Τζένη Μπολένα
3170117

Πρώτη Εργασία

Εισαγωγή

Παρακάτω αναφέρομαι σε έννοιες όπως physical reads, read-ahead reads και logical reads. Στα screenshots μαζι με αυτά εμφανίζονται και scan count. Ας δούμε τι είναι το κάθε ένα.

- **Scan count:** ο αριθμός των index ή table scan στον πίνακα.
- **Logical reads:** ο αριθμός των σελίδων που διαβάστηκαν από το data cache. Επειδή κάθε σελίδα ανεξάρτητα του αν ήρθε από τον δίσκο ή όχι διαβάζεται από το data cache τα logical reads στην ουσία είναι ο συνολικός αριθμος access σε σελίδες.
- **Physical reads:** ο αριθμός των σελίδων που διαβάστηκαν από τον δίσκο. Physical read έχουμε όταν πάμε να διαβάσουμε δεδομένα αλλά δεν υπάρχει κάτι στο data cache και πρέπει να περιμένουμε να έρθει από τον δίσκο.
- **Read-ahead reads:** ο optimizer “προνοεί” και φέρνει κάποιες σελίδες από τον δίσκο στο data cache, ανεξάρτητα αν αυτά θα χρησιμοποιηθούν ή όχι. Γίνεται παράλληλα με όταν γίνεται process το query. Έτσι με αυτόν τον μηχανισμό μπορούν να μειώνονται και τα physical reads.

Αν καταφέρουμε να μειώσουμε αυτούς του αριθμό, με πολύ σημαντικό ρόλο να παίζουν physical, read-aheads, logical τότε τα index είναι αποδοτικά.

Ζήτημα Πρώτο

1.

Το index που θα επιλέξουμε είναι αυτό στα γνωρίσματα pyear και title.

```
create nonclustered index Index_movies_pyear_title
on movies (pyear, title)
```

Αρχικά επιλέγουμε το pyear καθώς είναι αυτό που χρησιμοποιείται από το order by που περιέχει ταξινόμηση η οποία είναι επίπονη διαδικασία, καθώς επίσης χρησιμοποιείται και στο filtering στο between. Έπειτα χρησιμοποιούμε το title ως δεύτερο στο order by και επίσης στο select.
Η αποδοτικότητα του index αυτού φαίνεται στην μείωση που επιφέρει στα logical reads και στα read-ahead reads.

BEFORE INDEX (τα ερωτήματα με την σειρά που δίνονται)

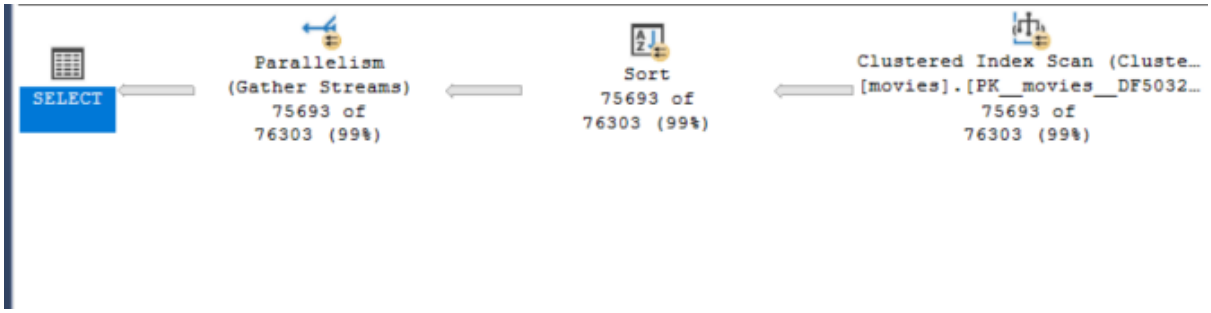
```
Table 'movies'. Scan count 1, logical reads 1918, physical reads 2, read-ahead reads 1917,

Table 'movies'. Scan count 1, logical reads 1918, physical reads 2, read-ahead reads 1917,

Table 'movies'. Scan count 5, logical reads 2012, physical reads 2, read-ahead reads 1917,
```

AFTER INDEX (τα ερωτήματα με την σειρά που δίνονται)

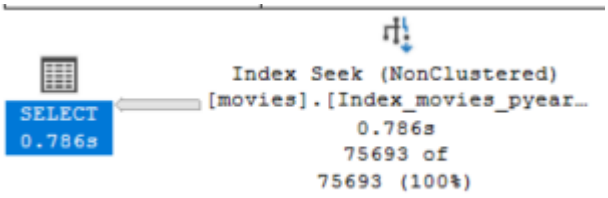
```
Table 'movies'. Scan count 1, logical reads 351, physical reads 2, read-ahead reads 361,

Table 'movies'. Scan count 1, logical reads 351, physical reads 2, read-ahead reads 361,

Table 'movies'. Scan count 1, logical reads 351, physical reads 2, read-ahead reads 361,
```

Και πλέον βλέπουμε(για παράδειγμα στο τρίτο query) ότι έχουμε χρήση index seek.

Before Index



After Index



ΕΞΤΡΑ

AFTER INDEX (clear buffer και έπειτα δημιουργία index)

```
Table 'movies'. Scan count 1, logical reads 351, physical reads 0, read-ahead reads 5,

Table 'movies'. Scan count 1, logical reads 351, physical reads 0, read-ahead reads 5,

Table 'movies'. Scan count 1, logical reads 351, physical reads 0, read-ahead reads 5,
```

2.

Τρέχοντας τα queries σε batch(δέσμη) βλέπουμε ότι το πρώτο έχει σχετικό κόστος 41% και το δεύτερο 59%.

Estimated query progress:100%	Query 1: Query cost (relative to the batch): 41% select mid, count(rating) from user_movies group by mid order by mid

Estimated query progress:100%	Query 2: Query cost (relative to the batch): 59% END BEGIN select userid, count(rating) from user_movies group by userid order by userid

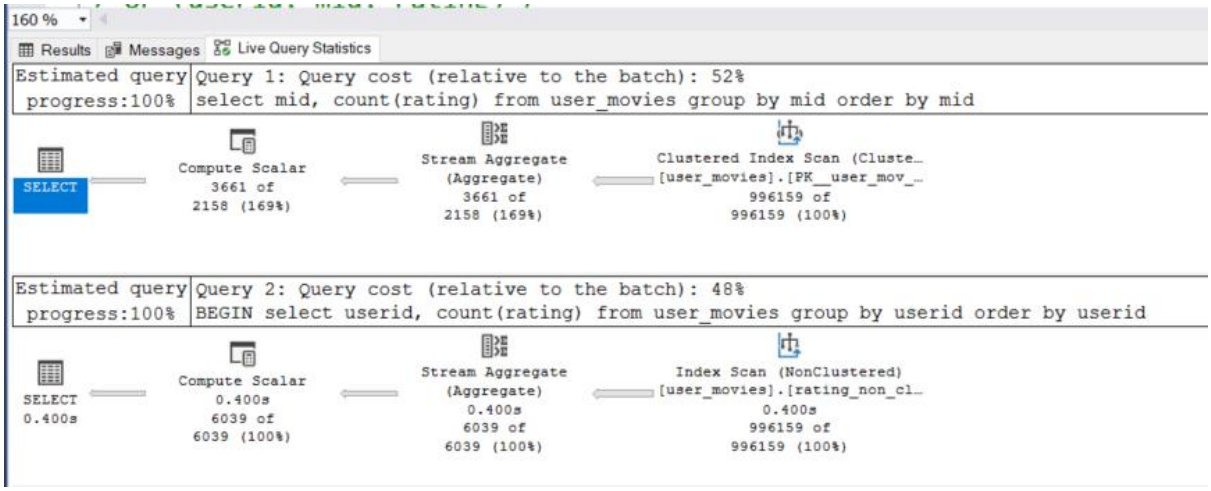
Επιπλέον στον πίνακα(user_movies) έχουμε primary key το mid, userid. Επομένως το column mid είναι ήδη ταξινομημένο.
Για τους λόγους αυτούς θα επιλέξουμε στο index το column userid το οποίο έχει και μπροστά order by. Ακολουθείται από το rating.

```
create nonclustered index Index_userMovies_userId_rating
on user_movies ( userid, rating)
```

BEFORE INDEX (τα ερωτήματα με την σειρά που δίνονται)

```
Table 'user_movies'. Scan count 1, logical reads 2601, physical reads 2, read-ahead reads 2603,
Table 'user_movies'. Scan count 5, logical reads 2733, physical reads 2, read-ahead reads 2603,
```

AFTER INDEX (τα ερωτήματα με την σειρά που δίνονται)



Με αυτόν τρόπο επίσης βλέπουμε και καλύτερη εξισορρόπηση στον χρόνο που παίρνουν τα ερωτήματα.

```
Table 'user_movies'. Scan count 1, logical reads 2601, physical reads 2, read-ahead reads 2603,
Table 'user_movies'. Scan count 1, logical reads 2230, physical reads 1, read-ahead reads 2244,
```

ΕΞΤΡΑ
AFTER INDEX (clear buffer και έπειτα δημιουργία index)

```
 (3661 rows affected)
 Table 'user_movies'. Scan count 1, logical reads 2601, physical reads 0, read-ahead reads 5,

(6039 rows affected)
Table 'user_movies'. Scan count 1, logical reads 2230, physical reads 0, read-ahead reads 15,
```

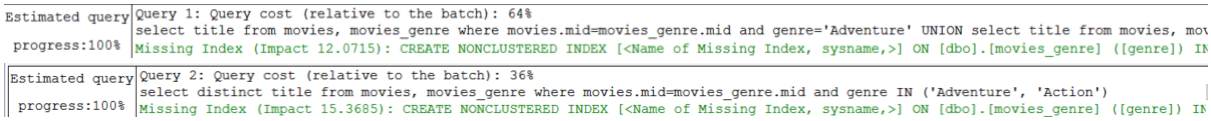
Ζήτημα Δεύτερο

1.

Το αποδοτικότερο ερώτημα σε SQL που έγραψα είναι:

```
select distinct title
 from movies, movies_genre
 where movies.mid=movies_genre.mid and genre IN ('Adventure', 'Action')
```

Τρέχοντας τα queries σε batch(δέσμη) βλέπουμε ότι το πρώτο έχει σχετικό κόστος 64% και το δεύτερο που μόλις έγραψα 36% κάνοντας το πιο αποδοτικό. Σε αυτό κυρίως οφείλεται ότι το αρχικό query κάνει τρία merge join(2 inner join και ένα union) όπως και 2 sort και 2 sort(distinct). Το νέο και πιο αποδοτικό query κάνει ένα hash match(inner join) και ένα sort(distinct).



Μπορούμε ωστόσο να κάνουμε και άλλο optimize με την χρήση index.
Ο πίνακας movies_genre δεν έχει ούτε primary key ούτε κάνει χρήση κάποιου άλλου index. Θα επιλέξουμε το γνώρισμα(column) genre που χρησιμοποιείται στο IN για να επιτύχουμε μεγαλύτερη ταχύτητα. Θα το ακολουθήσει το column mid που χρησιμοποιείται στο join.

```
CREATE NONCLUSTERED INDEX Index_moviesGenre_genre_mid
ON movies_genre (genre, mid)
```

Στον πίνακα movies θα χρησιμοποιήσουμε σαν index το title το οποίο και γίνεται select.

```
CREATE NONCLUSTERED INDEX Index_movies_title
ON movies ( title)
```

Before Index για το νέο query

```
Table 'movies_genre'. Scan count 5, logical reads 1123, physical reads 0, read-ahead reads 1123
Table 'movies'. Scan count 5, logical reads 2012, physical reads 2, read-ahead reads 1917, lob
```

After Index για το νέο query

```
Table 'movies_genre'. Scan count 9, logical reads 95, physical reads 1, read-ahead reads 84,
Table 'movies'. Scan count 5, logical reads 1478, physical reads 1, read-ahead reads 1418,
```

ΕΞΤΡΑ

AFTER INDEX (clear buffer και έπειτα δημιουργία index)

```
Table 'movies_genre'. Scan count 9, logical reads 95, physical reads 0, read-ahead reads 0,
Table 'movies'. Scan count 5, logical reads 1478, physical reads 0, read-ahead reads 19, lob
```

2.

Λύση Query 1

```
select title from
    (select mid from
        (
        select  moviesAll.mid, moviesAll.movieIdCount, moviesMenOne.genderCount from
            (select gender, mid , count(gender) as genderCount from
                actors, roles
                where roles.aid = actors.aid and gender='M'
                group by mid, gender) moviesMenOne
            inner join
            (select mid , count(mid) as movieIdCount from
                actors, roles
                where roles.aid = actors.aid
                group by mid) moviesAll on moviesMenOne.mid = moviesAll.mid
        ) moviesAllCount where moviesAllCount.genderCount = moviesAllCount.movieIdCount)
        moviesOnlyMen, movies
    where moviesOnlyMen.mid = movies.mid
```

Λύση Query 2

```
select title from
    (select distinct mid from
            (select mid from
                actors, roles
                where roles.aid = actors.aid and gender='M') moviesMenOne
        where mid NOT IN
            (select  mid from
                actors, roles
                where roles.aid = actors.aid and gender='F')) moviesOnlyMen, movies
    where moviesOnlyMen.mid = movies.mid
```

Αν τρέξουμε τα δύο αυτα queries σε batch βλέπουμε ότι το query1 παίρνει 56% και το query2 44%. Άρα το query2 είναι πιο γρήγορο και αποδοτικό. Πάμε να κάνουμε και έξτρα optimization προσθέτοντας indexes.

Indexes

Στον πίνακα actors θα επιλέξουμε να βάλουμε index το γνώρισμα gender το οποίο στην επιλογή του έχει συνθήκη.

```
create nonclustered index Index_actors_gender
on actors ( gender)
```

Στον πίνακα roles θα επιλέξουμε να βάλουμε index το γνώρισμα aid το οποίο στην επιλέγεται στην ισότητα του join. Ο πίνακας roles δεν έχει primary key και ούτε κάποιο άλλο index. Η προσθήκη του aid σε index είναι αναγκαία για πιο αποδοτικό execution.

```
create nonclustered index Index_roles_aid
on roles (aid)
```

Στον πίνακα movies θα επιλέξουμε να βάλουμε index το γνώρισμα title που γίνεται project(π title).

```
create nonclustered index Index_movies_title
on movies (title)
```

Statistics Before Index

Query 1

```
Table 'actors'. Scan count 10, logical reads 7060, physical reads 2, read-ahead reads 3362,
Table 'roles'. Scan count 10, logical reads 8978, physical reads 1, read-ahead reads 4424,
Table 'movies'. Scan count 5, logical reads 2012, physical reads 2, read-ahead reads 1917,
```

Query 2

```
Table 'actors'. Scan count 5, logical reads 334367, physical reads 2, read-ahead reads 3362,
Table 'roles'. Scan count 10, logical reads 8978, physical reads 1, read-ahead reads 4423,
Table 'movies'. Scan count 0, logical reads 125457, physical reads 242, read-ahead reads 0,
```

Statistics After Index

Αν τρέξουμε τα δύο αυτα queries ξανα σε batch αφού τώρα έχουμε προσθέσει τα index. Βλέπουμε ότι **το query1 παίρνει 62% και το query2 παίρνει 38%.**

Query2 stats

```
Table 'movies'. Scan count 0, logical reads 125457, physical reads 242, read-ahead reads 0,
Table 'roles'. Scan count 2, logical reads 3206, physical reads 2, read-ahead reads 1910,
Table 'actors'. Scan count 2, logical reads 1123, physical reads 2, read-ahead reads 1114,
```

ΕΞΤΡΑ

AFTER INDEX (clear buffer και έπειτα δημιουργία index)

Query 2

```
Table 'movies'. Scan count 0, logical reads 125457, physical reads 0, read-ahead reads 0,
Table 'actors'. Scan count 1, logical reads 331228, physical reads 0, read-ahead reads 11,
Table 'roles'. Scan count 2, logical reads 3802, physical reads 0, read-ahead reads 19,
```

Γενικά σχόλια:

Η αποφυγή των group by στο query2, η χρήση not in και τα λιγότερα joins το καθιστούν πιο γρήγορο από το query1. Η χρήση index βοηθάει στο optimization των queries.

Ζήτημα Τρίτο

Query

“Εμφάνισε τον συνολικό αριθμό κριτικών ανά ηλικία χρηστών και την ηλικία, για ταινίες είδους Περιπέτειας όπου η ηλικία του χρήστη είναι από 18 έως και 30 χρονών. ”
(**Χρησιμότητα query:** θα μπορούσε να χρησιμοποιηθεί από ένα νέο production ταινίας περιπέτειας για να δο΄νε που να στοχεύουν την διαφημιστική καμπάνια. Μην ξεχνάμε ότι το age και genre θα μπορούσαμε να τα δώσουμε παραμετρικά και άρα να χρησιμεύσει και σε άλλες περιπτώσεις ταινιών.)

```
select age, count(age) as countPerAge from movies_genre,
(select users.userid, mid, age from users, user_movies
where users.userid = user_movies.userid  and age between 18 and 45) userRatingMovies
where userRatingMovies.mid = movies_genre.mid and genre = 'Adventure'
group by age
```

Before Index

```
Table 'users'. Scan count 5, logical reads 82, physical reads 1, read-ahead reads 34, lob logic
Table 'movies_genre'. Scan count 5, logical reads 1123, physical reads 0, read-ahead reads 1123,
Table 'user_movies'. Scan count 5, logical reads 2733, physical reads 2, read-ahead reads 2603,

 SQL Server Execution Times:
   CPU time = 279 ms,  elapsed time = 218 ms.

SQL Server Execution Times:
   CPU time = 249 ms,  elapsed time = 147 ms.
```

Index

```
create nonclustered index Index_userMovies_userid
on user_movies (userid) /*new*/

create nonclustered index Index_users_age
on users (age) /*new index*/

create nonclustered index Index_moviesGenre_genre_mid
on movies_genre (genre, mid) /*already have it*/
```

- ➔ Χρήση του Index_userMovies_userid καθώς το userid χρησιμοποιείται στο join και ενώ υπάρχει στο primary key έρχεται ως δεύτερο γνώρισμα.
- ➔ Χρήση του Index_users_age καθώς το age χρησιμοποιείται στο group by και στο between.
- ➔ Χρήση του Index_moviesGenre_genre_mid καθώς το genre χρησιμοποιείται για filtering και το mid στο join. (Ο πίνακας genre δεν έχει primary key.)

After Index

```
Table 'users'. Scan count 5, logical reads 19, physical reads 1, read-ahead reads 13, lob logica
Table 'movies_genre'. Scan count 5, logical reads 40, physical reads 1, read-ahead reads 35, lob
Table 'user_movies'. Scan count 5, logical reads 1817, physical reads 3, read-ahead reads 1741,
```

ΕΞΤΡΑ
AFTER INDEX (clear buffer και έπειτα δημιουργία index)

```
Table 'users'. Scan count 5, logical reads 19, physical reads 0, read-ahead reads 0, lob logi
Table 'movies_genre'. Scan count 4, logical reads 40, physical reads 0, read-ahead reads 0, l
Table 'user_movies'. Scan count 5, logical reads 1817, physical reads 0, read-ahead reads 14,

SQL Server Execution Times:
  CPU time = 107 ms,  elapsed time = 103 ms.
```

Query

“Εμφάνισε το ονοματεπώνυμο των σκηνοθετών, μαζί με αριθμό ταινιών, που την δεκαετία του 90’ έχουν σκηνοθετήσει τουλάχιστον 2 ταινίες με κατάταξη ταινίας μεγαλύτερη ή ίση του 9, σε αύξουσα σειρά με βάση το επίθετο και μετά το όνομα.”
(**Χρησιμότητα query:** θα μπορούσε να χρησιμοποιηθεί από site όπως imdb για να δείχνει στατιστικά κτλ. Δεδομένου κιολας ότι το pyear και mrank μπορούν να δοθούν παραμετρικά.)

```
select  lastName, firstName, dirMoviesCount from directors,
    (select  did, count(did) as dirMoviesCount from movies, movie_directors
    where movies.mid = movie_directors.mid and pyear between 1990 and 1999 and movies.mrank >= 9
    group by did) movies_per_dir_90s
where directors.did = movies_per_dir_90s.did and dirMoviesCount >=2 order by lastname, firstName
```

Before Index

```
Table 'directors'. Scan count 0, logical reads 18, physical reads 10, read-ahead reads 0, lob log
Table 'movie_directors'. Scan count 1, logical reads 674, physical reads 2, read-ahead reads 678,
Table 'movies'. Scan count 1, logical reads 1918, physical reads 2, read-ahead reads 1917, lob lo

SQL Server Execution Times:
  CPU time = 140 ms,  elapsed time = 244 ms.
```

Index

```
create nonclustered index Index_movies_pyear_mrank
on movies (pyear, mrank) /*new*/

create nonclustered index Index_movie_directors_did
on movie_directors (did) /*new*/

create nonclustered index Index_directors_lastname_firstname
on directors (lastname, firstname) /*new*/
```

- ➔ Χρήση του Index_movies_pyear_mrank καθώς το mrank χρησιμοποιείται στο filtering και το pyear στο between(άρα filtering και αυτό).
- ➔ Χρήση του Index_movie_directors_did καθώς το did χρησιμοποιείται στο group by(και στο join και ενώ υπάρχει στο primary key έρχεται ως δεύτερο γνώρισμα).

➔ Χρήση του Index_directors_lastname_firstname καθώς το lastname και το firstname χρησιμοποιούνται στο order by.

After Index

```
Table 'directors'. Scan count 0, logical reads 18, physical reads 10, read-ahead reads 0, lob logi
Table 'movie_directors'. Scan count 1, logical reads 674, physical reads 2, read-ahead reads 678,
Table 'movies'. Scan count 1, logical reads 149, physical reads 3, read-ahead reads 145, lob logic
```

ΕΞΤΡΑ

AFTER INDEX (clear buffer και έπειτα δημιουργία index)

```
Table 'directors'. Scan count 0, logical reads 18, physical reads 0, read-ahead reads 0, lob log
Table 'movie_directors'. Scan count 1, logical reads 674, physical reads 0, read-ahead reads 0,
Table 'movies'. Scan count 1, logical reads 149, physical reads 0, read-ahead reads 0, lob logic

 SQL Server Execution Times:
   CPU time = 63 ms,  elapsed time = 94 ms.
```
