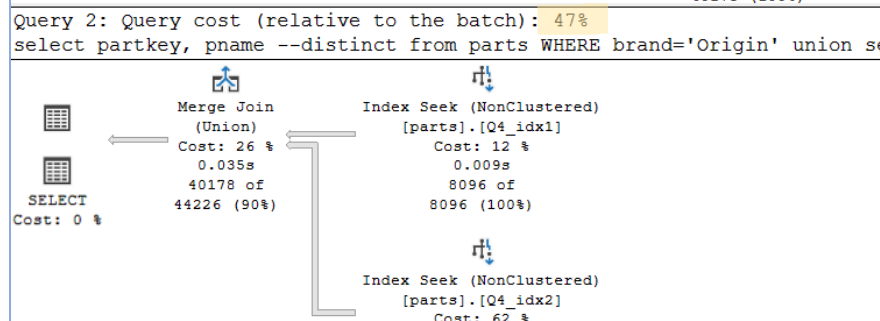
Με τα ευρετήρια `Q4_idx1` και `Q4_idx2`, τα συγκριτικά αποτελέσματα για τα δύο επερωτήματα είναι:

I. (or)



Scan Count	Logical Reads	Physical Reads
2	346	8
2	346	8

II. (union)



Scan Count	Logical Reads	Physical Reads
2	346	8
2	346	8

Παρατηρούμε ότι τώρα που έχουν κατασκευαστεί και τα δύο ευρετήρια (QA_idx1, Q4_idx2) αξιοποιούνται και τα δύο (ενώ πριν δεν χρησιμοποιήθηκε ούτε το ένα ούτε το άλλο). Και στα δύο επερωτήματα γίνεται πλέον Index Seek σε κάθε ένα από τα δύο ευρετήρια.

Συγκριτικά για τα δύο επερωτήματα, παρατηρούμε ότι επιφέρουν το ίδιο πλήθος logical reads (346) αλλά παρ' όλα αυτά το II (με τη χρήση union) είναι πιο αποδοτικό όπως φαίνεται από τα ποσοστά σχετικού κόστους.

2° επερώτημα

Προτείνεται ο ορισμός και των δύο ευρετηρίων Q4_idx1, Q4_idx2.
Επιπλέον κατασκευάζεται το ευρετήριο:

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

Επίσης εξετάζεται το ερώτημα επαναδιατυπωμένο με τη χρήση and ως εξής:

To intersect
αφαιρεί τα
διπλότυπα

I.

Αρχικό ερώτημα

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

intersect

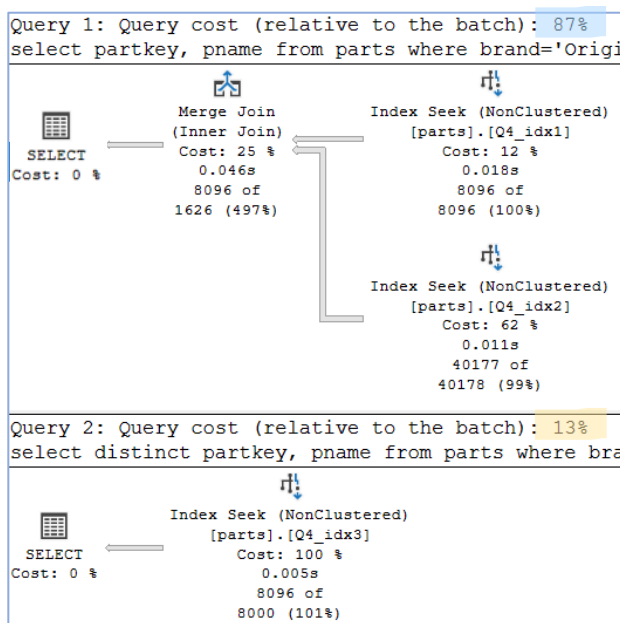
```
select partkey, pname
from parts
where manufacturer='Domkapa'
```

II.

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

Τυπικά, το ισοδύναμο επερώτημα
χρειάζεται distinct. Αλλά εφόσον το
partkey είναι μοναδικό (κλειδί) και
άρα το (partkey, pname) είναι
μοναδικό, αυτό είναι περιττό.

I. (intersect)



II. (and)

Table	Scan Count	Logical Reads	Physical Reads
parts	2	345	6
Total	2	345	6

	CPU	Elapsed
SQL Server parse and compile time:	00:00:00.000	00:00:00.000
SQL Server Execution Times:	00:00:00.031	00:00:00.090
Total	00:00:00.031	00:00:00.090

Table	Scan Count	Logical Reads	Physical Reads
parts	1	71	3
Total	1	71	3

	CPU	Elapsed
SQL Server parse and compile time:	00:00:00.000	00:00:00.000
SQL Server Execution Times:	00:00:00.000	00:00:00.139
Total	00:00:00.000	00:00:00.139

Παρατηρούμε ότι το I (είναι το αρχικό ερώτημα, με τη χρήση intersect) αξιοποιεί τα ευρετήρια Q4_idx1, Q4_idx2 για να κάνει Index Seek. Στη συνέχεια τα ενδιάμεσα αποτελέσματα τροφοδοτούνται στο Merge Join.

Από την άλλη το II (με τη χρήση and) καταφέρνει να αξιοποιήσει το Q4_idx3 (που ορίστηκε με αυτό ακριβώς το σκεπτικό) ώστε δεν χρειάζεται καμία άλλη πράξη! Το ευρετήριο έχει κατασκευαστεί στα γνωρίσματα brand, manufacturer (με αυτή τη σειρά) ώστε πρωτίστως να περιοριστούν οι εγγραφές ως προς το brand='Origin' (μόνο 8096 εγγραφές θα ικανοποιούν αυτή τη συνθήκη) και στη συνέχεια από αυτές να επιλεγθούν μόνον όσες ικανοποιούν και τη συνθήκη manufacturer='Domkapa'. Ο λόγος που επιλέγεται αυτή η σειρά είναι ότι το γνώρισμα brand έχει μεγαλύτερη επιλεκτικότητα (selectivity *) σε σχέση με το manufacturer και άρα το ευρετήριο αυτό θα εξυπηρετεί εν γένει καλύτερα τέτοιου είδους επερωτήματα (όχι δηλ. μόνο για το συγκεκριμένο brand και manufacturer). Τέλος, έχει οριστεί έτσι ώστε να είναι ευρετήριο επικάλυψης για το επερώτημα (μέσω του include).

* selectivity:

<code>select count(distinct brand)</code>	→ 25	/ 200000	←	<code>select count(*)</code>
<code>from parts</code>				<code>from parts</code>
<code>select count(distinct manufacturer)</code>	→ 5	/ 200000	←	
<code>from parts</code>				

Συγκριτικά για τα δύο επερωτήματα, παρατηρούμε ότι το II. (με τη χρήση and) επιφέρει σημαντικά λιγότερα logical reads (71) σε σχέση με το αρχικό (345) όταν έχουν κατασκευαστεί τα ίδια ευρετήρια (Q4_idx1, Q4_idx2, Q4_idx3) και συνεπώς είναι πιο αποδοτικό όπως φαίνεται και από τα ποσοστά σχετικού κόστους.

Ζήτηση 5°

Σημείωση: Τα σκέλη (1), (2) της άσκησης αντιμετωπίζονται μαζί, δηλαδή η απάντηση παρακάτω οργανώνεται ανά επερώτημα.

1° επερώτημα

Η εταιρεία πρέπει να επικοινωνεί με τους προμηθευτές της για τα προϊόντα που έχουν επιστραφεί ώστε να διερευνηθούν, να αντικατασταθούν ενδεχομένως με άλλα κ.λπ. Έστω ότι οι επιστροφές από τους πελάτες γίνονται δεκτές εντός 15 ημερών από την ημερομηνία παραλαβής του προϊόντος, συνεπώς στα μέσα του μήνα Ιουνίου (1995) πρέπει να ελεγχθούν οι επιστροφές που αφορούν παραγγελίες που έχουν παραδοθεί τον Μάιο (1995).

Το ακόλουθο επερώτημα επιστρέφει τους προμηθευτές (όνομα) με τους οποίους πρέπει να επικοινωνήσει η εταιρεία, το τηλέφωνο επικοινωνίας τους, το όνομα και το brand του προϊόντος που έχει επιστραφεί και τα συνολικά του τεμάχια (από όλες τις παραγγελίες) για τις παραγγελίες που έχουν παραδοθεί τον μήνα Μάιο 1995. Τα αποτελέσματα παρουσιάζονται ταξινομημένα ως προς τον προμηθευτή και το όνομα του προϊόντος.

```
select max(s.sname) as supplier, max(sphone) as phone, max(p.pname)
as "defective part", max(brand) as brand, sum(quantity) as units
from lineitem l, suppliers s, parts p
where l.suppkey = s.suppkey and p.partkey = l.partkey and
l.returnflag = 'R' and l.receiptdate between '1995-05-01' and '1995-05-31'
group by s.suppkey, p.partkey
order by s.suppkey, p.partkey
```

Για το ερωτήμα όπως παρουσιάστηκε κατασκευάζεται το ακόλουθο πλάνο εκτέλεσης (actual execution plan) όπου φαίνεται ότι για τους πίνακες lineitem, parts και suppliers γίνεται Clustered Index Scan (αντίστοιχη λειτουργία με το Table Scan, απλώς αξιοποιείται το ήδη υπάρχον ευρετήριο στα κλειδιά PK_ lineitem ____, PK_ parts ___ και PK_ supplier ____, αντίστοιχα:

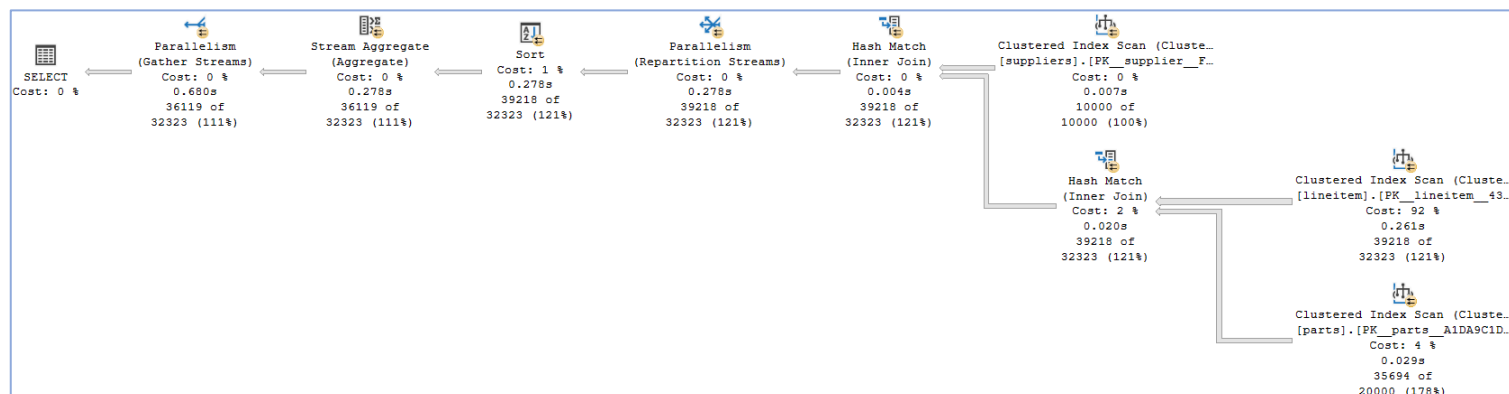


Table	Scan Count	Logical Reads	Physical Reads	Read-Ahead Reads	LOB Logical Reads	LOB Physical Reads	LOB Read-Ahead Reads	% Logical Reads of Total Reads
lineitem	7	60,387	1	59,311	0	0	0	94.668
parts	7	2,935	2	2,798	0	0	0	4.601
suppliers	7	466	1	162	0	0	0	0.731
Worktable	0	0	0	0	0	0	0	0.000
Total	21	63,788	4	62,271	0	0	0	

	CPU	Elapsed
SQL Server parse and compile time:	00:00:00.000	00:00:00.000
SQL Server Execution Times:	00:00:00.312	00:00:00.767
Total	00:00:00.312	00:00:00.767

Παρατηρούμε ότι οι πράξεις Clustered Index Scan (ουσιαστικά Table Scan) είναι κοστοβόρες και χρονοβόρες (ειδικά στον πολύ μεγάλο πίνακα lineitem όπου πρόκειται για το 92% του κόστους) οπότε θα προσπαθήσουμε να τις αποφύγουμε μέσω της δημιουργίας κατάλληλων ευρετηρίων στους τρεις πίνακες που συμμετέχουν στο ερωτήμα:

```

select max(s.sname) as supplier, max(sphone) as phone, max(p.pname)
as "defective part", max(brand) as brand, sum(quantity) as units
from lineitem l, suppliers s, parts p
where l.supkey = s.supkey and p.partkey = l.partkey and
l.returnflag = 'R' and l.receiptdate between '1995-05-01' and '1995-05-31'
group by s.supkey, p.partkey
order by s.supkey, p.partkey
  
```

```

create index supkeySuppliers on
suppliers(supkey) include (sname,
sphone)
  
```

```

create index
returnflag_receiptdate_partkey_supkeyLineitem
on lineitem (returnflag, receiptdate,
partkey, supkey) include (quantity)
  
```

```

create index partkeyParts on
parts(partkey) include (pname, brand)
  
```

Η κατασκευή των ευρητήριών αυτών έχει ως αποτέλεσμα το ακόλουθο πλάνο εκτέλεσης (actual execution plan) και στατιστικά:

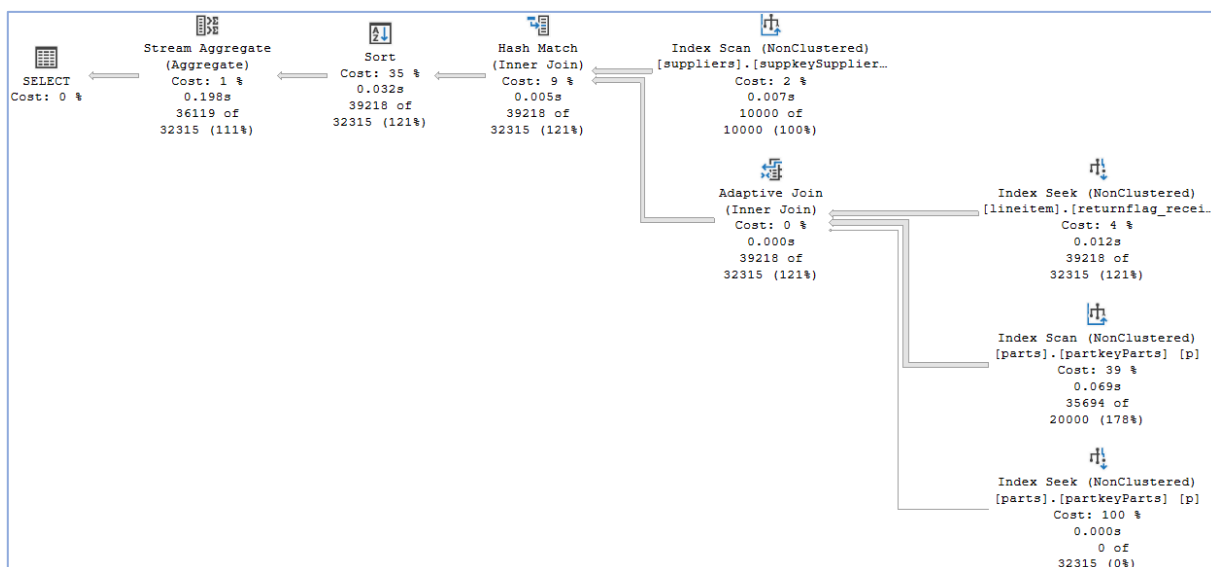


Table	Scan Count	Logical Reads	Physical Reads	Read-Ahead Reads	LOB Logical Reads	LOB Physical Reads	LOB Read-Ahead Reads	% Logical Reads of Total Reads
lineitem	1	149	2	146	0	0	0	9.313
parts	1	1,390	3	1,407	0	0	0	86.875
suppliers	1	61	1	59	0	0	0	3.813
Worktable	0	0	0	0	0	0	0	0.000
Total	3	1,600	6	1,612	0	0	0	

	CPU	Elapsed
SQL Server parse and compile time:	00:00:00.000	00:00:00.000
SQL Server Execution Times:	00:00:00.047	00:00:00.390
Total	00:00:00.047	00:00:00.390

Παρατηρούμε ότι και τα τρία ευρητήρια που κατασκευάστηκαν αξιοποιούνται.

Όσον αφορά τον πίνακα lineitem, γίνεται Index Seek πάνω στο ευρητήριο returnflag_receiptdate_partkey_suppkeyLineitem. Το ευρητήριο έχει κατασκευαστεί στα γνωρίσματα returnflag, receiptdate, partkey, suppkey (με αυτή τη σειρά) ώστε πρωτίστως να περιοριστούν οι εγγραφές ως προς το returnflag = 'R' και στη συνέχεια από αυτές να επιλεχθούν μόνον όσες ικανοποιούν και τη συνθήκη receiptdate between '1995-05-01' and '1995-05-31'. Στη συνέχεια, διευκολύνεται ο έλεγχος της συνθήκης p.partkey = 1.partkey της πρώτης σύζευξης που πραγματοποιείται (με τον πίνακα parts) και ακολούθως της συνθήκης l.suppkey = s.suppkey για τη δεύτερη σύζευξη (με τον πίνακα suppliers). Τέλος, λόγω της συμπερίληψης (include) του γνωρίσματος quantity δεν απαιτούνται πρόσθετες ανακτήσεις για το επερώτημα. Όλα αυτά έχουν ως αποτέλεσμα τα logical reads να μειώνονται ραγδαία (60387→149).

Για τον πίνακα parts γίνεται πλέον Index Scan πάνω στο ευρητήριο partkeyParts. Το ευρητήριο δεν χρησιμοποιείται για seek εφόσον δεν υπάρχει κάποια συνθήκη που αφορά το γνώρισμα rname αλλά παρ' όλα αυτά τα logical reads που αφορούν τον πίνακα parts μειώνονται σημαντικά (2935→1390), εφόσον το ευρητήριο είναι πολύ μικρότερο από τον

ίδιο τον πίνακα. Επιπλέον λόγω της συμπερίληψης (include) των γνωρισμάτων pname και brand δεν απαιτούνται πρόσθετες ανακτήσεις για το επερώτημα, γεγονός που συμβάλλει στη μείωση αυτή.

Τέλος, για τον πίνακα suppliers, γίνεται Index Scan πάνω στο ευρετήριο supkeySuppliers. Το ευρετήριο δεν χρησιμοποιείται για seek εφόσον δεν υπάρχει κάποια συνθήκη που αφορά το γνώρισμα sname αλλά παρ' όλα αυτά τα logical reads που αφορούν τον πίνακα suppliers μειώνονται σημαντικά (466→61), εφόσον το ευρετήριο είναι πολύ μικρότερο από τον ίδιο τον πίνακα. Επιπλέον λόγω της συμπερίληψης (include) του γνωρίσματος sname και sphone δεν απαιτούνται πρόσθετες ανακτήσεις για το επερώτημα, γεγονός που συμβάλλει στη μείωση αυτή.

2ο επερώτημα

Η εταιρία ενδιαφέρεται να παρακολουθεί τον μηνιαίο τζίρο για κάθε χώρα στην οποία πωλεί.

Το παρακάτω επερώτημα επιστρέφει για κάθε χώρα στην οποία πωλεί τα προϊόντα της η εταιρεία τον συνολικό τζίρο για τον μήνα Μάιο (1995).

```
select nation, sum(totalprice) as turnover
from nations n, customers c, orders o
where n.nationkey = c.nationkey and c.custkey = o.custkey and
o.orderdate between '1995-05-01' AND '1995-05-31'
group by nation
order by turnover desc
```

Για το επερώτημα όπως παρουσιάστηκε κατασκευάζεται το ακόλουθο πλάνο εκτέλεσης (actual execution plan) όπου φαίνεται ότι για τους πίνακες nations, customers και orders γίνεται Clustered Index Scan (αντίστοιχη λειτουργία με το Table Scan, απλώς αξιοποιείται το ήδη υπάρχον ευρετήριο στα κλειδιά PK_nations..., PK_customer... και PK_orders..., αντίστοιχα:

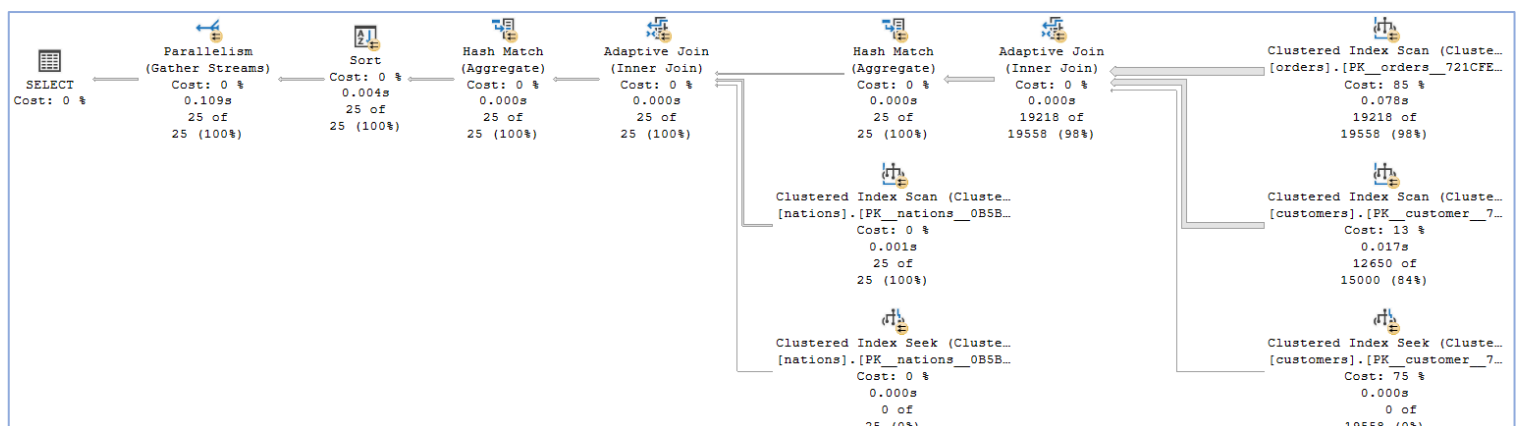


Table	Scan Count	Logical Reads	Physical Reads	Read-Ahead Reads	LOB Logical Reads	LOB Physical Reads	LOB Read-Ahead Reads	% Logical Reads of Total Reads
orders	7	16,961	1	16,729	0	0	0	86.316
customers	7	2,685	2	2,562	0	0	0	13.664
nations	7	4	1	0	0	0	0	0.020
Worktable	0	0	0	0	0	0	0	0.000
Total	21	19,650	4	20,721	0	0	0	

	CPU	Elapsed
SQL Server parse and compile time:	00:00:00.000	00:00:00.000
SQL Server Execution Times:	00:00:00.062	00:00:00.180
Total	00:00:00.062	00:00:00.180

Παρατηρούμε ότι οι πράξεις Clustered Index Scan (ουσιαστικά Table Scan) είναι κοστοβόρες και χρονοβόρες (ειδικά στον πολύ μεγάλο πίνακα orders όπου πρόκειται για το 85% του κόστους) οπότε θα προσπαθήσουμε να τις αποφύγουμε μέσω της δημιουργίας κατάλληλων ευρετηρίων στους πίνακες που συμμετέχουν στο επερώτημα. Όσον αφορά πάντως τον πίνακα nations, αυτός είναι πολύ μικρός (μόλις 25 εγγραφές, ενώ και μια ενδεχόμενη αύξηση του έχει άνω όριο) οπότε η δημιουργία οποιουδήποτε ευρετηρίου δεν θα είχε κανένα αντίκτυπο.

```
select nation, sum(totalprice) as turnover
from nations n, customers c, orders o
where n.nationkey = c.nationkey and c.custkey = o.custkey and
o.orderdate between '1995-05-01' AND '1995-05-31'
group by nation
order by turnover desc
```

```
create index nationkeyCustomers on
customers(nationkey)
```

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

Η κατασκευή των ευρετηρίων αυτών έχει ως αποτέλεσμα το ακόλουθο πλάνο εκτέλεσης (actual execution plan) και στατιστικά:

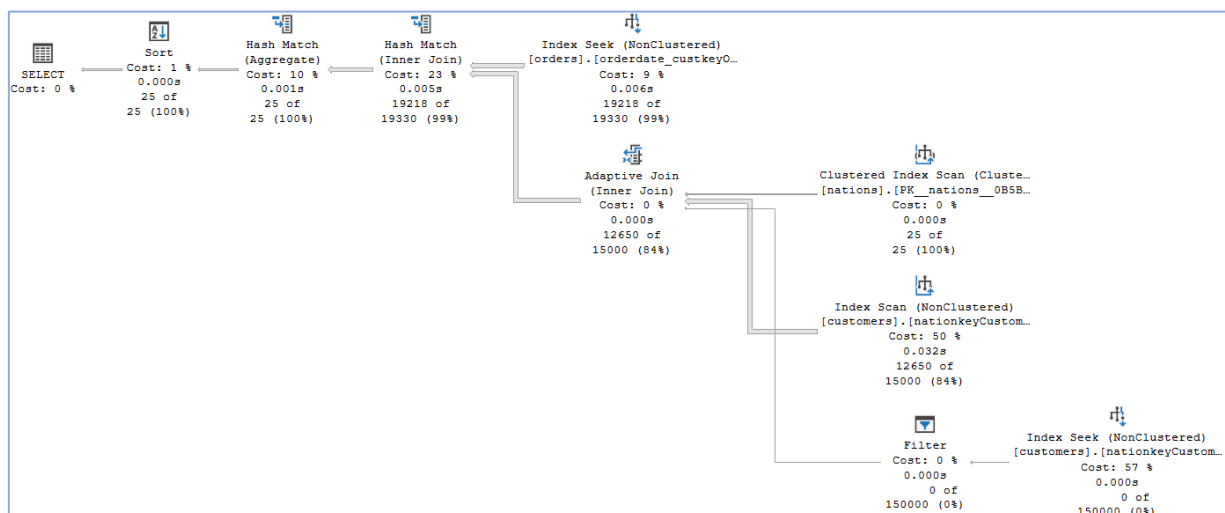


Table	Scan Count	Logical Reads	Physical Reads	Read-Ahead Reads	LOB Logical Reads	LOB Physical Reads	LOB Read-Ahead Reads	% Logical Reads of Total Reads
customers	1	264	2	269	0	0	0	81.734
nations	1	2	1	0	0	0	0	0.619
orders	1	57	2	54	0	0	0	17.647
Worktable	0	0	0	0	0	0	0	0.000
Total	3	323	5	323	0	0	0	

	CPU	Elapsed
SQL Server parse and compile time:	00:00:00.000	00:00:00.000
SQL Server Execution Times:	00:00:00.015	00:00:00.087
Total	00:00:00.015	00:00:00.087

Παρατηρούμε ότι και τα δύο ευρετήρια που κατασκευάστηκαν αξιοποιούνται.

Όσον αφορά τον πίνακα orders, γίνεται Index Seek πάνω στο ευρετήριο orderdate_custkeyOrders. Το ευρετήριο έχει κατασκευαστεί στα γνώρισμα orderdate, custkey ώστε να περιοριστούν μεν άμεσα οι εγγραφές σε αυτές που ικανοποιούν τη συνθήκη `o.orderdate between '1995-05-01' and '1995-05-31'` και στη συνέχεια να διευκολυνθεί ο έλεγχος της συνθήκης `c.custkey = o.custkey` της σύζευξης. Ως αποτέλεσμα τα logical reads να μειώνονται ραγδαία (16961→57).

Για τον πίνακα customers γίνεται πλέον Index Scan πάνω στο ευρετήριο nationkeyCustomers. Το ευρετήριο δεν χρησιμοποιείται για seek εφόσον δεν υπάρχει κάποια συνθήκη που αφορά το γνώρισμα nationkey αλλά παρ' όλα αυτά τα logical reads που αφορούν τον πίνακα customers μειώνονται σημαντικά (2685→264), εφόσον το ευρετήριο είναι πολύ μικρότερο από τον ίδιο τον πίνακα και άρα η ανάκτηση αυτού επιφέρει διάβασμα λιγότερων σελίδων (logical/physical reads).

Τέλος, παρατηρείται και μια μικρή μείωση των logical reads (4→2) που αφορούν τον πίνακα nations, ενδεχομένως λόγω των διαφορετικών επιλογών που έγιναν και φαίνονται στο πλάνο εκτέλεσης.