ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

Συστήματα Διαχείρισης Βάσεων Δεδομένων

Εργασία 1 2023-2024

Ονοματεπώνυμο: Κεχριώτη Ελένη

Ζήτημα 1

Εκτελούμε το αρχικό επερώτημα που δίνεται και καταγράφουμε τα στατιστικά εκτέλεσης του.

STATISTICS IO
(199550 rows affected)

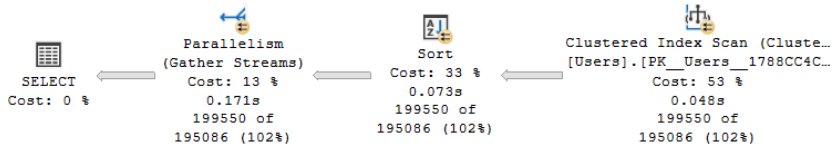
Row Num	Table	Scan Count	Logical Reads	Physical Reads	Read-Ahead Reads	LOB Logical Reads	LOB Physical Reads	LOB Read-Ahead Reads	% Logical Reads of Total Reads
	Users	13	6,001	2	0	0	0	0	100.000
	Total	**13**	**6,001**	**2**	**0**	**0**	**0**	**0**	

STATISTICS TIME

SQL Server Execution Times:
CPU time = 170 ms, elapsed time = 1536 ms.

Για το συγκεκριμένο επερώτημα κατασκευάστηκε το ακόλουθο πλάνο εκτέλεσης.

```
Query 1: Query cost (relative to the batch): 100%
SELECT displayName, profileviews FROM users WHERE YEAR(CreationDate)=2010 ORDER BY creationDate, profileViews
```



Από το πλάνο εκτέλεσης παρατηρούμε ότι γίνεται Clustered Index Scan για τον πίνακα Users (δηλαδή στην ουσία γίνεται Table Scan, απλώς χρησιμοποιείται το ήδη υπάρχων ευρετήριο στο πρωτεύον κλειδί PK_Users), το οποίο είναι περίπου το 53% του συνολικού κόστους. Σε συνδυασμό με τα παραπάνω στατιστικά, καταλαβαίνουμε πως η πράξη αυτή είναι πολύ χρονοβόρα και κοστοβόρα.

Αρχική ιδέα ήταν να δημιουργήσω ένα ευρετήριο πάνω στο CreationDate του πίνακα Users που να περιέχει τα γνωρίσματα DisplayName και ProfileViews.

```
create index creationDateUsers on
users(CreationDate) include(DisplayName, ProfileViews)
```

Ωστόσο, το ευρετήριο αυτό δεν χρησιμοποιείται λόγω του ότι χρησιμοποιείται η μέθοδος YEAR() πάνω στο CreationDate, η οποία δεν επιτρέπει στον optimizer να χρησιμοποιήσει το ευρετήριο για seek.
Αν αλλάξουμε τη συνθήκη στο WHERE με την ισοδύναμη της, δηλαδή

```
(WHERE YEAR(CreationDate) = 2010) == (WHERE CreationDate >= '2010-01-01' AND CreationDate< '2011-01-01')
```

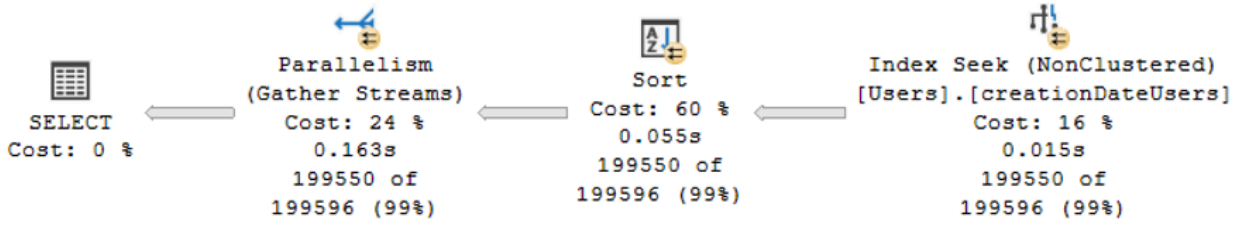
τότε παρατηρούμε ότι το ευρετήριο χρησιμοποιείται. Ας δούμε τα στατιστικά και το νέο πλάνο εκτέλεσης.

STATISTICS IO
(199550 rows affected)

Row Num	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
	Users	13	988	3	0	0	0	0	100.000
	Total	**13**	**988**	**3**	**0**	**0**	**0**	**0**	

STATISTICS TIME

SQL Server Execution Times:
CPU time = 263 ms, elapsed time = 2536 ms



Όπως βλέπουμε από τα στατιστικά και το πλάνο εκτέλεσης μειώθηκαν οι σελίδες που διαβάζονται κατά πολύ (6001 → 988) και μέσω του νέου ευρετηρίου μειώνεται το ποσοστό του κόστους στην πρώτη πράξη από 53% σε 16%. Ωστόσο αν και βελτιώθηκαν αρκετά αυτά τα δύο παρατηρούμε ότι έχει αυξηθεί κατα πολύ το κόστος που απαιτείται για την ταξινόμηση των αποτελεσμάτων (2η πράξη).
Δημιουργώντας ένα νέο ευρετήριο

```
create index creationDateUsers on
users(CreationDate, ProfileViews) include(DisplayName)
```

πάνω και στα δύο γνωρίσματα που γίνεται η ταξινόμηση συμπεριβαλομένου του γνωρίσματος DisplayName, ευελπιστώ ότι το ευρετήριο αυτό, όντας ευρετήριο κάλυψης, θα επαρκεί για την εκτέλεση του συγκεκριμένου επερωτήματος και θα βελτιστοποιεί ακόμα περισσότερο την εκτέλεση του.

Πράγματι μετά την εκτέλεση του επερωτήματος με το νέο αυτό ευρετήριο επιβεβαιώνομαι.

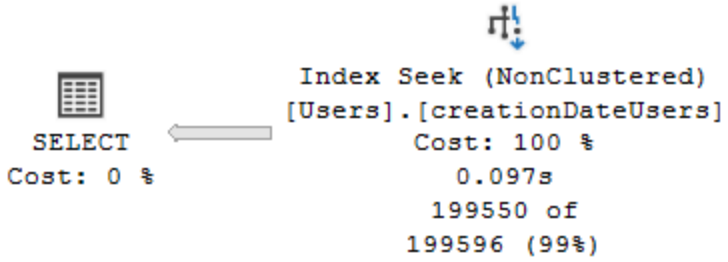
STATISTICS IO

(199550 rows affected)

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
Users	1	933	3	0	0	0	0	100.000
Total	**1**	**933**	**3**	**0**	**0**	**0**	**0**	

STATISTICS TIME

SQL Server Execution Times:
  CPU time = 0 ms, elapsed time = 2128 ms



Το ευρετήριο αυτό, όντως, επαρκεί και φαίνεται από το πλάνο εκτέλεσης που χρησιμοποιείται μόνο αυτό, ενώ ταυτόχρονα μειώθηκαν οι σελίδες ακόμα περισσότερο (988 → 933).

Ζήτημα 2

Τα στατιστικά της εκτέλεσης του επερωτήματος που δίνεται καθώς και το πλάνο εκτέλεσης φαίνονται παρακάτω.

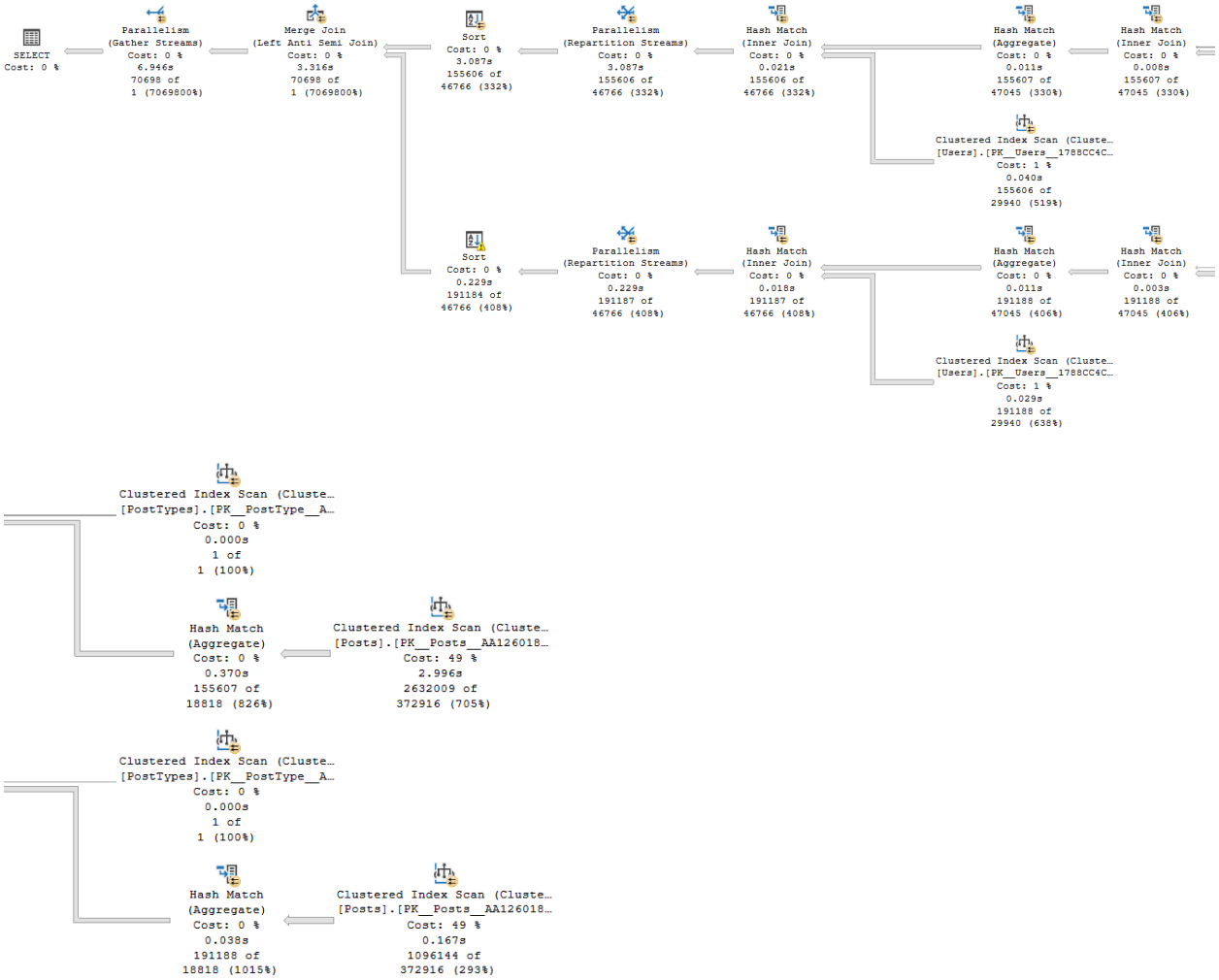
STATISTICS IO

(70698 rows affected)

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
Posts	26	746,319	2	0	0	0	0	98.416
PostTypes	5	8	1	0	0	0	0	0.001
Users	26	12,002	2	0	0	0	0	1.583
Total	**57**	**758,329**	**5**	**0**	**0**	**0**	**0**	

STATISTICS TIME

SQL Server Execution Times:
  CPU time = 3369 ms, elapsed time = 5083 ms



Από τα παραπάνω βλέπουμε ότι και εδώ γίνεται Clustered Index Scan για τους πίνακες Users, Posts, PostTypes (δηλαδή στην ουσία γίνεται Table Scan, απλώς χρησιμοποιείται το ήδη υπάρχων ευρετήριο στο πρωτεύον κλειδί PK_Users, PK_Posts, PK_PostTypes αντίστοιχα), με την πιο κοστοβόρα και χρονοβόρα να είναι η πράξη πάνω στο πίνακα Posts, η οποία αποτελεί το 49% του συνολικού κόστους. Θα μπορούσε να μειωθεί με την δημιουργία ενός ευρετηρίου, την οποία όμως θα εξετάσουμε αργότερα.
Αν εξετάσουμε λεπτομερώς τον παραπάνω πλάνο εκτέλεσης, αλλά και από τον ίδιο το επερώτημα, παρατηρούμε ότι γίνονται ίδιες πράξεις. Συγκεκριμένα το επερώτημα αποτελείται από 2 μέρη, με το πρώτο να είναι

```
SELECT users.*
 FROM users, posts, postTypes
WHERE users.userid=posts.ownerUserid and
 posts.postTypeid=PostTypes.postTypeid and postTypeName='Answer'
```

και το δεύτερο

```
SELECT users.*
```

```
 FROM users, posts, postTypes
WHERE users.userid=posts.ownerUserid and
 posts.postTypeid=PostTypes.postTypeid and postTypeName='Question'
```

Και στα δύο δημιουργούνται πίνακες με τους χρήστες που έχουν αναρτήσει απάντηση και ερώτηση αντίστοιχα. Οι ίδιες πράξεις που ανέφερα πριν και φαίνονται στο πλάνο εκτέλεσης βρίσκονται στα FROM - WHERE όπου γίνεται ένα καρτεσιανό γινόμενο 3 πινάκων και επιλέγονται οι πλειάδες που ικανοποιούν την συνθήκη "users.userid=posts.ownerUserid and posts.postTypeid=PostTypes.postTypeid".
Αυτές οι δύο πράξεις οδηγούν στο να διαβάζονται 746019 σελίδες για τον πίνακα Posts, οπότε μια ιδέα είναι να μειώσουμε στο μισό τις σελίδες αυτές κάνοντας ένα μόνο καρτεσιανό γινόμενο αντί για 2.
Κατέληξα στο παρακάτω επερώτημα

```
SELECT users.*
FROM users
JOIN posts ON users.userid=posts.ownerUserid
JOIN postTypes ON posts.postTypeid=PostTypes.postTypeid
GROUP BY users.UserId, AboutMe, users.CreationDate, DisplayName, DownVotes, LastAccessDate, UserLocation, Reputation, UpVotes, ProfileViews, WebsiteUrl
HAVING SUM(CASE WHEN postTypeName='Question' THEN 1 ELSE 0 END) = 0
```

στο οποίο γίνεται μια φορά Join στους πίνακες Users, Posts, PostTypes, και υπολογίζεται ως εξής:
Αναθέτουμε τιμές 1 και 0 ανάλογα με την τιμή του γνωρίσματος postTypeName, και υπολογίζουμε το άθροισμα αυτών των τιμών. Αν ένας χρήστης έχει απαντήσει σε ερωτήσεις αλλά δεν έχει δημοσιεύσει ο ίδιος μια ερώτηση τότε το άθροισμα του θα είναι 0, διαφορετικά θα είναι > 0. Στο επερώτημα αυτό κρατάμε μόνο τα 0 αθροίσματα.

Εκτέλεσα και τα δύο επερωτήματα μαζί σε δέσμη και παρακάτω φαίνονται τα αποτελέσματα.

```
Query 1: Query cost (relative to the batch): 67%
SELECT users.* FROM users, posts, postTypes WHERE users.useric
```

```
Query 2: Query cost (relative to the batch): 33%
select users.* from users join posts on users.useric
```

Το πρώτο επερώτημα (το αρχικό) καταλαμβάνει το 67% της δέσμης ενώ το δεύτερο μόλις το 33%, δηλαδή το μισό του πρώτου όπως αρχικά προβλέψαμε.

Ας δούμε όμως και τα στατιστικά του και το πλάνο εκτέλεσης.

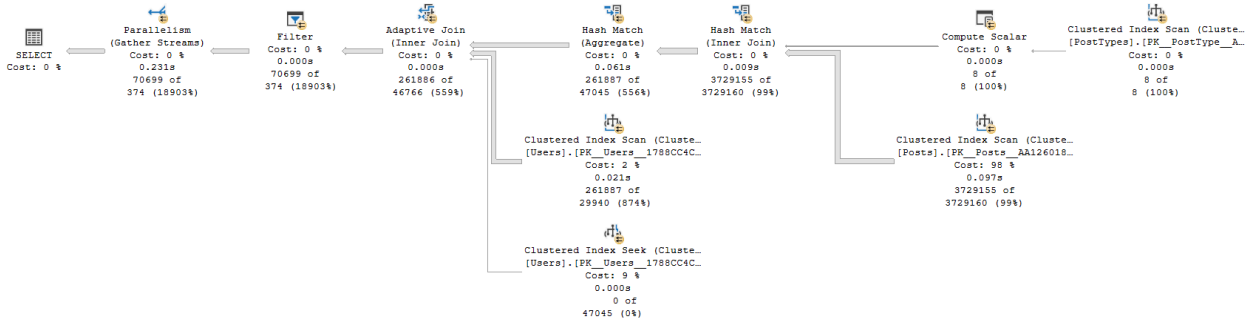
STATISTICS IO

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
Posts	13	371,152	2	0	0	0	0	98.416
PostTypes	11	4	1	0	0	0	0	0.001
Users	13	5,971	2	0	0	0	0	1.583
Total	**37**	**377,127**	**5**	**0**	**0**	**0**	**0**	

STATISTICS TIME

SQL Server Execution Times:
CPU time = 1422 ms, elapsed time = 1128 ms.

Πλάνο εκτέλεσης



Και από εδώ επιβεβαιωνόμαστε, καθώς και στους 3 πίνακες οι σελίδες που διαβάζονται για κάθε πίνακα έχουν πέσει ακριβώς στο μισό (12002 → 6001, 746019 → 375192, 8 → 4) και το πλάνο εκτέλεσης είναι φυσικά μικρότερο.
Ωστόσο, τώρα μπορούμε να παρατηρήσουμε ότι το 98% του συνολικού κόστους το καταλαμβάνει η πράξη Clustered Index Scan στο πίνακα Posts. Τώρα ήρθε η ώρα να συζητήσουμε για το ευρετήριο που ανέφερα και προηγουμένως.
Ο πίνακας Posts χρησιμοποιείται για να βρούμε τα γνωρίσματα ownerUserid και postTypeid για την αντιστοίχιση με τους πίνακες Users και PostTypes (κατά το join) αντίστοιχα. Θεωρητικά ένα ευρετήριο πάνω σε αυτά τα γνωρίσματα θα επιτάχυνε την εκτέλεση του επερωτήματος

Πράγματι μετά την δημιουργία του ευρετηρίου

```
create index OwnerIdPosts on
Posts(PostTypeId) include(OwnerUserId)
```

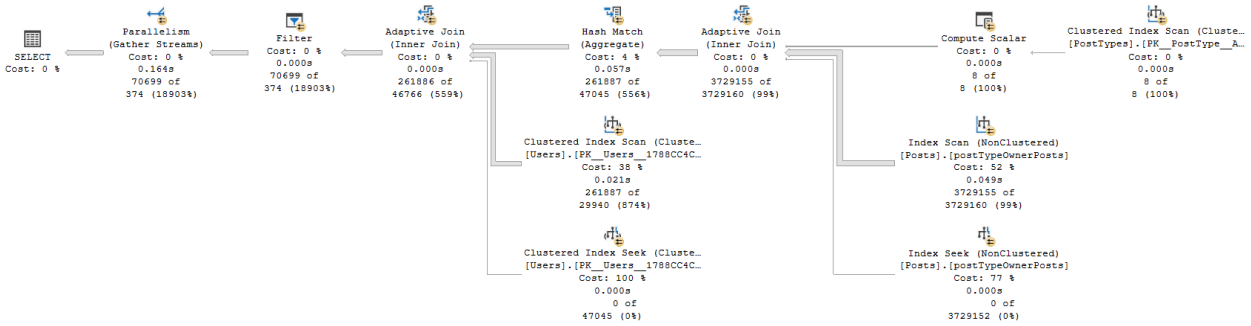
και την εκ νεου εκτέλεση του επερωτήματος βλέπουμε ότι το ευρετήριο βελτίωσε τις επιδόσεις του επερωτήματος.
Παρακάτω δίνονται τα στατιστικά και το πλάνο εκτέλεσης.

STATISTICS IO

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
Posts	13	6,990	1	0	0	0	0	54.077
PostTypes	10	4	1	0	0	0	0	0.031
Users	13	5,932	2	0	0	0	0	45.892
Total	**36**	**12,926**	**4**	**0**	**0**	**0**	**0**	

STATISTICS TIME

SQL Server Execution Times:
CPU time = 830 ms, elapsed time = 1010 ms.



Από τα παραπάνω παρατηρούμε ότι μειώθηκαν ραγδαία οι σελίδες στον πίνακα Posts (375192 → 6990). Στο πλάνο εκτέλεσης φαίνεται ότι χρησιμοποιείται το ευρετήριο όπως θέλαμε (Index Scan), αλλά δεν γίνεται Index Seek, γεγονός που είναι αρκετά κοστοβόρο για το επερώτημα. Αυτό οφείλεται λόγω της μεθόδου SUM() που χρησιμοποιείται, η οποία δεν επιτρέπει στον optimizer να χρησιμοποιήσει το ευρετήριο για seek. Δυστυχώς μια ισοδύναμη μέθοδο για SUM() θα απαιτούσε 2 νέα Join γυρνώντας μας πάλι στο αρχικό πρόβλημα που συζητήσαμε.

Ζήτημα 3

Τα στατιστικά και το πλάνο εκτέλεσης μετά την εκτέλεση του επερωτήματος που δίνεται φαίνονται παρακάτω.

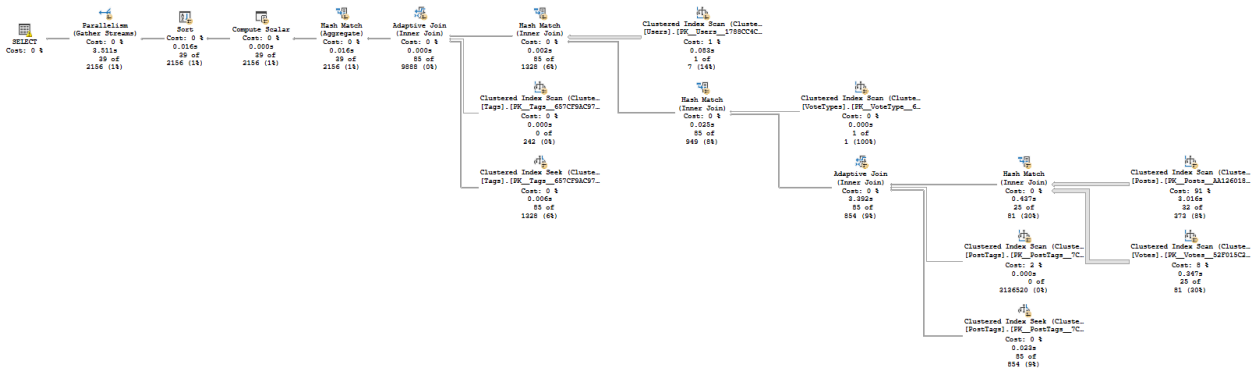
STATISTICS IO

(39 rows affected)

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
Posts	13	371,142	2	0	0	0	0	91.216
PostTags	24	72	17	0	0	0	0	0.018
Tags	0	170	17	0	0	0	0	0.042
Users	13	6,001	2	0	0	0	0	1.475
Votes	13	29,495	1	0	0	0	0	7.249
VoteTypes	13	4	1	0	0	0	0	0.001
Total	**76**	**406,884**	**40**	**0**	**0**	**0**	**0**	

STATISTICS TIME

SQL Server Execution Times:
CPU time = 620 ms, elapsed time = 3615 ms.



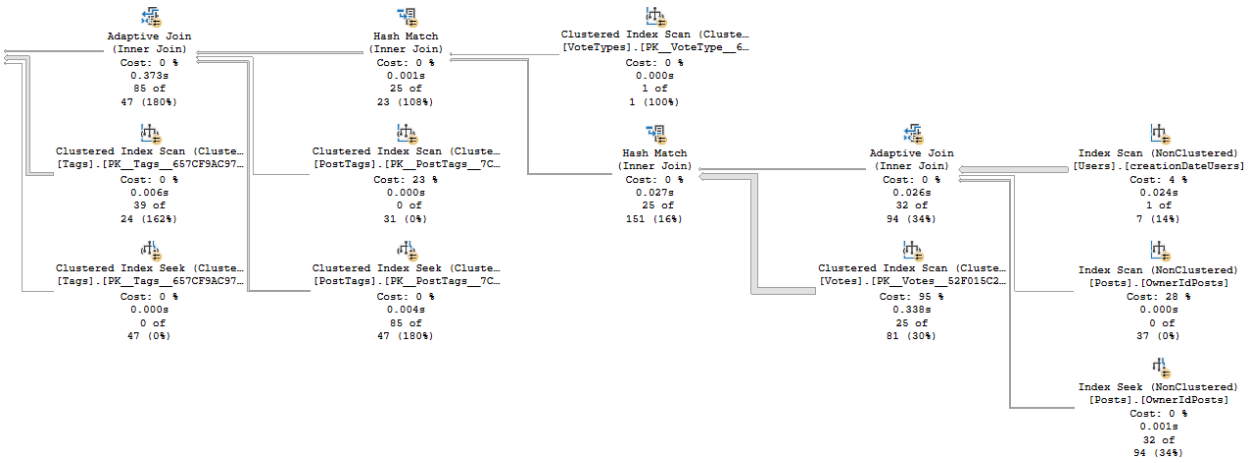
Παρατηρούμε ότι το 91% του κόστους που απαιτείται καταναλώνεται κατά το διάβασμα των σελίδων του πίνακα Posts στον οποίο γίνεται Index Scan πάνω στο πρωτεύον κλειδί PK_Posts_.. Ο πίνακας Posts χρησιμοποιείται μόνο για τα γνωρίσματα ParentId και OwnerUserId για τα Join με τους πίνακες PostTags και Users αντίστοιχα. Επομένως, ένα ευρετήριο πάνω σε αυτά τα γνωρίσματα θα βελτιστοποιούσε το επερώτημα.
Δημιούργησα το παρακάτω ευρετήριο

```
create index OwnerIdPosts on
Posts(OwnerUserId) include(PostId, ParentId)
```

και εκτέλεσα ξανά το επερώτημα.

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
Posts	1	3	3	0	0	0	0	0.010
PostTags	24	72	17	0	0	0	0	0.229
Tags	13	349	0	0	0	0	0	1.111
Users	13	1,482	3	0	0	0	0	4.719
Votes	13	29,495	1	0	0	0	0	93.918
VoteTypes	13	4	1	0	0	0	0	0.013
Total	**77**	**31,405**	**25**	**0**	**0**	**0**	**0**	

SQL Server Execution Times:
CPU time = 77 ms, elapsed time = 469 ms.



Το ευρετήριο χρησιμοποιείται όπως θα θέλαμε και επιταχύνει κατά πολύ την εκτέλεση του επερωτήματος. Μάλιστα, οι σελίδες που διαβάζονται κατά την εκτέλεση έχουν μειωθεί ραγδαία από 371.142 → 3. Ωστόσο, διπλασιάστηκαν οι σελίδες που χρειάζονται για τον πίνακα Tags.

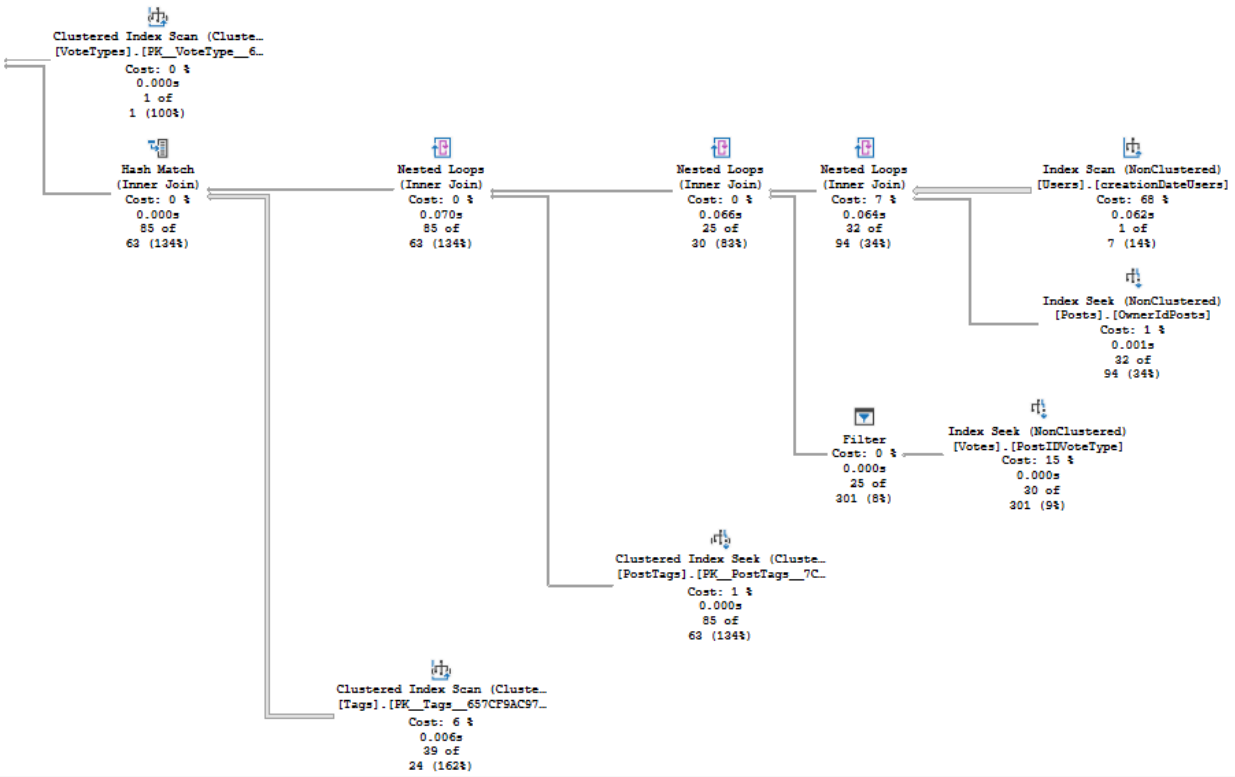
Επόμενο που φαίνεται ότι μπορούμε να βελτιώσουμε είναι να μειώσουμε το κόστος που χρειάζεται για τον πίνακα Votes, που είναι 84%.
Με την λογική που δημιούργησα και το προηγούμενο ευρετήριο σκέφτηκα και την δημιουργία του παρακάτω

```
create index PostIDVoteType on
Votes(PostId)  include(VoteTypeId)
```

Τα στατιστικά μετά την δημιουργία του ευρετηρίου και εκτέλεση του επερωτήματος φαίνονται παρακάτω.

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
Posts	1	3	3	0	0	0	0	0.153
PostTags	24	179	1	0	0	0	0	9.123
Tags	1	118	1	0	0	0	0	6.014
Users	1	1,404	3	0	0	0	0	71.560
Votes	32	256	1	0	0	0	0	13.048
VoteTypes	1	2	1	0	0	0	0	0.102
Total	**60**	**1,962**	**10**	**0**	**0**	**0**	**0**	

SQL Server Execution Times:
CPU time = 16 ms, elapsed time = 151 ms.



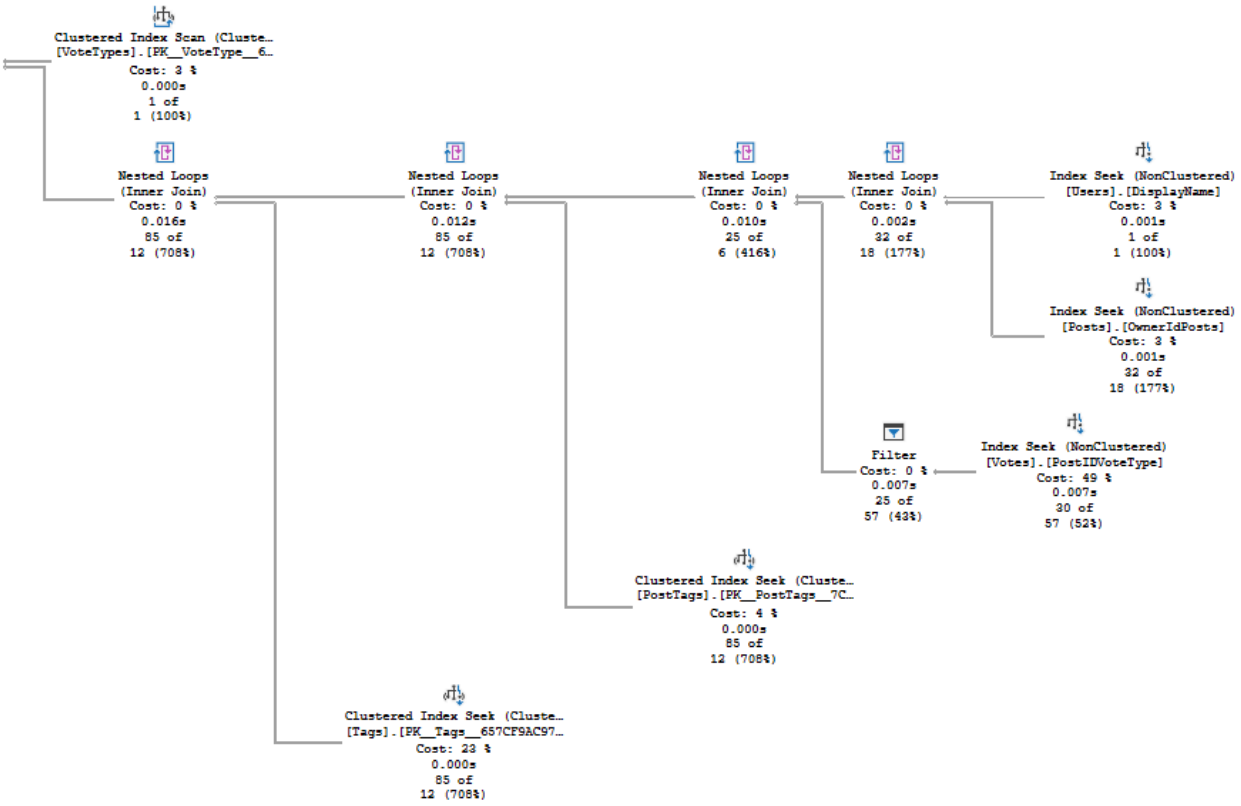
Αντίστοιχα και εδώ, μειώθηκε σημαντικά το κόστος στον πίνακα Votes (29.495→256), και στον πίνακα VoteTypes (4→2).

Τέλος, ένα σημαντικό ποσοστό του κόστους απαιτείται για τον πίνακα Users για τον οποίο χρειαζόμαστε μόνο το γνώρισμα DisplayName.
Επομένως, θα βελτίωνε ένα ευρετήριο πάνω σε αυτό το γνώρισμα.

```
create index DisplayName on
Users(DisplayName)
```

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
Posts	1	3	3	0	0	0	0	0.504
PostTags	24	172	1	0	0	0	0	28.908
Tags	0	319	1	0	0	0	0	53.613
Users	1	3	3	0	0	0	0	0.504
Votes	32	96	13	0	0	0	0	16.134
VoteTypes	1	2	1	0	0	0	0	0.336
Total	**59**	**595**	**22**	**0**	**0**	**0**	**0**	

SQL Server Execution Times:
CPU time = 0 ms, elapsed time = 84 ms



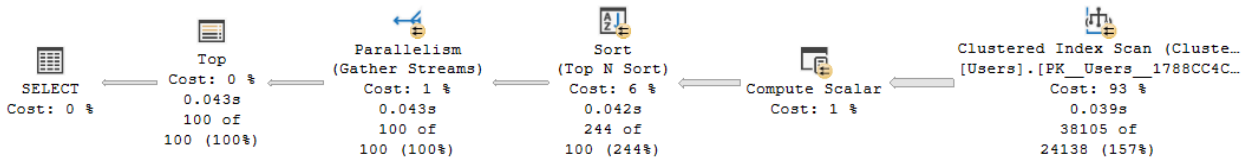
Απο τα παραπάνω βλέπουμε ότι το τελευταίο ευρετήριο μείωσε τις απαιτούμενες σελίδες από 6001 → 3.
Τελικά με τα 3 αυτά ευρετήρια που δημιουργήθηκαν ο χρόνος εκτέλεσης του επερωτήματος μειώθηκε ραγδαία όπως επίσης και οι συνολικές λογικές σελίδες που χρειαζόντουσαν για την εκτέλεση του επερωτήματος 406.884 → 595. Ταυτόχρονα, όμως μειώθηκαν σημαντικά και οι φυσικές σελίδες (40→22), που χρειάζεται να ανακτηθούν από τον δίσκο για την απάντηση του επερωτήματος.

Ζήτημα 4

Τα στατιστικά και το πλάνο εκτέλεσης μετά την εκτέλεση του επερωτήματος που δίνεται φαίνονται παρακάτω.

Row Num	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
	Users	13	6,001	2	0	0	0	0	100.000
	Total	**13**	**6,001**	**2**	**0**	**0**	**0**	**0**	

SQL Server Execution Times:
CPU time = 0 ms, elapsed time = 132 ms



Αν παρατηρήσουμε καλά το επερώτημα, θα δούμε ότι τα μόνα γνωρίσματα που χρησιμοποιούνται είναι το UserId, Reputation, UpVotes και DownVotes. Επομένως, ένα ευρετήριο το οποίο θα δημιουργούνταν πάνω σε κάποιο/α από αυτά τα γνωρίσματα δεν θα βελτιστοποιούσε το επερώτημα, αφού και τότε θα χρειαζόταν να γίνει Table Scan για να βρεθεί η τιμή του λοιπών γνωρίσματος. Άρα τα μόνα ευρετήρια που θα βελτιστοποιούσαν το επερώτημα θα ήταν ευρετήρια κάλυψης.
Κατέληξα στα παρακάτω δύο ευρετήρια τα οποία έχουν κατασκευαστεί στα γνωρίσματα UpVotes και Reputation, ενώ περιλαμβάνουν και το DownVotes.

```
create index UpRepDown on
Users(UpVotes) include(Reputation, DownVotes)

create index RepUpDown on
Users(Reputation) include(UpVotes, DownVotes)
```

Ο λόγος που περιλαμβάνεται μόνο το DownVotes είναι γιατί χρειαζόμαστε μόνο να το εμφανίσουμε στο τελικό αποτέλεσμα. Αντίθετα τα άλλα δύο τα χρησιμοποιούμε για να βρούμε τις εγγραφές που ικανοποιούν την συνθήκη Reputation > 1000 and Upvotes > 100. Αν και τα δύο ευρετήρια φαίνονται ίδια στην πραγματικότητα δεν είναι. Η σειρά με την οποία έχουν τοποθετηθεί τα γνωρίσματα έχει σημασία, καθώς με βάση αυτή τη σειρά κατασκευάζεται το ευρετήριο, το οποίο έχει διαφορετικά αποτελέσματα στην εκτέλεση του επερωτήματος.

Για να βρούμε πιο θα έχει την καλύτερη επίδοση μπορούμε να βρούμε την επιλεξιμότητα που έχει κάθε γνώρισμα.

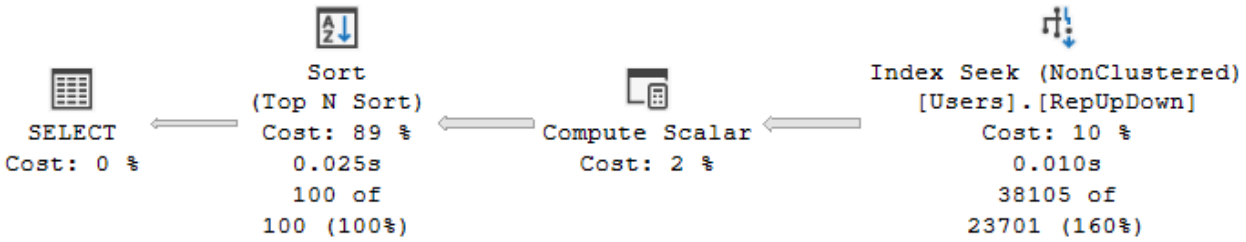
UpVotes → $\frac{4097}{299397}$ Reputation → $\frac{16280}{299397}$

Το Reputation έχει μεγαλύτερη επιλεξιμότητα, επομένως θα έχει και καλύτερες επιδόσεις.
Ας το δούμε όμως στην πράξη.

Παρακάτω φαίνονται τα αποτελέσματα για το ευρετήριο πάνω στο γνώρισμα Reputation.

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
Users	1	157	3	0	0	0	0	100.000
Total	**1**	**157**	**3**	**0**	**0**	**0**	**0**	

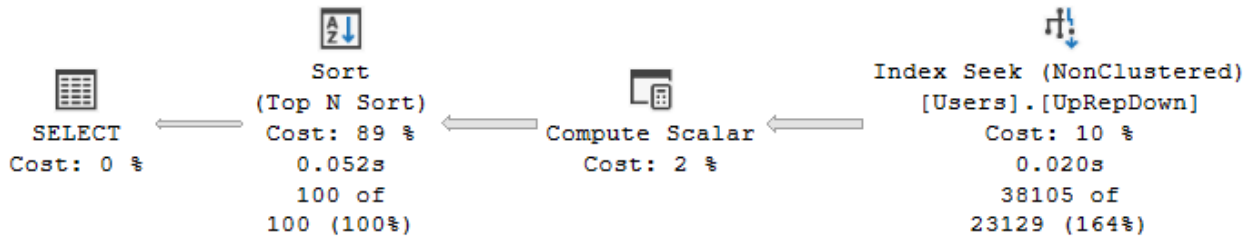
SQL Server Execution Times:
CPU time = 16 ms, elapsed time = 119 ms.



Και παρακάτω φαίνονται τα αποτελέσματα του ευρετηρίου πάνω στο UpVotes.

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
Users	1	153	3	0	0	0	0	100.000
Total	**1**	**153**	**3**	**0**	**0**	**0**	**0**	

SQL Server Execution Times:
CPU time = 0 ms, elapsed time = 306 ms.



Ζήτημα 5

1. Εμφάνισε για όλους τους χρήστες το id τους, το username τους και τον αριθμό των σχολίων που έχουν κάνει σε φθίνουσα σειρά.

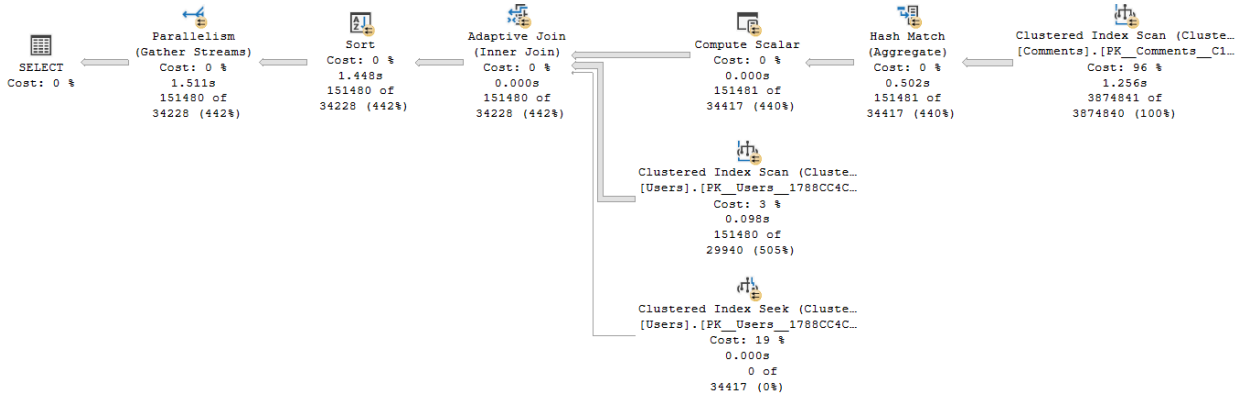
Το επερώτημα που απαντάει στο παραπάνω είναι

```
SELECT Users.UserId, DisplayName, COUNT(Users.UserId) as comments
FROM Users
JOIN Comments ON Users.UserId = Comments.UserId
GROUP BY Users.UserId, DisplayName
ORDER BY comments DESC
```

2. Τα στατιστικά εκτέλεσης του παραπάνω επερωτήματος μαζί με το πλάνος εκτέλεσης του φαίνεται παρακάτω

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
Comments	13	162,996	1	0	0	0	0	96.449
Users	13	6,001	2	0	0	0	0	3.551
Total	**26**	**168,997**	**3**	**0**	**0**	**0**	**0**	

SQL Server Execution Times:
CPU time = 1229 ms, elapsed time = 3023 ms



Δημιούργησα τα παρακάτω δύο ευρετήρια

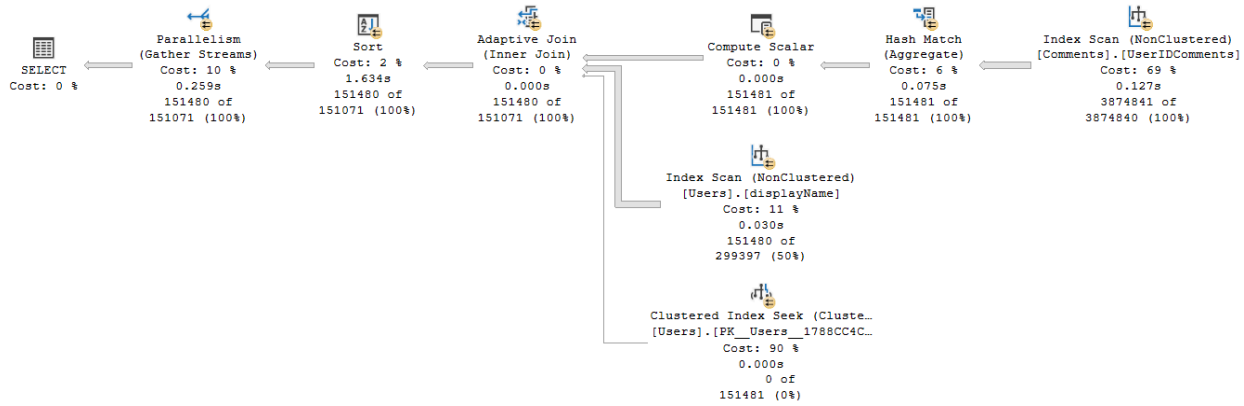
```
create index displayName on
Users(DisplayName)
```

create index UserIDComments on
Comments(UserId)

Και εκτέλεσα ξανά τα επερώτημα. Παρακάτω φαίνονται τα νέα στατιστικά και το πλάνο εκτέλεσης.

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
Comments	13	6,830	1	0	0	0	0	84.982
Users	13	1,207	3	0	0	0	0	15.018
Total	**26**	**8,037**	**4**	**0**	**0**	**0**	**0**	

SQL Server Execution Times:
CPU time = 733 ms, elapsed time = 2010 ms



Όπως φαίνεται και τα δύο ευρετήρια χρησιμοποιούνται και βελτιστοποιούν το επερώτημα, γεγονός που το καταλαβαίνουμε και από τον μειωμένο χρόνο εκτέλεσης, αλλά και από τις μειωμένες σελίδες (-160000 περίπου) που απαιτούνται για την απάντηση του επερωτήματος.