1ο ΠΡΟΤΖΕΚΤ ΣΧΕΔΙΑΣΜΟΣ ΒΑΣΕΩΝ ΔΕΔΟΜΕΝΩΝ

ΚΑΡΑΓΕΩΡΓΗΣ ΠΛΑΤΩΝ p3180068

ΖΗΤΗΜΑ 1

1ο Ερώτημα

Αρχικά, παραθέτω τον κώδικα πριν το index.

```
set statistics io on
set statistics time on
checkpoint
dbcc dropcleanbuffers
select title
from movies
where pyear between 1990 and 2000


checkpoint
dbcc dropcleanbuffers
select pyear,title
from movies
where pyear between 1990 and 2000


checkpoint
dbcc dropcleanbuffers
select title,pyear
from movies
where pyear between 1900 and 2000 order by pyear,title
```

Οι σελίδες που χρησιμοποιήθηκαν για τα 3 queries:

	1ο query	2ο query	3ο query
Logical reads	1918	1918	1918
Physical reads	3	3	3
Read-ahead reads	1914	1914	1914

Όπως βλέπουμε είναι όλες ίδιες και είναι λογικό καθώς τα queries κάνουν το ίδιο πράγμα απλά παρουσιάζουν διαφορετικά τα δεδομένα.

Το Estimated I/O cost είναι αντίστοιχα:

	1ο query	2ο query	3ο query
Estimated I/O cost	1.4172	1.4172	1.42173(+sort cost)

Το κόστος για τα δύο πρώτα queries είναι παρόμοιο. Στο 3ο query προσθέτουμε το κόστος του Sort και γι'αυτό είναι ελάχιστα μεγαλύτερο.

ΔΗΜΙΟΥΡΓΙΑ INDEX

Τα queries είναι σχεδόν πανομοιότυπα. Η διαφορά, είναι στο 3ο query που κάνουμε order by pyear, title και γι'αυτό θα ήταν καλό ένα index στα (pyear,title). Επίσης, υπάρχει clustered primary key στον πίνακα movies άρα δεν γίνεται το index να είναι clustered. Επομένως:

Χρησιμοποιώ το index: CREATE NONCLUSTERED INDEX first_index ON movies(pyear,title)

Οι σελίδες που χρησιμοποιήθηκαν για τα 3 queries μετά το index:

	1ο query	2ο query	3ο query
Logical reads	351	351	1384
Physical reads	3	3	3
Read-ahead reads	348	348	1381

Είναι ξεκάθαρη η βελτίωση στις σελίδες που χρησιμοποιούνται. Τα queries μεταξύ τους έχουν και πάλι ίδιο κόστος σε σελίδες.

Το Estimated I/O cost είναι αντίστοιχα:

	1ο query	2ο query	3ο query
Estimated I/O cost	0.254977	0.254977	0.254977

Φαίνεται και εδώ ότι το index ήταν αποδοτικό. Το sort cost που υπήρχε πριν από το index δεν υπάρχει πλέον επομένως τα queries έχουν πανομοιότυπο κόστος μεταξύ τους.

2Ο Ερώτημα

Αρχικά, παραθέτω τον κώδικα πριν το index.

```
set statistics io on
set statistics time on
checkpoint
dbcc dropcleanbuffers
select mid,count(rating)
from user_movies group by mid order by mid

checkpoint
dbcc dropcleanbuffers
select userid,count(rating)
from user_movies group by userid order by userid
```

Οι σελίδες που χρησιμοποιήθηκαν για τα 2 queries:

	1ο query	2ο query
Logical reads	2601	2601
Physical reads	3	3
Read-ahead reads	2597	2597

Τα κόστη είναι ίδια και για τα δύο queries όσον αφορά τις σελίδες. Κάτι αναμενόμενο αφού η λειτουργία τους είναι παρόμοια.

Το Estimated I/O cost είναι αντίστοιχα:

	1ο query	2ο query
Estimated I/O cost	1.91942	1.9306813(+sort cost)

Εδώ το 2ο query έχει ελάχιστα μεγαλύτερο κόστος I/O λόγω του sort cost.

ΔΗΜΙΟΥΡΓΙΑ INDEX

Χρησιμοποιώ το index: CREATE INDEX sec_index ON user_movies(mid,userid) INCLUDE (rating)

Οι σελίδες που χρησιμοποιήθηκαν για τα 2 queries:

	1ο query	2ο query
Logical reads	2226	2226
Physical reads	3	3
Read-ahead reads	2222	2222

Εδώ το κόστος στις σελίδες μειώθηκε μετά το index αλλά όχι τόσο όσο στο προηγούμενο query. Γενικά αυτά τα δύο queries είχαν διαφορετικούς στόχους από άποψη columns επομένως είναι αναμενόμενο ότι δεν βρίσκουμε ένα index που να δουλεύει πάρα πολύ αποτελεσματικά και για τα δύο. Ωστόσο αυτό που αναφέρθηκε παραπάνω, έστω και με μικρή μείωση, κάνει τη δουλειά.

Το Estimated I/O cost είναι αντίστοιχα:

	1ο query	2ο query
Estimated I/O cost	1.64609	1.6573513(+sort cost)

Αντίστοιχα με τις σελίδες η μείωση στο κόστος I/O δεν είναι πολύ μεγάλη, είναι της τάξης του 15%.

ΖΗΤΗΜΑ 2

1ο Ερώτημα

```
set statistics io on
set statistics time on
checkpoint
dbcc dropcleanbuffers
SELECT DISTINCT title
 FROM movies
 INNER JOIN movies_genre ON movies.mid=movies_genre.mid
WHERE  genre = 'Action' OR genre='Adventure'
```

Το σκρινσοτ παραπάνω επιστρέφει τα ίδια rows που επιστρέφει και το query της εκφώνησης και έχει καλύτερη απόδοση. Είναι μία ασυνήθιστη περίπτωση καθώς στον γενικό κανόνα το UNION είναι ταχύτερο του JOIN ωστόσο βλέπουμε ότι στη συγκεκριμένη περίπτωση αυτό δεν ισχύει.

Πρώτα, καταγράφω το κόστος σε σελίδες πριν τη χρήση οποιουδήποτε index.

	Για τον πίνακα Movies	Για τον Movies_genre
Logical Reads	1918	1123
Physical reads	0	0
Read-ahead reads	1837	1123

Το Estimated I/O cost για το query συνολικά είναι 2.2739586.

ΔΗΜΙΟΥΡΓΙΑ INDEX

Από τη στιγμή που σε αυτό το ερώτημα δεν υπάρχει περιορισμός να χρησιμοποιήσω μόνο ένα index, θα χρησιμοποιήσω δύο ώστε να πετύχω καλύτερη απόδοση. Στον πίνακα movies_genre από τη στιγμή που έχω τη δυνατότητα, θα χρησιμοποιήσω clustered index που είναι ταχύτερο του non-clustered, στη στήλη movies_genre. Στον πίνακα movies θα χρησιμοποιήσω ένα non-clustered index στη στήλη mid που χρησιμοποιείται στο inner join και θα κάνω include τη στήλη title που χρησιμοποιείται στο select.

```
CREATE CLUSTERED INDEX index1 ON movies_genre(genre)
CREATE NONCLUSTERED INDEX index2 ON movies(mid) INCLUDE (title)
```

Το κόστος σε σελίδες μετά τη χρήση των indices:

	Για τον πίνακα Movies	Για τον Movies_genre
Logical Reads	1397	94
Physical reads	3	2
Read-ahead reads	1393	87

Στον πίνακα movies_genre η διαφορά μετά τα indices είναι τεράστια, κυρίως επειδή το index που χρησιμοποιήθηκε για αυτόν τον πίνακα επιτάχυνε την διαδικασία στη συνθήκη WHERE που συνήθως είναι χρονοβόρα. Το γεγονός ότι είναι clustered πιθανότατα θα συνέβαλε στη μείωση. Παράλληλα, στον πίνακα movies η διαφορά δεν είναι τόσο μεγάλη όσο στον άλλο πίνακα ωστόσο είναι φανερό ότι και εκεί τα indices που χρησιμοποιήθηκαν έκαναν διαφορά.

Το Estimated I/O cost για το query συνολικά είναι 1.1213698. Και εδώ παρατηρείται μεγάλη διαφορά καθώς το κόστος μειώθηκε κατά 50%.

$2^{Ο}$ Ερώτημα

Το πρώτο query που επιστρέφει τα rows που ζητάει η εκφώνηση είναι:

```
set statistics io on
set statistics time on
checkpoint
dbcc dropcleanbuffers
SELECT DISTINCT movies.title
FROM movies
INNER JOIN roles on movies.mid = roles.mid
INNER JOIN actors on roles.aid = actors.aid
EXCEPT
SELECT DISTINCT movies.title
FROM movies
INNER JOIN roles on movies.mid = roles.mid
INNER JOIN actors on roles.aid = actors.aid
WHERE(EXISTS(SELECT movies.mid
                        FROM actors
                        WHERE movies.mid = roles.mid AND roles.aid = actors.aid AND actors.gender ='F'));
```

Η λογική πίσω από αυτό το query είναι η εξής: Κάνουμε JOIN τους τρεις πίνακες που χρειάζονται για την εύρεση του αποτελέσματος που ζητάει η εκφώνηση. Έπειτα, κρατάμε όλα τα rows που επιστρέφει χωρίς περιορισμό. Ύστερα, κάνουμε το ίδιο άλλα ζητάμε να πάρουμε τα rows που περιέχουν ταινίες που έχουν τουλάχιστον μία γυναίκα. Τέλος, με τη χρήση του EXCEPT αφαιρούμε αυτά τα δύο σύνολα και το αποτέλεσμα είναι όλες οι ταινίες που δεν έχουν καμία γυναίκα ηθοποιό, δηλαδή αυτές που έχουν μόνο άντρες. Στην περίπτωση που υπήρχε κάποια ταινία στην οποία δεν υπήρχε κανένας ηθοποιός καταχωρημένος, δηλαδή το πεδίο ήταν NULL, τότε θα είχαμε καταμετρήσει μία ταινία που δεν θα είχε κανέναν ηθοποιό και επομένως θα είχαμε κάνει λάθος. Όμως, όλες οι τιμές που χρησιμοποιούμε στο INNER JOIN είναι PRIMARY KEYS στους αντίστοιχους πίνακες και επομένως NOT NULL. Άρα είμαστε σίγουροι ότι δεν υπάρχει περίπτωση να έχουμε συμπεριλάβει κάποια NULL τιμή και επομένως το query δουλεύει σωστά.

Το κόστος σε σελίδες για το παραπάνω query:

	Πίνακας Roles	Πίνακας Movies	Πίνακας Actors
Logical Reads	8822	3836	10083
Physical reads	3	3	3
Read-ahead reads	4407	1914	3357

Το Estimated I/O cost για το query συνολικά είναι 16.8.

ΔΗΜΙΟΥΡΓΙΑ INDEX

Το query ουσιαστικά αποτελείται από δύο εσωτερικά subqueries, τα οποία είναι πανομοιότυπα μέχρι και πριν το WHERE. Έχουμε:

```
CREATE NONCLUSTERED INDEX index1 ON movies(mid) INCLUDE (title)
CREATE NONCLUSTERED INDEX index2 ON actors(gender)
CREATE NONCLUSTERED INDEX index3 ON roles(mid)
```

Φτιάχνουμε ένα index στον πίνακα movies στη στήλη mid και κάνουμε include το title που χρησιμοποιείται στο SELECT. Έπειτα, προσθέτουμε άλλα δύο indices τα οποία είναι το ένα στον πίνακα actors στη στήλη gender που χρησιμοποιείται στο WHERE clause και το άλλο στον πίνακα roles στη στήλη mid που χρησιμοποιείται για το INNER JOIN.

Το κόστος σε σελίδες μετά τη χρήση των indices:

	Πίνακας Roles	Πίνακας Movies	Πίνακας Actors
Logical Reads	3796	2794	2648
Physical reads	3	3	3
Read-ahead reads	1894	1393	1112

Παρατηρούμε ότι στον πίνακα actors και στον πίνακα roles υπήρξε μεγάλη διαφορά (ειδικά στον πίνακα actors). Στον πίνακα movies υπήρχε επίσης σημαντική μείωση άλλα όχι αντίστοιχη με τους άλλους πίνακες. Η χρήση των indices πάντως κρίνεται επιτυχής.

Το Estimated I/O cost για το query συνολικά είναι 6.834. Η διαφορά είναι μεγάλη με το προηγούμενο κόστος καθώς έχει πέσει κάτω από το μισό. Και από εδώ συμπεραίνουμε ότι η συμβολή των indices ήταν καθοριστική.

Το δεύτερο query που επιστρέφει τα rows που ζητάει η εκφώνηση είναι:

```
set statistics io on
set statistics time on
checkpoint
dbcc dropcleanbuffers
SELECT movies.title
FROM movies
INNER JOIN roles on movies.mid = roles.mid
INNER JOIN actors on roles.aid = actors.aid
GROUP BY movies.title
HAVING SUM(CASE WHEN actors.gender = 'F' THEN 1 ELSE 0 END)=0;
```

Η λογική εδώ είναι παρόμοια με πριν απλά είναι λίγο διαφορετική η εκτέλεση. Όπως πριν ενώνουμε με INNER JOIN τους τρεις πίνακες που χρειάζονται για την εύρεση του αποτελέσματος που ζητάει η εκφώνηση. Η διαφορά με πριν έγκειται μετά από το JOIN καθώς κάνουμε GROUP BY με βάση τον τίτλο της ταινίας και ύστερα δημιουργούμε ένα άθροισμα SUM. Κάθε φορά που μία ταινία έχει μία γυναίκα ηθοποιό αυξάνουμε το άθροισμα κατά ένα. Στο τέλος, κρατάμε όλες τις ταινίες που είχαν άθροισμα μηδέν. Αντίστοιχα με πρίν, είμαστε σίγουροι ότι δεν υπάρχει περίπτωση να έχουμε συμπεριλάβει κάποια NULL τιμή γιατί όλες οι

τιμές είναι PRIMARY KEYS στους πίνακες τους και επομένως το query δεν θα επιστρέψει rows που δεν χρειάζονται.

Το κόστος σε σελίδες για το παραπάνω query:

	Πίνακας Roles	Πίνακας Movies	Πίνακας Actors
Logical Reads	4411	1918	3361
Physical reads	3	3	3
Read-ahead reads	4407	1914	3357

Το Estimated I/O cost για το query συνολικά είναι 7.15234.

ΔΗΜΙΟΥΡΓΙΑ INDEX

Τα indices που θα χρειαστούν θα είναι τα ίδια με προηγουμένως, καθώς μπορεί να αλλάζει ο υπολογισμός στο WHERE αλλά πρακτικά αφορά πάλι τον πίνακα actors και το column genre. Παράλληλα, πριν από το WHERE clause τα queries είναι παρόμοια.

```
CREATE NONCLUSTERED INDEX index1 ON movies(mid) INCLUDE (title)
CREATE NONCLUSTERED INDEX index2 ON actors(gender)
CREATE NONCLUSTERED INDEX index3 ON roles(mid)
```

Το κόστος σε σελίδες μετά τη χρήση των indices:

	Πίνακας Roles	Πίνακας Movies	Πίνακας Actors
Logical Reads	1898	1397	1116
Physical reads	3	3	3
Read-ahead reads	1894	1393	1112

Η διαφορά είναι ξεκάθαρη στους πίνακες Roles, Actors. Όπως και στο προηγούμενο query έτσι και τώρα, ο πίνακας Movies επηρεάστηκε λιγότερο από τους άλλους.

Το Estimated I/O cost για το query συνολικά είναι 3.262717. Όπως και πριν, μειώθηκε σημαντικά καθώς είναι το μισό από αυτό που ήταν πριν τη χρήση οποιουδήποτε index.

ΖΗΤΗΜΑ 3

- Επέστρεψε τους τίτλους των ταινιών που έχουν κατηγορία 'Mystery' και βαθμολογία >=4.
- Επέστρεψε τα ονοματεπώνυμα όλων των σκηνοθετών που σκηνοθέτησαν ακριβώς ή περισσότερες από 10 ταινίες στη λίστα.

Για το πρώτο query:

```
set statistics io on
set statistics time on
checkpoint
dbcc dropcleanbuffers
SELECT DISTINCT movies.title
FROM movies
INNER JOIN movies_genre ON movies_genre.mid = movies.mid
INNER JOIN user_movies ON user_movies.mid = movies.mid
WHERE movies_genre.genre = 'Mystery' AND user_movies.rating>=4;
```

Το κόστος σε σελίδες για το παραπάνω query:

	Πίνακας user_movies	Πίνακας movies_genre	Πίνακας Movies
Logical Reads	2599	1409	1841
Physical reads	3	0	3
Read-ahead reads	2597	1409	1914

Το Estimated I/O cost για το query συνολικά είναι 4.394.

ΔΗΜΙΟΥΡΓΙΑ INDEX

Αρχικά δημιουργούμε ένα clustered index στον πίνακα movies_genre στο column που έχουμε τη δυνατότητα. Έπειτα δημιουργούμε ένα index στον πίνακα movies στο column mid που γίνεται το INNER JOIN και κάνουμε include το column title που το παίρνουμε στο SELECT. Δεν προσθέτω index για τον πίνακα user_movies γιατί όταν το δοκίμασα δεν έκανε διαφορά.

```
CREATE CLUSTERED INDEX index1 ON movies_genre(genre)
CREATE NONCLUSTERED INDEX index3 ON movies(mid) INCLUDE (title)
```

Το κόστος σε σελίδες μετά τη χρήση των indices:

	Πίνακας user_movies	Πίνακας movies_genre	Πίνακας Movies
Logical Reads	2599	22	1342
Physical reads	3	2	3
Read-ahead reads	2597	19	1393

Στον πίνακα movies_genre είναι προφανές ότι τα indices βοήθησαν καθώς μειώθηκε δραματικά το κόστος. Στον πίνακα movies υπάρχουν διαφορές που δείχνουν ότι βοήθησαν τα indices αλλά σε καμία περίπτωση δεν συγκρίνεται με τη μείωση στον πίνακα movies_genre. Στον πίνακα user_movies δεν βλέπουμε καμία μείωση διότι δεν χρησιμοποιήθηκε κανένα index σε αυτόν.

Το Estimated I/O cost για το query συνολικά είναι 2.9909. Υπάρχει μείωση κυρίως από την πλευρά των υπολογισμών στον πίνακα movies_genre. Σε γενικές γραμμές η συμβολή των indices κρίνεται επιτυχής.

Για το δεύτερο query:

```
set statistics io on
set statistics time on
checkpoint
dbcc dropcleanbuffers
SELECT DISTINCT
    d.firstName, d.lastname
FROM
    directors d
WHERE
    d.did IN (SELECT
            m.did
            FROM
            movie_directors m
            GROUP BY m.did
            HAVING COUNT(m.did) >10)
ORDER BY d.lastname;
```

Το κόστος σε σελίδες για το παραπάνω query:

	Πίνακας Directors	Πίνακας Movie_Directors
Logical Reads	13730	675
Physical reads	16	3
Read-ahead reads	334	671

Το Estimated I/O cost για το query συνολικά είναι: 0.5138073.

ΔΗΜΙΟΥΡΓΙΑ INDEX

Όλη η λειτουργία του query είναι στο εσωτερικό SELECT και συγκεκριμένα στον πίνακα movie_directors και στο column did. Από εκεί και πέρα δεν έχει ιδιαίτερο νόημα να προσθέσουμε άλλο index καθώς οι άλλες δύο στήλες που χρησιμοποιούνται (firstName,lastName) χρειάζονται για το SELECT και για το ORDER BY μόνο, δηλαδή για «οπτικές» διαφορές. Άρα χρησιμοποιώ το index:

```
CREATE NONCLUSTERED INDEX index1 ON movie_directors(did)
```

Το κόστος σε σελίδες μετά τη χρήση του index:

	Πίνακας Directors	Πίνακας Movie_Directors
Logical Reads	18554	557
Physical reads	1	3
Read-ahead reads	349	553

Παρατηρούμε ότι στον πίνακα movie_directors υπήρχε μία μικρή μείωση στις σελίδες που χρησιμοποιήθηκαν. Ωστόσο, στον πίνακα directors τα πράγματα είναι διαφορετικά. Ενώ τα physical reads μειώθηκαν πάρα πολύ, τα logical και read-ahead reads αυξήθηκαν. Παρόλα αυτά, εμάς μας αφορούν κυρίως τα physical reads γιατί το διάβασμα από τον δίσκο είναι πολύ πιο αργό από το διάβασμα από την cache επομένως θεωρούμε ότι το index βοήθησε.

Το Estimated I/O cost για το query συνολικά είναι: 0.4264003. Παρουσιάζει μία μικρή μείωση, ωστόσο σημαντική και γι'αυτό θεωρούμε ότι το index έχει σημαντική συμβολή.