ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΣΥΣΤΗΜΑΤΑ ΔΙΑΧΕΙΡΙΣΗΣ ΚΑΙ ΑΝΑΛΥΣΗΣ ΔΕΔΟΜΕΝΩΝ

1η Εργασία

Ονοματεπώνυμο: ***Μαρία Σχοινάκη***

Αριθμός Μητρώου: ***3210191***

Email: ***p3210191@aueb.gr***

Ζήτημα 1ο

Για αυτούσιο το SQL επερώτημα του πρώτου ζητήματος, κατασκευάζεται το παρακάτω πλάνο εκτέλεσης (*actual execution plan*), όπου όπως φαίνεται και στα στάδια του πλάνου, ο πίνακας **users** υπόκεινται σε **Clustered Index Scan**, δηλαδή αναζητούνται δεδομένα που δεν μπορούν να εντοπιστούν αποτελεσματικά μέσω του ευρετηρίου (*ήδη υπάρχον ευρετήριο του πρωτεύοντος κλειδιού*) και άρα διαβάζονται όλες οι εγγραφές του ευρετηριασμένου πίνακα από την αρχή μέχρι το τέλος. Έτσι, το **Clustered Index Scan** είναι μια ένδειξη ότι το επερώτημα θα μπορούσε να βελτιωθεί με καλύτερη σχεδίαση ευρετηρίου ή προσαρμογή των συνθηκών ερωτήματος για να εξυπηρετηθούν καλύτερα τα δεδομένα που αναζητούνται.

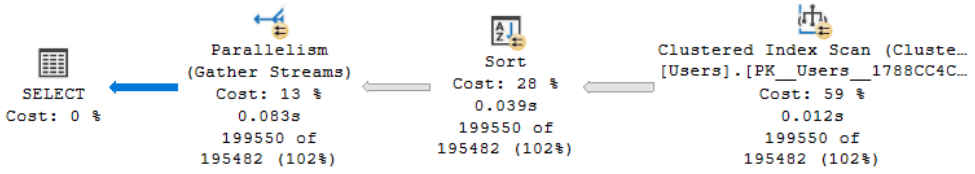


Table	Scan Count	Logical Reads	Physical Reads
Users	9	6001	2

Metric	CPU time	Elapsed time
SQL Server Execution Times:	126 ms	879 ms

Όπως φαίνεται από τους πίνακες, τα **Clustered Index Scan** αποτελούν πολύ κοστοβόρες και χρονοβόρες πράξεις. Έτσι, προκειμένου να βελτιστοποιήσουμε την αναζήτησή μας, θα προσπαθήσουμε να δημιουργήσουμε ευρετήρια και μετέπειτα, να γράψουμε εναλλακτικές λύσεις επερωτήσεων, που παράγουν το επιθυμητό αποτέλεσμα, αλλά σε μια οικονομικότερη κλίμακα.

Το αρχικό ερώτημα SQL είναι απαλλαγμένο από ευρετήρια. Με την προσθήκη ενός ευρετηρίου, επιδιώκουμε η αναζήτηση να γίνεται απλούστερα και αποτελεσματικότερα. Η επιλογή του ευρετηρίου, έγινε βάση του τρόπου πρόσβασης του επερωτήματος στα δεδομένα. Εφόσον η ταξινόμηση βασίζεται στα πεδία **CreationDate** και **ProfileViews**, ένα σύνθετο ευρετήριο σε αυτά τα πεδία θα βελτιστοποιήσει την ανάκτηση και την ταξινόμηση των εγγραφών, ικανοποιώντας ταυτόχρονα και το φιλτράρισμα. Είναι πιο αποδοτικό ένα ευρετήριο που είναι χωρισμένο σε πρώτο επίπεδο ως προς το πιο **επιλέξιμο** γνώρισμα (***CreationDate***), καθώς μπορεί να πάρει μόνο **884** τιμές, όπως μπορούμε να επαληθεύσουμε και από το SQL script. Αντίθετα, το **profileViews**, μπορεί να πάρει **4737** τιμές, όπως μπορούμε να επαληθεύσουμε. Δημιουργώντας τα αντίστοιχα ευρετήρια πειραματικά και συγκρίνοντάς τα με βάση τα στατιστικά, όντως καταλήγουμε ότι το (**CreationDate, profileViews**) είναι το πιο αποδοτικό. Άρα καταλήγουμε σε ένα ευρετήριο που είναι αποτελεσματικά δομημένο έτσι ώστε να προσδίδει αποδοτικότητα σε τέτοιου είδους επερωτήματα. Επίσης, η συμπερίληψη του γνωρίσματος **displayName** στο ευρετήριο, επιτρέπει την εκτέλεση του ερωτήματος και άρα ανάκτηση του γνωρίσματος ήδη από το ευρετήριο, χωρίς να χρειάζεται επιπλέον αναζήτηση/ανάκτηση από τον πίνακα users, μειώνοντας έτσι το υπολογιστικό κόστος.

```
SELECT COUNT(DISTINCT CreationDate)
FROM users
```

	(No column name)
1	884

```
SELECT COUNT(DISTINCT ProfileViews)
FROM users
```

	(No column name)
1	4737

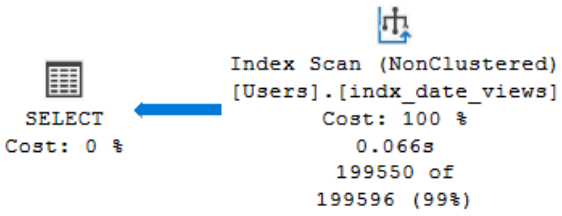


Table	Scan Count	Logical Reads	Physical Reads
Users	1	1399	3

Metric	CPU time	Elapsed time
SQL Server Execution Times:	32 ms	740 ms

Από το παραπάνω πλάνο εκτέλεσης (*actual execution plan*), φαίνεται ότι το ευρετήριο που δημιουργήθηκε, αξιοποιείται και μάλιστα πλέον γίνεται Index Sekek, με αποτέλεσμα να έχει μειωθεί σημαντικά ο χρόνος αναζήτησης προσδίδοντας έτσι σαφέστερη αποτελεσματικότητα και μετρική βελτιστοποίηση. Τα στατιστικά επαληθεύουν τον ισχυρισμό καθώς, τα **logical reads** έχουν πέσει σημαντικά **6001** ⇒**1399** και ο **χρόνος επεξεργασίας** έχει μειωθεί κατά πολύ **126ms** ⇒**32ms**. Μπορεί πειραματικά να δοκιμαστεί σε συνδυασμό με το ευρετήριο, η συγγραφή εναλλακτικών επερωτήσεων που παράγουν το επιθυμητό αποτέλεσμα.

Η συνάρτηση **YEAR**, καταναλώνει ένα τεράστιο ποσοστό του υπολογιστικού κόστους, με αποτέλεσμα να επιβραδύνει και να επιφορτώνει την αναζήτηση. Μία απλή προσέγγιση είναι να αντικαταστήσουμε την σημασιολογία της συνάρτησης αυτής, με ένα απλοϊκό ερώτημα που προσδιορίζει την ίδια χρονολογική περίοδο, όπως η **YEAR**.

```
SELECT displayName, profileviews
FROM users
WHERE YEAR(CreationDate)=2010
ORDER BY CreationDate, profileViews
```

```
CREATE INDEX indx_date_views ON
users (CreationDate, ProfileViews)
INCLUDE (displayName);
```

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

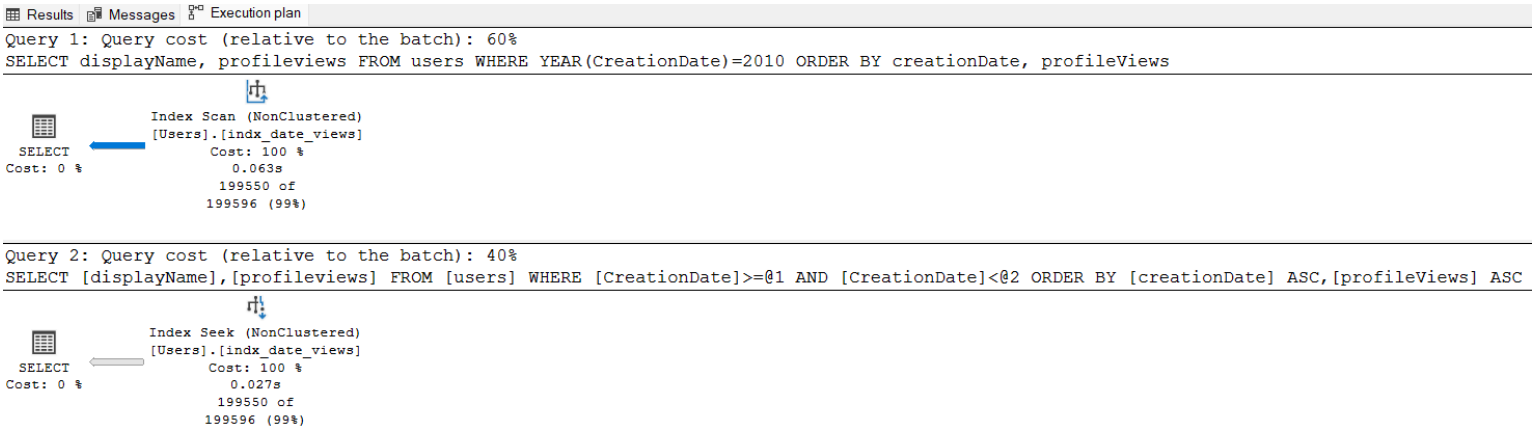


Table	Scan Count	Logical Reads	Physical Reads
Users	1	928	3

Metric	CPU time	Elapsed time
SQL Server Execution Times:	0 ms	754 ms

Συγκριτικά για τα 2 επερωτήματα, παρατηρούμε ότι το 2ο (*επανασυγγραφή*) έχει πολύ λιγότερα **logical reads (928)** σε σχέση με το αρχικό **(1399)** και είναι αποδοτικότερο όπως φαίνεται συγκριτικά στο κόστος δέσμης. Όπως παρατηρούμε από τα στατιστικά και τους χρόνους εκτέλεσης, η επανασυγγραφή του κώδικα βοήθησε πολύ στην αποδοτικότητα του ερωτήματος, καθώς απόπλεξε την πολυπλοκότητα του προσφέροντας μετρική βελτίωση και χρονική βελτίωση.

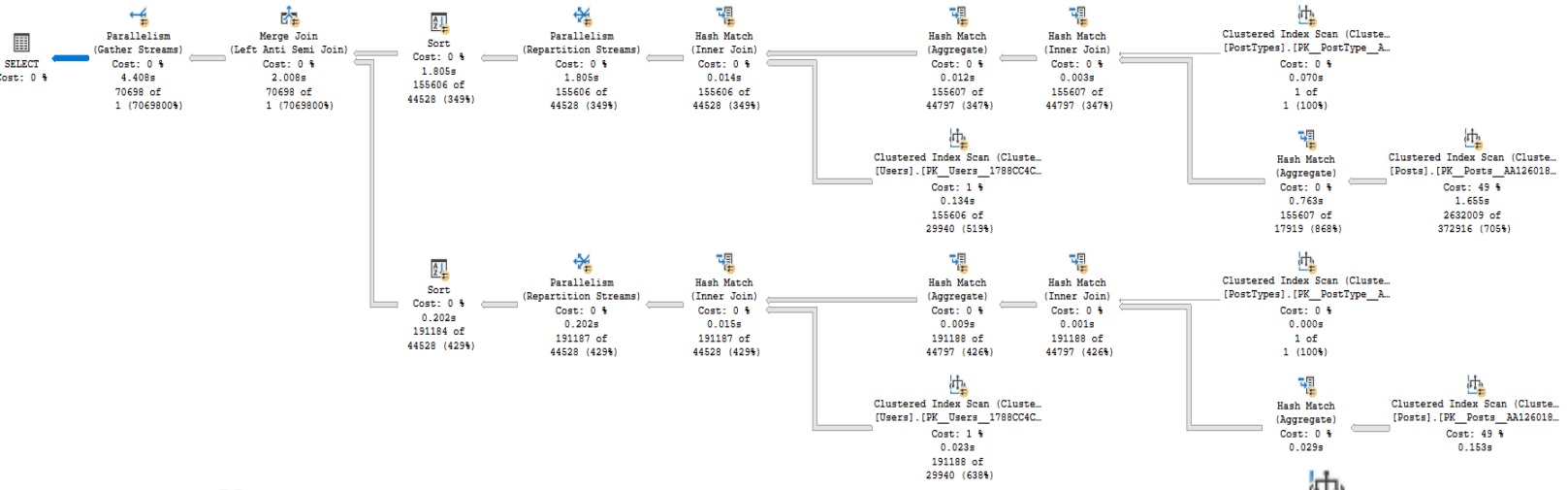
Για τον πίνακα **users**, λόγω της δημιουργίας του ευρετηρίου **indx_date_views**, γίνεται **Index Seek** πάνω σε αυτό προκειμένου να εντοπιστούν οι εγγραφές που ικανοποιούν την συνθήκη [CreationDate >= '2010-01-01' AND CreationDate < '2011-01-01'] (*δηλαδή έτος 2010*) και να βοηθηθεί μετέπειτα, η ταξινόμηση βάση το **CreationDate** και το **profileViews**. Επίσης λόγω της συμπερίληψης (*include*), του γνωρίσματος **displayName**, δεν χρειάζονται επιπλέον προσπελάσεις/ανακτήσεις για τον πίνακα αυτό. Η παρουσία του **include** στο ευρετήριο επιτρέπει την εκτέλεση ερωτημάτων που ανακτούν το **displayName** χωρίς την ανάγκη αναζήτησης στον κύριο πίνακα **users**, μειώνοντας τον χρόνο ανάκτησης και το κόστος σε πόρους.

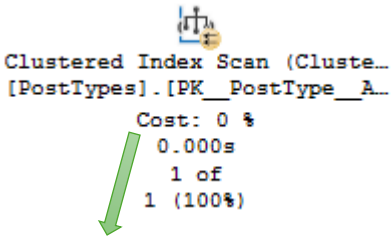
Όπως παρατηρείται και από τα στατιστικά πριν και μετά την προσπάθεια βελτιστοποίησης, υπάρχει μια σημαντική μείωση των **logical reads 6001⟹928**, καθώς και του **χρόνου επεξεργασίας 126ms⟹0ms**. Το γεγονός ότι τα **logical reads** και ο **χρόνος εκτέλεσης** μειώθηκαν τόσο δραματικά είναι ένδειξη ότι η τοποθέτηση των κατάλληλων ευρετηρίων, καθώς και η στρατηγική οργάνωση των πεδίων εντός αυτών, είναι κρίσιμη για την αποδοτική ανάκτηση δεδομένων.

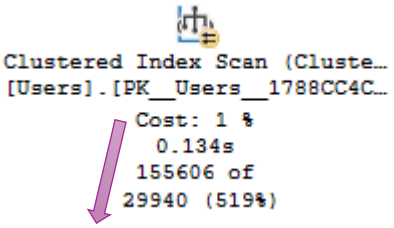
Συμπερασματικά, η προσαρμογή του επερωτήματος και η δημιουργία του ευρετηρίου βελτιστοποίησαν στο έπακρο τους χρόνους επεξεργασίας και τις μετρικές αναζήτησης, κάνοντας την αναζήτηση ταχύτερη και πιο αποτελεσματική.

Ζήτημα 2ο

Για αυτούσιο το SQL επερώτημα του δεύτερου ζητήματος, κατασκευάζεται το παρακάτω πλάνο εκτέλεσης (*actual execution plan*). Το αρχικό πλάνο είναι αρκετά μεγάλο και καθόλου ευανάγνωστο οπότε για να εξυπηρετήσουμε την τεκμηρίωση του ερωτήματος θα μεγεθύνουμε τα βασικά του σημεία. Οι πίνακες **users, posts, postTypes** υπόκεινται σε Clustered Index Scan, δηλαδή αναζητούνται δεδομένα που δεν μπορούν να εντοπιστούν αποτελεσματικά μέσω του ευρετηρίου (*ήδη υπάρχον ευρετήριο του πρωτεύοντος κλειδιού*) και άρα διαβάζονται όλες οι εγγραφές του ευρετηριασμένου πίνακα από την αρχή μέχρι το τέλος.







Το κόστος της συγκεκριμένης πράξης είναι **0%**, οπότε δεν υπάρχει σχετική ανάγκη βελτίωσης.

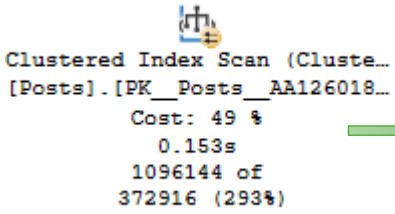
Ο πίνακας users, χρησιμοποιείται για το φιλτράρισμα των εγγραφών: `users.userid=posts.ownerUserid`.

Ο πίνακας postTypes είναι ένας μικρός πίνακας με μόλις **8** εγγραφές όπως αναφέρεται στην εκφώνηση. Μπορούμε και τρέχοντας το SQL script να το επαληθεύσουμε. Έτσι, οποιαδήποτε δημιουργία ευρετηρίου μάλλον θα ήταν περιττή και μάλιστα δεν θα ικανοποιούσε αρκετά την αναλογία δημιουργίας του ως προς το κόστος δημιουργίας, σε σχέση με το κόστος επεξεργασίας.

Δημιουργήθηκε αντίστοιχο ευρετήριο στο γνώρισμα **userId** πειραματικά προκειμένου να βελτιωθεί η απόδοση του ερωτήματος στον πίνακα users, όμως τα στατιστικά παρέμειναν τα ίδια. Ο λόγος είναι ότι το ιδανικό ευρετήριο θα πρέπει να συμπεριλαμβάνει (*include*) όλα τα γνωρίσματα που ζητάει το επερώτημα να ανακτηθούν, τα οποία στο συγκεκριμένο επερώτημα λόγω του select, είναι όλα τα γνωρίσματα του πίνακα users. Άρα στην ουσία το ευρετήριο θα πρέπει να είναι όλος ο πίνακας users, το οποίο είναι ανώφελο και υπολογιστικά ανούσιο. Άρα δεν προτείνεται ευρετήριο για τον συγκεκριμένο πίνακα.

```
SELECT *
FROM PostTypes
```

	PostTypeId	PostTypeName
1	1	Question
2	2	Answer
3	3	Wiki
4	4	TagWikiExerpt
5	5	TagWiki
6	6	ModeratorNomination
7	7	WikiPlaceholder
8	8	PrivilegeWiki



Ο πίνακας posts, χρησιμοποιείται για το φιλτράρισμα των εγγραφών: users.userid=posts.ownerUserid και posts.postTypeid=PostTypes.postTypeid.

Άρα μπορούμε να κατασκευάσουμε ευρετήριο στον πίνακα posts πάνω στα γνωρίσματα αυτά.

```
CREATE INDEX IX_posts ON posts
(postTypeid, ownerUserid);
```

Ο λόγος που το ευρετήριο δημιουργήθηκε πάνω σε αυτήν την σειρά των γνωρισμάτων είναι καθαρά για λόγους δόμησης. Είναι πιο αποδοτικό ένα ευρετήριο που είναι χωρισμένο σε πρώτο επίπεδο ως προς το πιο **επιλέξιμο** γνώρισμα (***postTypeid***), καθώς μπορεί να πάρει μόνο **4** τιμές, όπως μπορούμε να επαληθεύσουμε και από το SQL script. Αντίθετα, το **ownerUserid**, μπορεί να πάρει **261886** τιμές, όπως μπορούμε να επαληθεύσουμε. Δημιουργώντας τα αντίστοιχα ευρετήρια πειραματικά και συγκρίνοντάς τα με βάση τα στατιστικά, όντως καταλήγουμε ότι το (**postTypeid, ownerUserid**) είναι το πιο αποδοτικό. Άρα καταλήγουμε σε ένα ευρετήριο που είναι αποτελεσματικά δομημένο έτσι ώστε να προσδίδει αποδοτικότητα σε τέτοιου είδους επερωτήματα.

```
SELECT count(DISTINCT ownerUserid)
FROM posts
```

	(No column name)
1	261886

```
SELECT count(DISTINCT postTypeid)
FROM posts
```

	(No column name)
1	4

Τα στατιστικά πριν την προσπάθεια βελτιστοποίησης του επερωτήματος φαίνονται παρακάτω:

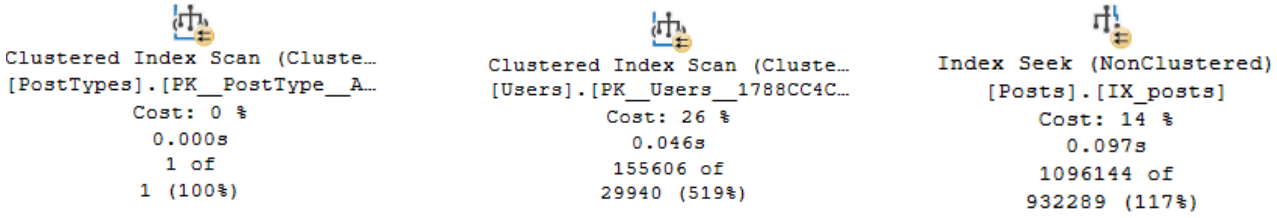
Table	Scan Count	Logical Reads	Physical Reads
Users	18	12002	2
Posts	18	747359	2
PostTypes	11	8	1
Worktable	0	0	0

Metric	CPU time	Elapsed time
SQL Server Execution Times:	2671 ms	3487 ms

Τα στατιστικά μετά την προσπάθεια βελτιστοποίησης με την χρήση του ευρετηρίου φαίνονται παρακάτω:

Table	Scan Count	Logical Reads	Physical Reads
PostTypes	18	8	1
Users	18	12002	2
Posts	2	6965	3
Worktable	0	0	0

Metric	CPU time	Elapsed time
SQL Server Execution Times:	1076 ms	1349 ms



Παρατηρείται μία αρκετά μεγάλη πτώση των **logical reads** που αφορούν τον πίνακα **posts** (**747359 6965**), καθώς και του **χρόνου επεξεργασίας** (**2671ms 1076ms**). Άρα, όπως φαίνεται και από το πλάνο εκτέλεσης, το ευρετήριο αξιοποιείται αποδοτικά και μάλιστα βελτιστοποιεί πολύ την απόδοση του ερωτήματος. Μπορεί πειραματικά να δοκιμαστεί σε συνδυασμό με το ευρετήριο, η συγγραφή εναλλακτικών επερωτήσεων που παράγουν το επιθυμητό αποτέλεσμα.

Η λειτουργία/τελεστής **EXCEPT**, της SQL προσδίδει χαμηλή απόδοση στο επερώτημα, καθώς επεξεργάζεται 2 φορές τα ίδια δεδομένα (*2 queries*), προκειμένου να καταλήξει στην διαφορά τους. Μπορούμε να προσπαθήσουμε να αντικαταστήσουμε την **EXCEPT**, ώστε να επιτύχουμε βελτίωση.

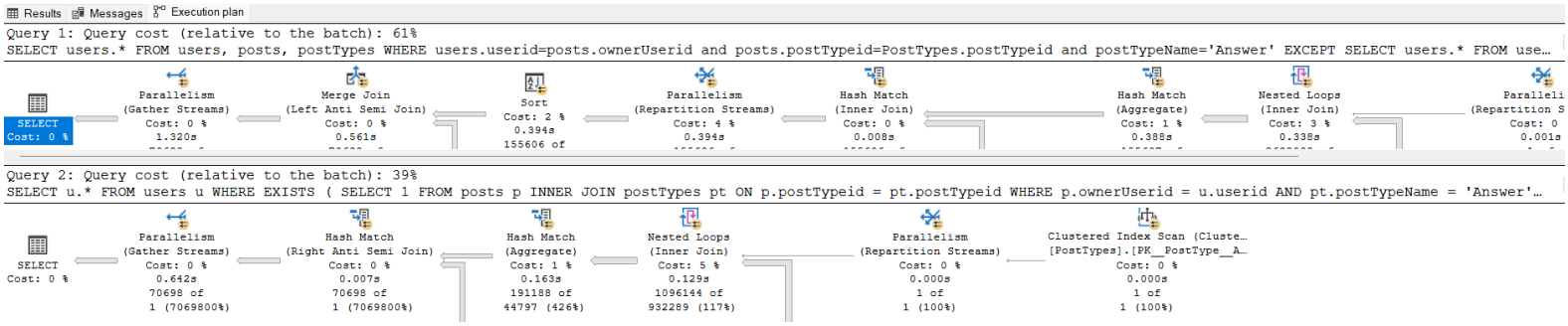
```sql
SELECT u.*
FROM users u
WHERE EXISTS (
    SELECT 1
    FROM posts p
    INNER JOIN postTypes pt ON p.postTypeid = pt.postTypeid
    WHERE p.ownerUserid = u.userid AND pt.postTypeName = 'Answer'
)
AND NOT EXISTS (
    SELECT 1
    FROM posts p
    INNER JOIN postTypes pt ON p.postTypeid = pt.postTypeid
    WHERE p.ownerUserid = u.userid AND pt.postTypeName = 'Question'
);
```

Ο λόγος που επιλέχθηκε η συγκεκριμένη δομή, είναι επειδή ψάχνει τουλάχιστον 1 απάντηση την φορά. Αν για παράδειγμα ένας χρήστης έχει πολλές απαντήσεις στο **EXISTS**, το επερώτημα θα σταματήσει να ψάχνει με το που βρει ότι έχει τουλάχιστον 1. Έτσι, γλιτώνονται άσκοπες αναζητήσεις και ανακτήσεις δεδομένων. Το ίδιο ισχύει και εσωτερικά του **NOT EXISTS**. Το **EXISTS** ελέγχει αν ο χρήστης έχει όντως μια απάντηση, ενώ το **NOT EXISTS**, ελέγχει αν ο χρήστης δεν έχει ερώτηση (*καμία*).

Όπως παρατηρούμε και στα στατιστικά παρακάτω, η επανασυγγραφή κώδικα, βοήθησε αρκετά καθώς περιόρισε την άσκοπη επεξεργασία δεδομένων. Συγκριτικά για τα 2 επερωτήματα, παρατηρούμε ότι το 2ο (*επανασυγγραφή*) έχει πολύ λιγότερα **logical reads (5986)** σε σχέση με το αρχικό **(12002)** και είναι αποδοτικότερο όπως φαίνεται συγκριτικά στο κόστος δέσμης.

Table	Scan Count	Logical Reads	Physical Reads
PostTypes	18	8	1
Users	9	5986	2
Posts	2	6965	3
Worktable	0	0	0

Metric	CPU time	Elapsed time
SQL Server Execution Times:	609 ms	1140 ms



Παρατηρείται ραγδαία μείωση των **logical reads** του πίνακα **users (12002⟹6965)**, καθώς πλέον με την συμβολή του query μειώθηκε η άσκοπη προσπέλαση δεδομένων. Γι' αυτό και ο **χρόνος επεξεργασίας** μειώθηκε (**1076ms⟹ 609ms**). Επίσης παρατηρούμε από το πλάνο εκτέλεσης (*actual execution plan*), ότι οι πράξεις μειώθηκαν κατά πολύ.

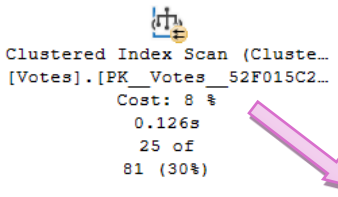
Συμπερασματικά, η προσαρμογή του επερωτήματος και η δημιουργία του ευρετηρίου βελτιστοποίησαν στο έπακρο τους χρόνους επεξεργασίας και τις μετρικές αναζήτησης, κάνοντας την αναζήτηση ταχύτερη και πιο αποτελεσματική, αποφεύγοντας άσκοπες προσπελάσεις δεομένων.

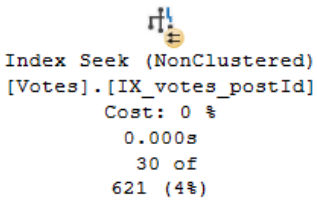
Ζήτημα 3ο

Για αυτούσιο το SQL επερώτημα του τρίτου ζητήματος, κατασκευάζεται το παρακάτω πλάνο εκτέλεσης (*actual execution plan*). Το αρχικό πλάνο είναι αρκετά μεγάλο και καθόλου ευανάγνωστο οπότε για να εξυπηρετήσουμε την τεκμηρίωση του ερωτήματος θα μεγεθύνουμε τα βασικά του σημεία. Οι πίνακες **users, posts, postTags, voteTypes, Votes, Tags** υπόκεινται σε **Clustered Index Scan**, δηλαδή αναζητούνται δεδομένα που δεν μπορούν να εντοπιστούν αποτελεσματικά μέσω του ευρετηρίου (*ήδη υπάρχον ευρετήριο του πρωτεύοντος κλειδιού*) και άρα διαβάζονται όλες οι εγγραφές του ευρετηριασμένου πίνακα από την αρχή μέχρι το τέλος.

Table	Scan Count	Logical Reads	Physical Reads
Tags	0	170	37
PostTags	24	72	25
Votes	9	29319	1
Posts	9	375192	3
VoteTypes	9	4	1
Users	9	6001	3
Worktable	0	0	0

Metric	CPU time	Elapsed time
SQL Server Execution Times:	451 ms	1132 ms

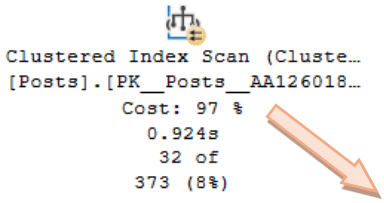


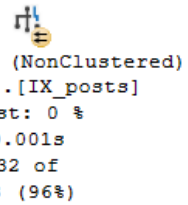



```
CREATE INDEX IX_votes_postId ON
Votes (postId);
```

Καταφέραμε να μετατρέψουμε την κοστοβόρα **Clustered Index Scan** σε **Index Seek**!

Δημιουργήθηκε αντίστοιχο ευρετήριο στο γνώρισμα **PostId** πειραματικά προκειμένου να βελτιωθεί η απόδοση του ερωτήματος στον πίνακα **votes**. Σύμφωνα με τα μετέπειτα στατιστικά, όντως το ευρετήριο βοήθησε την απόδοση και μείωσε τα **logical reads** από **29319** σε **436**, του πίνακα **votes**, αλλά και βοήθησε εμμέσως στην μείωση ανάκτησης δεδομένων και στους άλλους πίνακες.

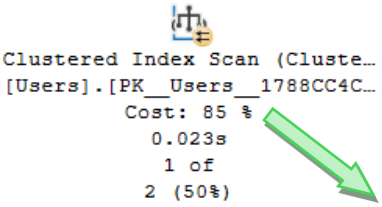


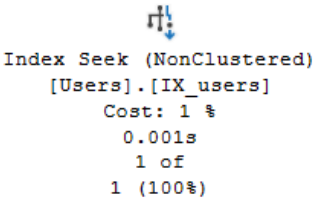



```
CREATE INDEX IX_posts ON posts
(OwnerUserId, ParentId);
```

Καταφέραμε να μετατρέψουμε την κοστοβόρα **Clustered Index Scan** σε **Index Seek**!

Δημιουργήθηκε αντίστοιχο ευρετήριο στα γνωρίσματα **OwnerUserId**, **ParentId** πειραματικά προκειμένου να βελτιωθεί η απόδοση του ερωτήματος στον πίνακα **posts**. Σύμφωνα με τα μετέπειτα στατιστικά, όντως το ευρετήριο βοήθησε την απόδοση και μείωσε ραγδαία τα **logical reads** από **375195** σε **3**, του πίνακα **posts**, αλλά και βοήθησε εμμέσως στην μείωση ανάκτησης δεδομένων και στους άλλους πίνακες.

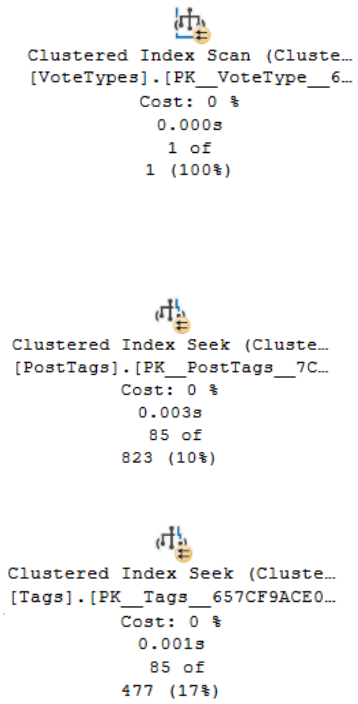





```
CREATE INDEX IX_users ON users
(displayName);
```

Καταφέραμε να μετατρέψουμε την κοστοβόρα **Clustered Index Scan** σε **Index Seek**!

Δημιουργήθηκε αντίστοιχο ευρετήριο στο γνώρισμα **displayName**, πειραματικά προκειμένου να βελτιωθεί η απόδοση του ερωτήματος στον πίνακα **users**. Σύμφωνα με τα μετέπειτα στατιστικά, όντως το ευρετήριο βοήθησε την απόδοση και μείωσε ραγδαία τα **logical reads** από **6001** σε **3**, του πίνακα **users**, αλλά και βοήθησε εμμέσως στην μείωση ανάκτησης δεδομένων και στους άλλους πίνακες. Στο ευρετήριο δεν χρησιμοποιήθηκε το γνώρισμα **userId**, καθώς δεν επηρέαζε καθόλου τα στατιστικά και το πλάνο εκτέλεσης οπότε θα ήταν ανούσιο και υπολογιστικά περιττό.



Ο πίνακας **voteType** είναι ένας μικρός πίνακας με μόλις **8** εγγραφές όπως αναφέρεται στην εκφώνηση. Μπορούμε και τρέχοντας το SQL script να το επαληθεύσουμε. Έτσι, οποιαδήποτε δημιουργία ευρετηρίου μάλλον θα ήταν περιττή και μάλιστα δεν θα ικανοποιούσε αρκετά την αναλογία δημιουργίας του ως προς το κόστος δημιουργίας, σε σχέση με το κόστος επεξεργασίας.

Οι συγκεκριμένοι πίνακες, δεν επιδέχονται βελτίωση στο συγκεκριμένο ερώτημα, καθώς το ευρετήριο by default του πρωτεύοντος κλειδιού κάνει άψογα την δουλειά της αναζήτησης για το ερώτημα και δεν χρειάζεται άλλο ευρετήριο. Επίσης παρατηρούμε ότι υπόκεινται ήδη σε **Clustered Index Seek** και μάλιστα με κόστος 0%.

Όπως παρατηρούμε και πειραματικά, οποιοδήποτε ευρετήριο πάνω σε αυτούς τους πίνακες δεν προσφέρει καμία βελτίωση, ή προσφέρει ελάχιστη, τόση όση να μην δικαιολογεί το κόστος δημιουργίας του.

Παρατηρούμε μια ραγδαία πτώση των στατιστικών του ερωτήματος μετρικά και χρονικά. Πλέον έχουμε παντού στο πλάνο εκτέλεσης (*actual execution plan*), λειτουργίες **Seek** και όχι **Scan**, και ακόμα και **Scan** να έχουμε είναι τόσο αμελητέα η επιβάρυνσή του που δεν αξίζει να δημιουργηθεί επιπλέον ευρετήριο. Καταφέραμε να μειώσουμε πολύ το υπολογιστικό κόστος με την δημιουργία μόλις 3 ευρετηρίων.

Table	Scan Count	Logical Reads	Physical Reads
VoteTypes	1	2	1
Worktable	0	0	0
Tags	0	315	1
PostTags	24	172	1
Votes	32	254	14
Posts	1	3	3
Users	1	3	3

Metric	CPU time	Elapsed time
SQL Server Execution Times:	0 ms	0 ms

Ζήτημα 4ο

Για αυτούσιο το SQL επερώτημα του τέταρτου ζητήματος, κατασκευάζεται το παρακάτω πλάνο εκτέλεσης (*actual execution plan*), όπου όπως φαίνεται και στα στάδια του πλάνου, ο πίνακας **users** υπόκεινται σε **Clustered Index Scan**, δηλαδή αναζητούνται δεδομένα που δεν μπορούν να εντοπιστούν αποτελεσματικά μέσω του ευρετηρίου (*ήδη υπάρχον ευρετήριο του πρωτεύοντος κλειδιού*) και άρα διαβάζονται όλες οι εγγραφές του ευρετηριασμένου πίνακα από την αρχή μέχρι το τέλος.

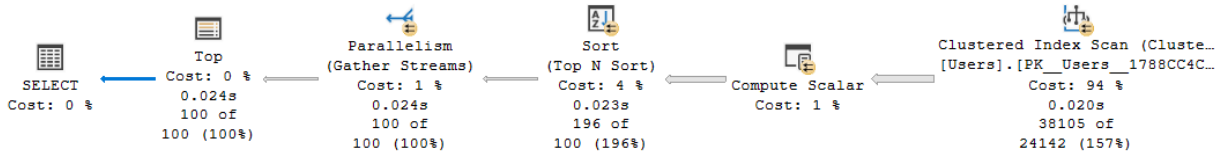


Table	Scan Count	Logical Reads	Physical Reads
Users	9	6001	2

Metric	CPU time	Elapsed time
SQL Server Execution Times:	0 ms	0 ms

Για την βελτιστοποίηση του επερωτήματος, μπορούμε να εξετάσουμε την δημιουργία ευρετηρίων στα γνωρίσματα που χρησιμοποιούνται στην συνθήκη **WHERE** και την **ORDER BY**.

1ο Προτεινόμενο Ευρετήριο

```
CREATE INDEX index_1 ON users (Reputation, UpVotes)
INCLUDE (DownVotes);
```

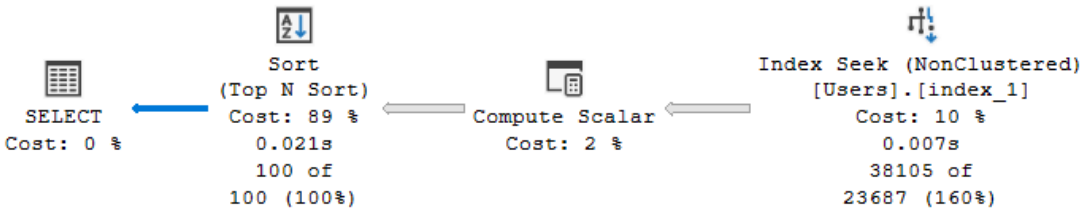


Table	Scan Count	Logical Reads	Physical Reads
Users	1	157	3

Metric	CPU time	Elapsed time
SQL Server Execution Times:	15 ms	78 ms

Όπως παρατηρούμε από τα στατιστικά, τα **logical reads** μειώθηκαν (*τα **physical reads** αυξήθηκαν κατά 1 αλλά είναι αναλογικά κερδοφόρο ως προς τα άλλα μετρικά*) από **6001** σε **157**. Έχουμε μία μικρή αύξηση του **χρόνου επεξεργασίας** από **0ms** σε **15ms**.

2ο Προτεινόμενο Ευρετήριο

```
CREATE INDEX index_2 ON users (UpVotes, Reputation)
INCLUDE (DownVotes);
```

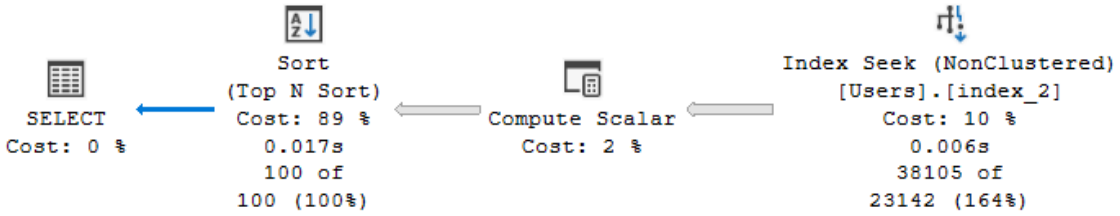


Table	Scan Count	Logical Reads	Physical Reads
Users	1	150	3

Metric	CPU time	Elapsed time
SQL Server Execution Times:	0 ms	73 ms

Όπως παρατηρούμε από τα στατιστικά, τα **logical reads** μειώθηκαν (*τα **physical reads** αυξήθηκαν κατά 1 αλλά είναι αναλογικά κερδοφόρο ως προς τα άλλα μετρικά*) από **6001** σε **150**. Ο χρόνος επεξεργασίας παρέμεινε ίδιος.

Ας προσεγγίσουμε το ερώτημα θεωρητικά. Το ευρετήριο που θα ταίριαζε θεωρητικά θα ήταν αυτό που θα ικανοποιούσε την επιλεξιμότητα. Το γνώρισμα **UpVotes** όπως μπορούμε εύκολα να συμπεράνουμε έχει μεγαλύτερη επιλεξιμότητα από το **Reputation**, οπότε ένα ευρετήριο το οποίο σε πρώτη φάση περιορίζει τις εγγραφές του βάση ενός μικρού συνόλου είναι πιο αποδοτικό.

Γενικά στην δημιουργία των ευρετηρίων αυτών δεν θα μπορούσε να υπάρχει άλλη πιθανή επιλογή. Θα έπρεπε οπωσδήποτε να μπει ως συμπερίληψη το γνώρισμα **DownVotes** καθώς χρησιμοποιείται στο **select** και λόγω του **WHERE clause** θα έπρεπε το ευρετήριο να ευρετηριάσει τον πίνακα βάση των πεδίων του **WHERE**, δηλαδή τα γνωρίσματα **Reputation** και **UpVotes**. Άρα η μόνη πιθανή διαφοροποίηση είναι η σειρά της ευρετηρίασης βάση των γνωρισμάτων που θεωρητικά επιλέγουμε το γνώρισμα με την μεγαλύτερη επιλεξιμότητα.

Προσεγγίζοντας το ερώτημα πρακτικά, επαληθεύουμε όλους τους ισχυρισμούς ως προς την καταλληλότητα των 2 ευρετηρίων και μάλιστα την επικράτεια του **2ου**. Ήδη από τα πλάνα εκτέλεσης (*actual execution plan*) ξεχωριστά μπορούμε να παρατηρήσουμε ότι πλέον η λειτουργία **Clustered Index Scan**, έγινε **Index Seek**. Αυτό σημαίνει ότι κάθε ευρετήριο χρησιμοποιείται και μάλιστα αποδοτικά (*1 ευρετήριο σε κάθε εκτέλεση*). Ακόμη, παρατηρούμε μία μικρή διαφορά στα **logical reads**, καθώς το **1ο** ευρετήριο προσφέρει **7** παραπάνω, πράγμα που καθιστά το **2ο** ευρετήριο το πιο κατάλληλο και αποδοτικό, όπως ακριβώς ισχυριστήκαμε. Η διαφορά και χρονικά και μετρικά είναι μικρή ανάμεσα στα 2 ευρετήρια στα πλαίσια αυτού του ερωτήματος. Υπό άλλες συνθήκες και δεδομένα, η επιλεξιμότητα θα μπορούσε να παίξει καθοριστικό παράγοντα απόδοσης.

Ζήτημα 5ο

Το ακόλουθο επερώτημα επιστρέφει με φθίνουσα σειρά κατά πλήθος σχολίων τους τίτλους των αναρτήσεων που έχουν σχόλια με score πάνω από 100 μαζί με το πλήθος των σχολίων αυτών.

```
SELECT P.Title, COUNT(C.Cid) AS Comments
FROM Posts AS P
LEFT JOIN Comments AS C ON P.PostId = C.PostId
WHERE C.Score > 100
GROUP BY P.PostId, P.Title
ORDER BY Comments DESC
```

Κατασκευάζεται το παρακάτω πλάνο εκτέλεσης (*actual execution plan*), πάνω στο επερώτημα.

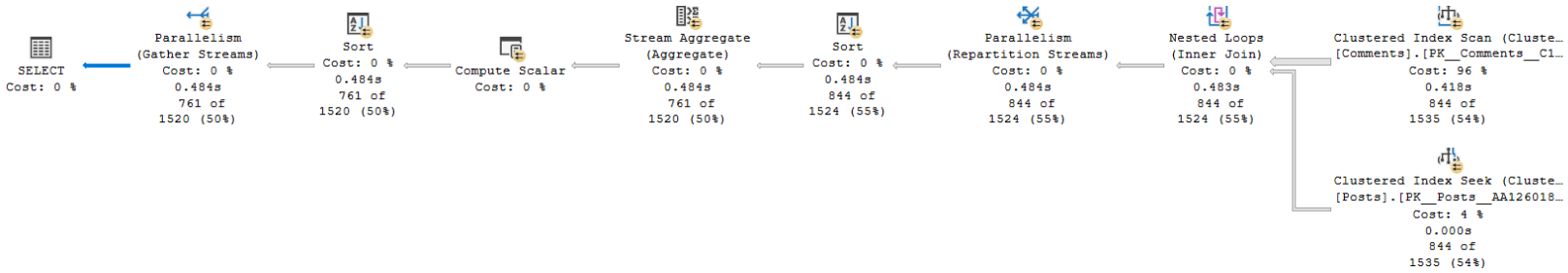


Table	Scan Count	Logical Reads	Physical Reads
Posts	0	6480	1
Worktable	0	0	0
Comments	9	163456	1
Worktable	0	0	0

Metric	CPU time	Elapsed time
SQL Server Execution Times:	156 ms	549 ms

Για την βελτιστοποίηση του επερωτήματος, μπορούμε να εξετάσουμε την δημιουργία ευρετηρίων στα γνωρίσματα που χρησιμοποιούνται στην συνθήκη **WHERE** και **GROUP BY**.

Το 1ο ευρετήριο για τον πίνακα **comments**, ευρετηριάζει τις εγγραφές σύμφωνα με το όρισμα **score** που υπάρχει στην συνθήκη **WHERE**, καθώς και συμπεριλαμβάνει το γνώρισμα **postId** ώστε να περιοριστούν περιττές ανακτήσεις δεδομένων.

Το 2ο ευρετήριο για τον πίνακα **posts**, ευρετηριάζει τις εγγραφές σύμφωνα με το όρισμα **postId** που υπάρχει στην συνθήκη ελέγχου, καθώς και συμπεριλαμβάνει το γνώρισμα **Title** ώστε να περιοριστούν οι περιττές ανακτήσεις δεδομένων.

```
CREATE INDEX IX_comments ON comments (score) INCLUDE
(PostId) ;
```

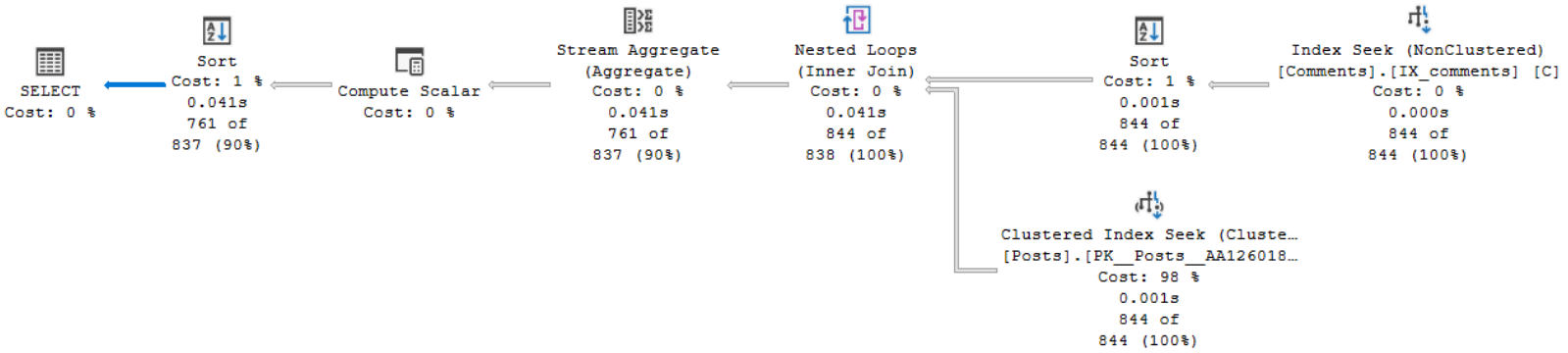


Table	Scan Count	Logical Reads	Physical Reads
Worktable	0	0	0
Posts	0	6223	1
Comments	1	5	2

Metric	CPU time	Elapsed time
SQL Server Execution Times:	31 ms	102 ms

Όπως παρατηρούμε από τα στατιστικά και το πλάνο εκτέλεσης, το ευρετήριο χρησιμοποιείται και μάλιστα αποδοτικά, καθώς μείωσε τον χρόνο επεξεργασίας, μείωσε τα **logical reads** από **163456** σε **5**. Επίσης όπως παρατηρείται στο πλάνο εκτέλεσης, αντικαταστάθηκε το **Clustered Index Scan** με **Index Seek**.

```
CREATE INDEX IX_post ON posts (PostId) INCLUDE (Title);
```

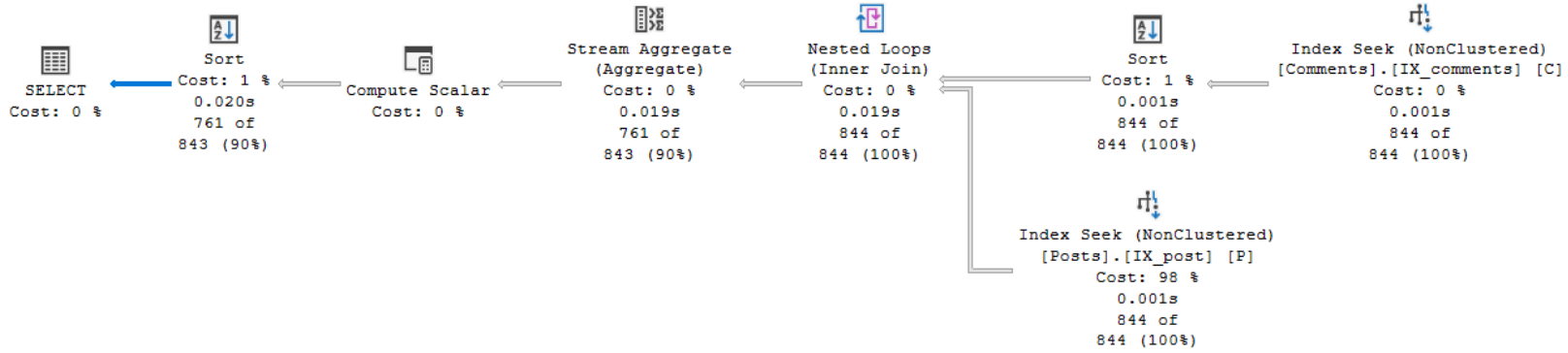


Table	Scan Count	Logical Reads	Physical Reads
Worktable	0	0	0
Posts	0	4511	1
Comments	1	5	2

Metric	CPU time	Elapsed time
SQL Server Execution Times:	0 ms	82 ms

Όπως παρατηρούμε από τα στατιστικά και το πλάνο εκτέλεσης, το ευρετήριο χρησιμοποιείται και μάλιστα αποδοτικά, καθώς μείωσε τον χρόνο επεξεργασίας, μείωσε τα **logical reads** από **6223** σε **4511**. Το 2ο ευρετήριο δεν έκανε τόσο μεγάλη διαφορά όσο το 1ο καθώς το ήδη υπάρχον (*πρωτεύον*) ήταν σχετικά καλό.