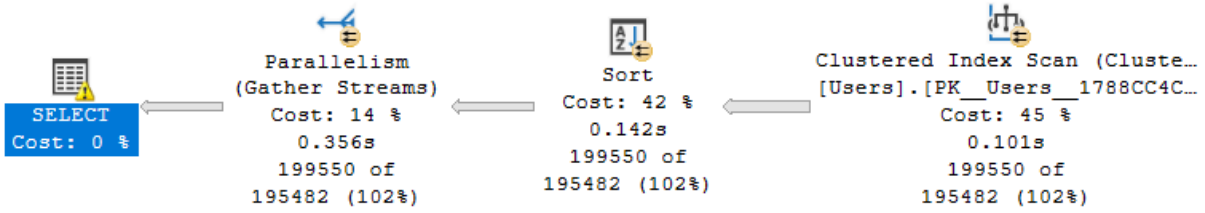
Συστήματα Διαχείρισης και Ανάλυσης Δεδομένων

1ο Project

Παντελίδης Ιπποκράτης – p3210150

Ζήτημα 1ο

Το ερώτημα του 1ου ζητουμένου μας δίνει το ακόλουθο πλάνο εκτέλεσης :



Παρατηρούμε ότι για τον πίνακα Users γίνεται Clustered Index Scan στο ευρετήριο που δημιουργεί αυτόματα ο SQL Server για τα πρωτεύοντα κλειδιά. Η πράξη αυτή καθώς και αυτή του sorting των εγγραφών είναι χρονοβόρες και επιδέχονται βελτιστοποιήσεις.

Στην συνέχεια βλέπουμε τα στατιστικά στοιχεία I/O και χρόνου εκτέλεσης του ερωτήματος πριν χρησιμοποιήσουμε κάποιο ευρετήριο ή γράψουμε το επερώτημα με εναλλακτικό τρόπο :

Table	Scan Count	Logical Reads	Physical Reads	Read-ahead Reads
Users	9	6001	2	5717
Worktable	0	0	0	0
Total	9	6001	2	5717

Action	CPU Time (ms)	Elapsed Time (ms)
SQL Server Execution Time	266	909
Parse and Compile Time	0	0
Total	266	909

Για την βελτιστοποίηση του επερωτήματος αρχικά θα αφαιρέσουμε την συνάρτηση YEAR() η οποία μπορεί να αποτρέψει την χρήση ευρετηρίων κατά την εκτέλεση οπότε θα αντικαταστήσουμε την εντολή `WHERE YEAR(CreationDate)=2010` με την ακριβώς ισοδύναμη της `WHERE CreationDate >= '2010-01-01' AND CreationDate < '2011-01-01'`

Στην συνέχεια θα χρησιμοποιήσουμε τα ακόλουθο ευρετήριο :

```
CREATE INDEX idx_users_creationdate_profileviews_displayname ON users (CreationDate, profileViews) INCLUDE (displayName);
```

Το ευρετήριο idx_users_creationdate_profileviews_displayname βοηθά στη βελτίωση του ερωτήματος, επιτρέποντας στον μηχανισμό της βάσης δεδομένων να εντοπίσει γρήγορα τις εγγραφές που πληρούν τις τιμές CreationDate που βρίσκονται μέσα στο χρονικό διάστημα που ορίζει το **WHERE clause**. Στη συνέχεια, τα αποτελέσματα ταξινομούνται βάσει των γνωρισμάτων CreationDate και των profileViews, και για αυτή την δουλειά χρησιμοποιείται επίσης το ευρετήριο για **αποτελεσματικότερη** αναζήτηση και **ταξινόμηση** των δεδομένων. Επιπλέον, η δήλωση **INCLUDE** στον ορισμό του ευρετηρίου καθορίζει ότι η στήλη display Name πρέπει να συμπεριληφθεί στο ευρετήριο. Αυτό σημαίνει ότι ο μηχανισμός της βάσης δεδομένων μπορεί να **ανακτήσει** τιμές της στήλης **displayName** απευθείας από το ευρετήριο, χωρίς να χρειάζεται να εκτελέσει μια ξεχωριστή λειτουργία lookup στον πίνακα Users.

Μετά την βελτιστοποίηση του επερωτήματος έχουμε το ακόλουθο πλάνο εκτέλεσης καθώς και τα εμφανώς βελτιωμένα στατιστικά I/O και χρόνου εκτέλεσης :

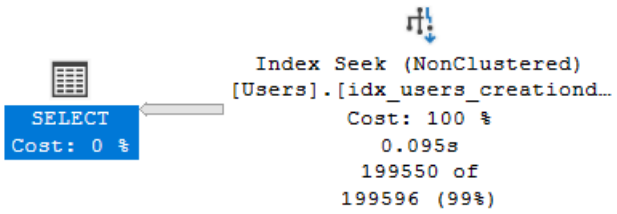


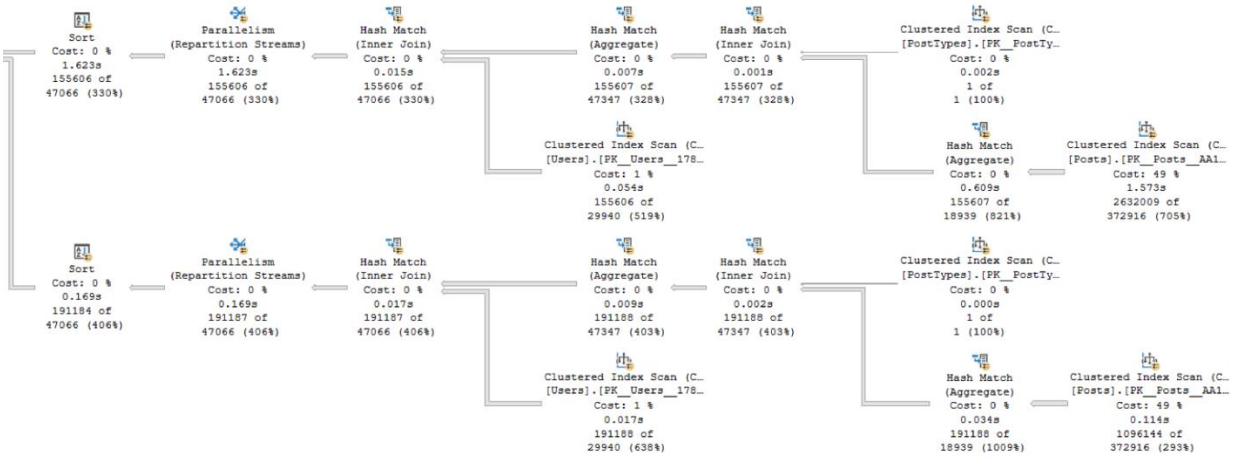
Table	Scan Count	Logical Reads	Physical Reads	Read-ahead Reads
Users	1	927	3	958
Worktable	0	0	0	0
Total	1	927	3	958

Action	CPU Time (ms)	Elapsed Time (ms)
SQL Server Execution Time	94	857
Parse and Compile Time	0	0
Total	94	857

Όπως φαίνεται παραπάνω το Clustered Index Seek που είχαμε προηγουμένως έχει αντικατασταθεί από ένα απλό Index Seek με πολύ μικρότερο κόστος, και βλέπουμε επίσης την σημαντική διαφορά στα logical reads και τα Read-ahead Reads.

Ζήτημα 2ο

Το ερώτημα του 2ου ζητουμένου μας δίνει το ακόλουθο πλάνο εκτέλεσης :



Μπορούμε να παρατηρήσουμε από το πλάνο εκτέλεσης ότι το Clustered Index Scan στον πίνακα Posts είναι πολύ κοστοβόρο οπότε θα πρέπει να το βελτιστοποιήσουμε.

Αυτά είναι τα στατιστικά στοιχεία I/O του ερωτήματος πριν χρησιμοποιήσουμε κάποιο ευρετήριο ή γράψουμε το επερώτημα με εναλλακτικό τρόπο :

Table	Scan Count	Logical Reads	Physical Reads	Read-Ahead Reads
Users	26	12002	2	5717
Posts	26	744699	2	368393
PostTypes	15	8	1	0
Worktable	0	0	0	0
Total	**67**	**756709**	**5**	**374110**

Αρχικά, θα μετασχηματίσουμε το επερώτημα που μας δόθηκε στο ακόλουθο έτσι ώστε να επωφεληθούμε από την χρήση των τελεστών EXISTS και NOT EXISTS έναντι του EXCEPT. Με αυτήν την αλλαγή αποφεύγεται ο έλεγχος για διπλότυπα και αυξάνεται η απόδοση και η ταχύτητα της εκτέλεσης. Επομένως, το ισοδύναμο query που προκύπτει είναι το εξής :

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

Στην συνέχεια θα χρησιμοποιήσουμε το ακόλουθο ευρετήριο :

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

Το επερώτημα της άσκησης αποτελείται από δύο υποερωτήματα, εκ των οποίο το ένα βρίσκει τα posts που δημιουργήθηκαν από έναν χρήστη και έχουν postTypeId ‘Question’ και το δεύτερο αυτά που δημιουργήθηκαν από τον ίδιο χρήστη και έχουν postTypeId ‘Answer’. Επομένως, το να φτιάξουμε ευρετήριο με τα πεδία ownerUserid και postTypeId βοηθάει τον μηχανισμό της βάσης δεδομένων να μην σκανάρει ολόκληρο τον πίνακα posts, μειώνοντας έτσι των αριθμό των logical reads που απαιτούνται για την εκτέλεση της ερώτησης.

Μετά την βελτιστοποίηση του επερωτήματος έχουμε το ακόλουθο πλάνο εκτέλεσης καθώς και τα εμφανώς βελτιωμένα στατιστικά I/O και χρόνου εκτέλεσης :

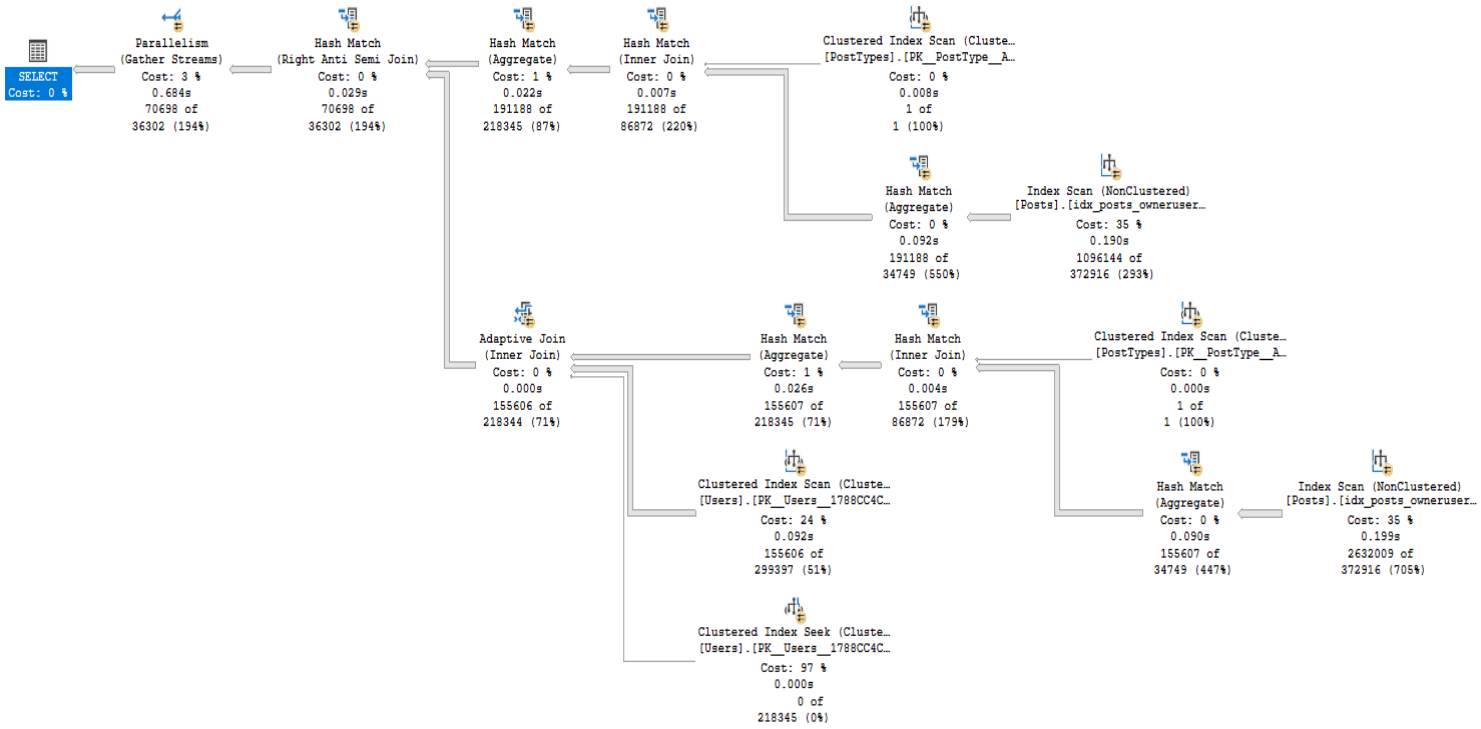


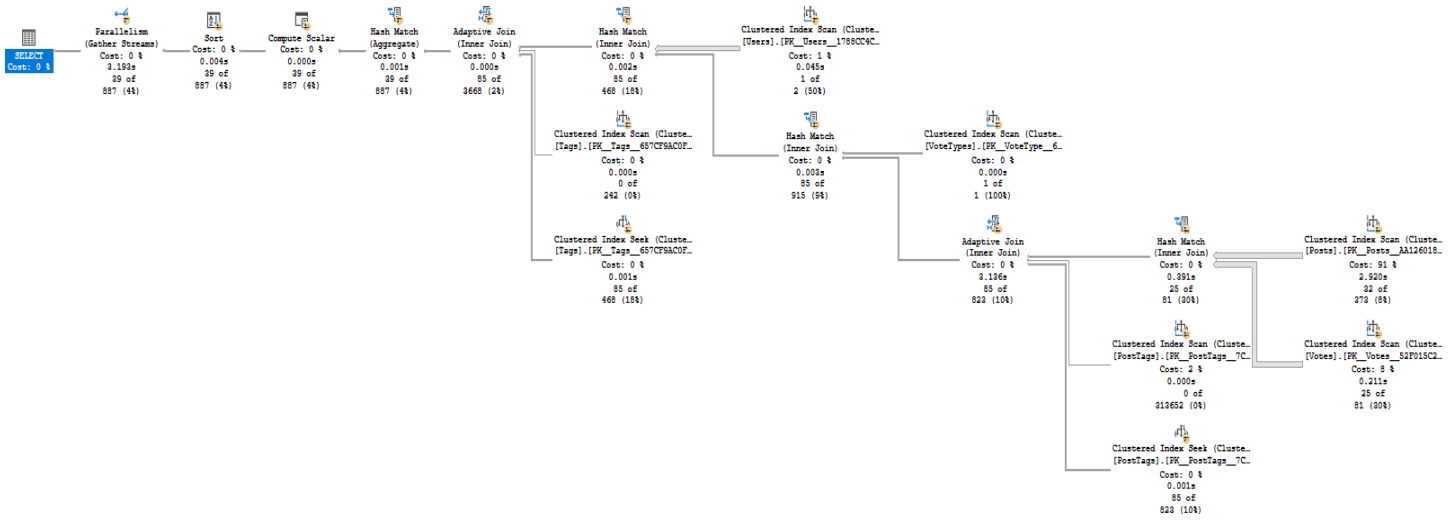
Table	Scan Count	Logical Reads	Physical Reads	Read-Ahead Reads
Users	9	5902	3	5712
Posts	18	14084	1	6946
PostTypes	8	8	1	0
Worktable	0	0	0	0
Total	35	20094	5	12658

Action	CPU Time (ms)	Elapsed Time (ms)
SQL Server Execution	2843	1492
Total	2843	1492

Με βάση τα παραπάνω στατιστικά στοιχεία και πλάνα μπορούμε να παρατηρήσουμε ότι το ερώτημα βελτιστοποιήθηκε σε πολύ μεγάλο βαθμό από την αρχική του μορφή, παρόλο που ακόμα μπορεί να κάνει αρκετά logical reads κατά την εκτέλεση του.

Ζήτημα 3ο

Το ερώτημα του 3ου ζητουμένου μας δίνει το ακόλουθο πλάνο εκτέλεσης :



Παρατηρούμε ότι οι πράξεις Clustered Index Scan στους πίνακες Posts και Votes (ειδικότερα στον Posts) είναι πολύ κοστοβόρες και για την βελτιστοποίηση τους θα φτιάξουμε ευρετήρια.

Τα στατιστικά I/O καθώς και του χρόνου εκτέλεσης χωρίς να έχουμε να παρέμβει στο αρχικό ερώτημα φαίνονται παρακάτω :

Table	Scan Count	Logical Reads	Physical Reads	Read-Ahead Reads
Tags	0	170	37	0
PostTags	24	72	25	0
Votes	9	29495	1	29068
Posts	9	370542	2	368382
Worktable	0	0	0	0
Total	42	400279	65	397450

Action	CPU Time (ms)	Elapsed Time (ms)
CPU Time	685	3339
SQL Server Exec	0	0
Total	**685**	**3339**

Δεν θα γράψουμε το query με εναλλακτικό τρόπο, ωστόσο θα χρησιμοποιήσουμε τα ακόλουθα 3 ευρετήρια για να βελτιστοποιήσουμε το ερώτημα :

```
CREATE INDEX idx_posts_owneruserid ON Posts (OwnerUserId);
CREATE INDEX idx_votes_postid ON Votes (PostId, VoteTypeId);
CREATE INDEX idx_users_displayname ON Users (DisplayName);
```

Ξεκινώντας από το τρίτο κατά σειρά ευρετήριο, δηλαδή αυτό που φτιάχνουμε στον πίνακα των users, μπορούμε να πούμε ότι η χρήση του είναι πολύ σημαντική για την βελτιστοποίηση του ερωτήματος, καθώς με την ύπαρξη του όταν αντιμετωπίζουμε ένα WHERE ερώτημα δεν χρειάζεται να διαβάζουμε ολόκληρο τον πίνακα των χρηστών, αλλά παραμόνο λίγες εγγραφές. Στην περίπτωση μας η εύρεση της εγγραφής που έχει στο πεδίο displayName την τιμή 'Dominik Weber' γίνεται πολύ γρήγορα και αποδοτικά.

Το πρώτο ευρετήριο χρησιμοποιείται ώστε να μπορέσουμε να εντοπίσουμε πολύ γρήγορα όλες τις εγγραφές του πίνακα Posts που η τιμή του πεδίου τους ownerUserid ταιριάζει με αυτή του 'Dominik Weber'. Με την ίδια λογική, η χρήση του ευρετηρίου μας βοηθάει να αποφύγουμε να διαβάσουμε ολόκληρο τον πίνακα Posts, κάτι που θα ήταν πολύ χρονοβόρο και κοστοβόρο, αλλά και να βρούμε με πολύ μεγάλη ταχύτητα τις εγγραφές που χρειαζόμαστε.

Τέλος, το δεύτερο ευρετήριο, το οποίο χτίζεται πάνω στα πεδία PostId και VoteTypeId, επιταχύνει την εύρεση των εγγραφών του πίνακα Votes των οποίων το PostId ταιριάζει με το id των posts που ανήκουν στον 'Dominik Weber', αλλά και έχουν στο πεδίο VoteTypeId την τιμή 'UpVote'. Το συγκεκριμένο ευρετήριο μας βοηθάει τόσο στο WHERE clause με την πρώτη συνθήκη τόσο και στο JOIN.

Μετά την βελτιστοποίηση του επερωτήματος έχουμε το ακόλουθο πλάνο εκτέλεσης καθώς και τα εμφανώς βελτιωμένα στατιστικά I/O και χρόνου εκτέλεσης :

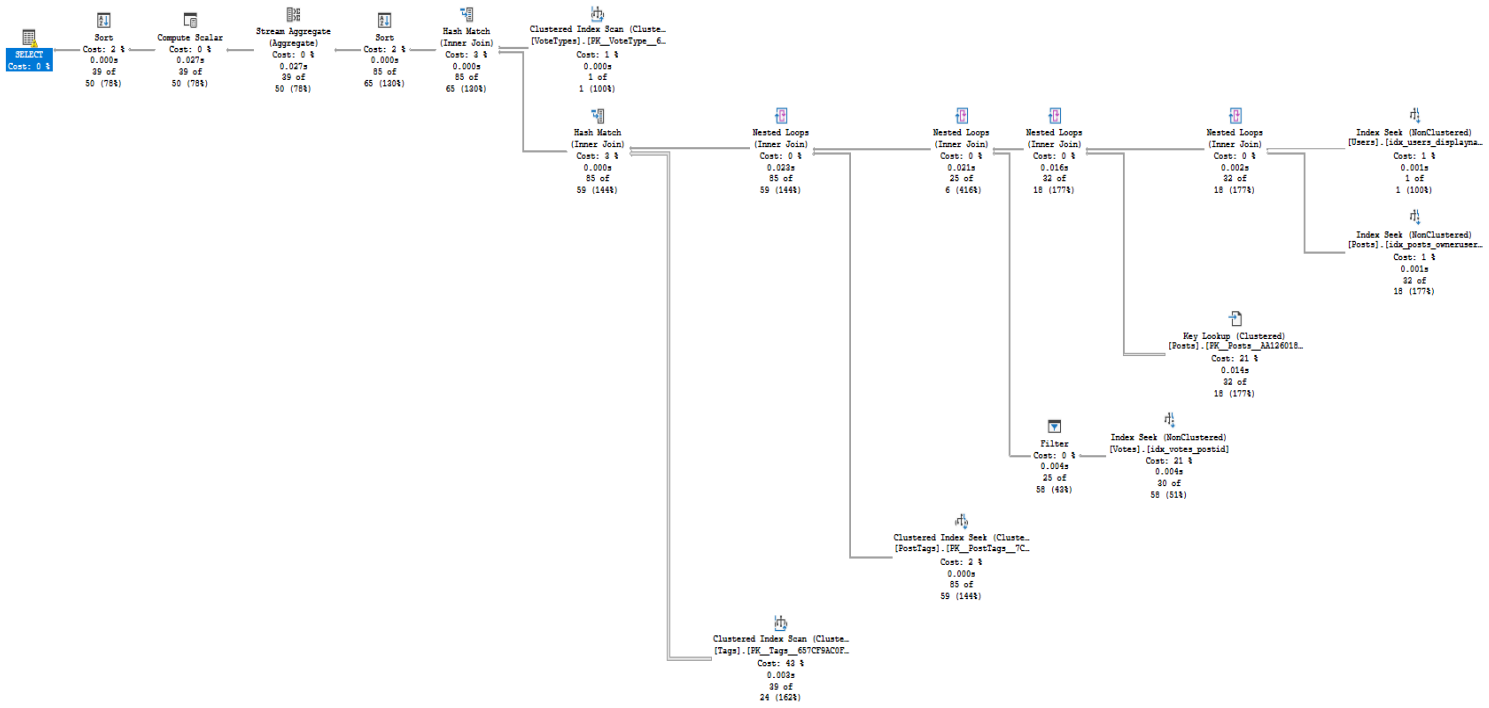


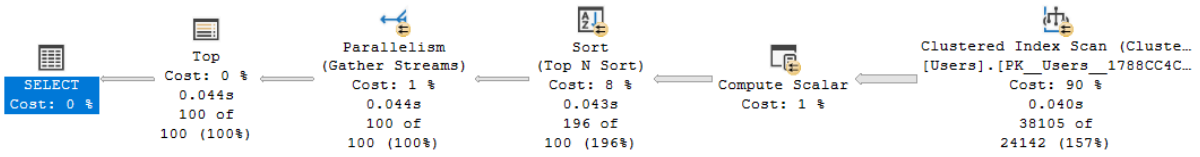
Table	Scan Count	Logical Reads	Physical Reads	Read-Ahead Reads
Tags	1	118	1	116
VoteTypes	1	2	1	0
PostTags	24	177	1	24
Votes	32	96	20	0
Posts	1	131	46	0
Users	1	3	3	0
Worktable	0	0	0	0
Total	**60**	**527**	**72**	**140**

Action	CPU Time (ms)	Elapsed Time (ms)
Action	0	64
SQL Server Execution	0	0
Total	**0**	**64**

Εμφανής είναι η βελτιστοποίηση του συγκεκριμένου ερωτήματος, καθώς υπάρχει τρομερή μείωση στις τιμές των στατιστικών I/O και πιο συγκεκριμένα σε αυτές των logical reads και read-ahead reads. Έχουμε φτάσει από εξαψήφιο αριθμό μόλις σε τριψήφιο. Επίσης, όπως μπορούμε να δούμε από το execution plan έχουμε καταφέρει να αντικαταστήσουμε το χρονοβόρο Clustered Index Scan σε Index Seek για τους πίνακες Posts, Votes και Users.

Ζήτημα 4ο

Το ερώτημα του 4ου ζητουμένου μας δίνει το ακόλουθο πλάνο εκτέλεσης :



Βλέπουμε πάλι ότι η πράξη που καθυστερεί περισσότερο είναι αυτή του Clustered Index Scan στον πίνακα Users για αυτό και θα προσπαθήσουμε να την βελτιστοποιήσουμε.

Παρακάτω βλέπουμε τα στατιστικά στοιχεία I/O και χρόνου εκτέλεσης του ερωτήματος χωρίς την χρήση ευρετηρίων :

Table	Scan Count	Logical Reads	Physical Reads	Read-Ahead Reads
Users	9	6001	3	5712
Total	9	6001	3	5712

Action	CPU Time (ms)	Elapsed Time (ms)
SQL Server Execution	31	114
Total	31	114

Για το συγκεκριμένο ζήτημα θα χρησιμοποιήσουμε δύο εναλλακτικά ευρετήρια για την βελτιστοποίηση του ερωτήματος και θα επιλέξουμε εκ των δύο το καταλληλότερο, δηλαδή αυτό που βελτιστοποιεί καλύτερα το επερώτημα.

1ο Ευρετήριο :

Το πρώτο ευρετήριο που θα χρησιμοποιήσουμε είναι το :

```
CREATE INDEX idx_reputation_upvotes_downvotes ON Users (Reputation, UpVotes) INCLUDE (DownVotes);
```

Το ευρετήριο είναι πολύ αποδοτικό και επιταχύνει σε μεγάλο βαθμό το WHERE clause, καθώς με την χρήση μπορούμε να βρούμε πολύ εύκολα και χωρίς να εξετάσουμε πολλά rows, αυτά που πληρούν τις συνθήκες του WHERE σχετικά με τα πεδία Reputation και UpVotes. Παράλληλα με το INCLUDE, το πεδίο DownVotes αποθηκεύεται μαζί με τα άλλα δύο στο ευρετήριο και βοηθάει την εντολή SELECT μειώνοντας τον αριθμό των I/O που απαιτούνται για να πάρουμε τις στήλες του πίνακα που χρειάζονται. Μπορεί το συγκεκριμενο ευρετήριο να μην βοηθάει άμεσα στο sorting των εγγραφών, καθώς το πεδίο με το οποίο κάνουμε την ταξινόμηση είναι υπολογιζόμενο [Ratio %], ωστόσο βοηθάει σημαντικά στο να βελτιστοποιήσουμε τις συνιστώσες αυτού του πεδίου.

Μετά την βελτιστοποίηση του επερωτήματος έχουμε το ακόλουθο πλάνο εκτέλεσης καθώς και τα εμφανώς βελτιωμένα στατιστικά I/O και χρόνου εκτέλεσης :

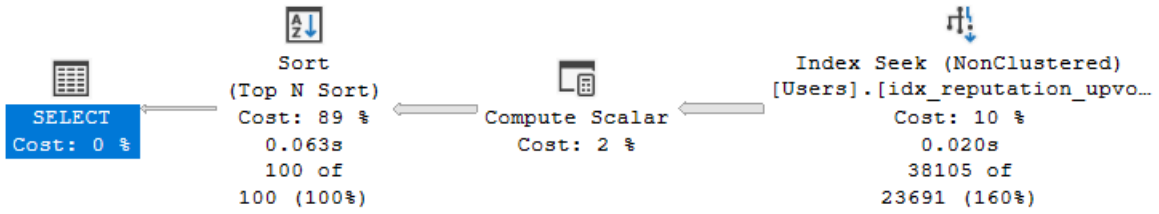


Table	Scan Count	Logical Reads	Physical Reads	Read-Ahead Reads
Users	1	156	3	152
Total	**1**	**156**	**3**	**152**

Action	CPU Time (ms)	Elapsed Time (ms)
SQL Server Execution	15	151
SQL Server Parse and Compile	0	0
Total	**15**	**151**

Μετά την βελτιστοποίηση του ερωτήματος, πέρα από την σημαντική μείωση των στατιστικών μπορούμε να παρατηρήσουμε ότι βελτιώθηκε σε πολύ μεγάλο βαθμό η χρονοβόρα πράξη του Clustered Index Scan στον πίνακα των Users και έχει αντικατασταθεί με το πιο γρήγορο Index Seek.

2ο Ευρετήριο :

Το δεύτερο ευρετήριο που θα χρησιμοποιήσουμε είναι το :

```
CREATE INDEX idx_upvotes_reputation_downvotes ON Users (UpVotes, Reputation) INCLUDE (DownVotes);
```

Το συγκεκριμένο ευρετήριο μοιάζει πάρα πολύ με το προηγούμενο και διαφέρουν μόνο στην σειρά των γνωρισμάτων στα οποία χτίζεται το ευρετήριο. Δηλαδή αντί να δηλώνονται με σειρά (UpVotes, Reputation), εδώ έχουμε το (Reputation, UpVotes). Βελτιστοποιεί με τον ίδιο ακριβώς τρόπο το ερώτημα βοηθώντας το ίδιο το WHERE, το SELECT και την ταξινόμηση.

Μετά την βελτιστοποίηση του επερωτήματος έχουμε το ακόλουθο πλάνο εκτέλεσης καθώς και τα εμφανώς βελτιωμένα στατιστικά I/O και χρόνου εκτέλεσης :

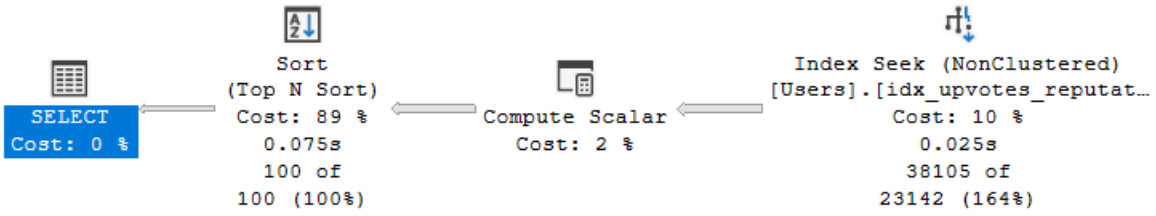


Table	Scan Count	Logical Reads	Physical Reads	Read-Ahead Reads
Users	1	150	3	146
Total	**1**	**150**	**3**	**146**

Action	CPU Time (ms)	Elapsed Time (ms)
SQL Server Execution Times	0	133
SQL Server Parse and Compile	0	0
Total	**0**	**133**

Παρατηρούμε ότι τα στατιστικά είναι σχεδόν ίδια με αυτά προηγουμένως, απλά με μία πολύ μικρή μείωση.

Επιλογή Καταλληλότερου Ευρετηρίου

Γενικότερα όταν φτιάχνουμε ένα ευρετήριο με πολλαπλά γνωρίσματα θεωρείται σαν καλή πρακτική να βάζουμε σαν πρώτο γνώρισμα αυτό που έχει το πιο επιλεκτικό φίλτρο στο ερώτημα που θέλουμε να βελτιστοποιήσουμε. Η έννοια 'επιλεκτικό' έχει να κάνει με το πόσες γραμμές επιστρέφονται από το φίλτρο, επομένως πιο επιλεκτικό είναι αυτό που επιστρέφει λιγότερες γραμμές σαν αποτέλεσμα. Στην περίπτωση μας, το πρώτο φίλτρο από τα δύο (Reputation > 1000) επιστρέφει 55,443 τιμές και 15,280 μοναδικές τιμές, ενώ το δεύτερο φίλτρο (UpVotes > 100) μας φέρνει συνολικά 52,759 τιμές από τις οποίες οι 3,996 είναι μοναδικές. Παρατηρούμε λοιπόν ότι το δεύτερο φίλτρο είναι πιο επιλεκτικό οπότε προτιμάμε να βάλουμε το πεδίο αυτού πρώτο σε σειρά στην δήλωση του ευρετηρίου. Επομένως, το καταλληλότερο ευρετήριο στο οποίο καταλήγουμε, συνδυάζοντας αυτά τα στοιχεία καθώς και τα ελαφρώς βελτιωμένα στατιστικά είναι το δεύτερο, δηλαδή το :

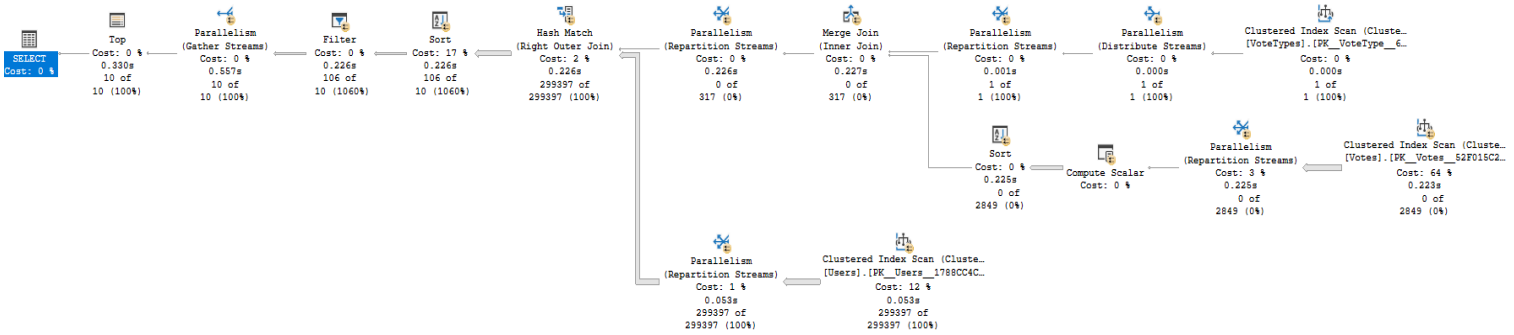
```
CREATE INDEX idx_upvotes_reputation_downvotes ON Users (UpVotes, Reputation) INCLUDE (DownVotes);
```

Ζήτημα 5ο

Για το 5ο ζητούμενο επιλέγουμε το ερώτημα που μας δίνει τους *top 10 χρήστες με το μεγαλύτερο reputation που δεν έχουν λάβει κανένα downVote τις τελευταίες 30 μέρες.* Το ερώτημα που παίρνει από την βάση αυτή την πληροφορία είναι το ακόλουθο :

```
SELECT TOP 10 u.UserId, u.displayName, u.Reputation
FROM Users u
LEFT JOIN (
    SELECT v.UserId, v.PostId
    FROM Votes v
    JOIN VoteTypes vt ON v.VoteTypeId = vt.VoteTypeId
    WHERE vt.VoteTypeName = 'DownVote'
        AND v.CreationDate >= DATEADD(day, -30, GETDATE())
) downVotes
ON u.UserId = downVotes.UserId
WHERE downVotes.UserId IS NULL
ORDER BY u.Reputation DESC;
```

Το ερώτημα μας δίνει το ακόλουθο πλάνο εκτέλεσης :



Βλέπουμε ότι και σε αυτό το ζήτημα οι πιο κοστοβόρες πράξεις είναι αυτές του Clustered Index Scan στους πίνακες Users και Votes και αυτές θα προσπαθήσουμε με την χρήση ευρετηρίων να βελτιστοποιήσουμε.

Παρακάτω βλέπουμε τα στατιστικά στοιχεία I/O και χρόνου εκτέλεσης του ερωτήματος χωρίς την χρήση ευρετηρίων :

Table	Scan Count	Logical Reads	Physical Reads	Read-Ahead Reads
VoteTypes	1	2	1	0
Votes	9	29495	1	29069
Users	9	6001	3	5712
Worktable	0	0	0	0
Workfile	0	0	0	0
Total	**19**	**35498**	**5**	**34781**

Action	CPU Time (ms)	Elapsed Time (ms)
SQL Server Execution	636	377
SQL Server Parse and Compile	0	0
Total	**636**	**377**

Για την βελτιστοποίηση του ερωτήματος θα χρησιμοποιήσουμε τα 2 ευρετήρια παρακάτω :

```
CREATE INDEX idx_votes_creationdate_userid_votetypeid ON Votes (CreationDate) INCLUDE (userId, VoteTypeId);
CREATE INDEX idx_users_reputation_userId_displayName ON Users (Reputation DESC) INCLUDE (userId, displayName);
```

εκ των οποίων το πρώτο βελτιστοποιεί κυρίως το εμφωλευμένο επερώτημα που βρίσκεται στο εσωτερικό του LEFT JOIN και πιο συγκεκριμένα μας βοηθάει να βρούμε πολύ πιο εύκολα και γρήγορα τις γραμμές που ικανοποιούν τις συνθήκες του WHERE clause. Παράλληλα βάζοντας το userId στο INCLUDE κρατάμε πληροφορία για τους χρήστες στο ευρετήριο και μπορούμε να την πάρουμε από εκεί χωρίς να διαβάσουμε ολόκληρο τον πίνακα των users όταν την χρειαστούμε. Από την άλλη το δεύτερο ευρετήριο μας βοηθάει στην αποδοτική ταξινόμηση με το πεδίο Reputation καθώς πάλι με την χρήση του INCLUDE να κρατήσουμε πληροφορία για το userId και το displayName του χρήστη και να μην χρειαστεί να σκανάρουμε ολόκληρο τον πίνακα Users για να τα βρούμε κατά το SELECT.

Μετά την βελτιστοποίηση του επερωτήματος έχουμε το ακόλουθο πλάνο εκτέλεσης καθώς και τα εμφανώς βελτιωμένα στατιστικά I/O και χρόνου εκτέλεσης :

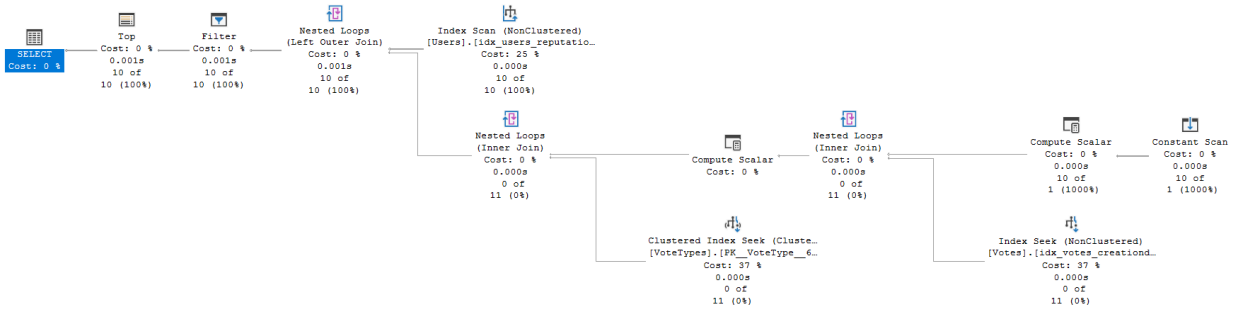


Table	Scan Count	Logical Reads	Physical Reads	Read-Ahead Reads
Worktable	0	0	0	0
Votes	10	30	3	0
Users	1	3	3	0
Total	**11**	**33**	**6**	**0**

Action	CPU Time (ms)	Elapsed Time (ms)
SQL Server Execution	0	32
SQL Server Parse and Compile	0	0
Total	**0**	**32**

Όπως μπορούμε να δούμε από το πλάνο εκτέλεσης τα ευρετήρια μας χρησιμοποιούνται κανονικά και έχουν καταφέρει να αντικαταστήσουν τις κοστοβόρες πράξεις Clustered Index Scan που είχαμε προηγουμένως σε Index Seek για τον πίνακα Votes και σε Index Scan για τον Users, με μέγεθος 10 εγγραφών. Παρατηρούμε επίσης ότι τα στατιστικά στοιχεία I/O και πιο συγκεκριμένα τα logical reads έχουν μειωθεί σημαντικά πολύ.