

# 1ο μέρος Project, Σχεδιασμός βάσεων Δεδομένων

p3170122 | Κων/νος Νικολούτσος

## Πριν τρέξουμε query

\* Για να είμαστε πιο αντικειμενικοί στις απαντήσεις μας χρησιμοποιούμε σε όλα τα query τις παρακατω εντολές.

- Καθαρίζουμε την buffer cache

```
CHECKPOINT;  
DBCC DROPCLEANBUFFERS;  
DBCC FREESYSTEMCACHE('ALL');
```
- Ενεργοποιούμε τα STATISTICS

```
SET STATISTICS IO ON;
```

## Άσκηση 1

1.

SQL ερώτημα:

```
SELECT title  
FROM bibrecs  
WHERE title LIKE 'Οικ%'  
ORDER BY title
```

Δημιουργία δείκτη:

```
CREATE NONCLUSTERED INDEX index_title ON bibrecs(title)
```

2.

α.

```
SELECT title  
FROM bibrecs  
WHERE title LIKE '% πληροφορική %'  
OR title LIKE 'πληροφορική %'  
OR title LIKE '% πληροφορική'  
OR title LIKE 'πληροφορική'
```

\*(Θεωρώ ότι ψάχνω αυστηρά την λέξη 'πληροφορική'. Αν δεν βάζαμε τόσες περιπτώσεις τότε θα πιανόταν και η λέξη 'τηλεπληροφορική' ως 'πληροφορική')

β.

```
SELECT title, material
FROM bibrecs
WHERE title = 'Economics'
```

γ.

```
SELECT title
FROM bibrecs
WHERE title LIKE 'Economics %'
OR title LIKE 'Economics'
```

Ας εξετάσουμε αν το index του ερωτήματος (1) βελτιώνει την ταχύτητα των παραπάνω queries:

α'.

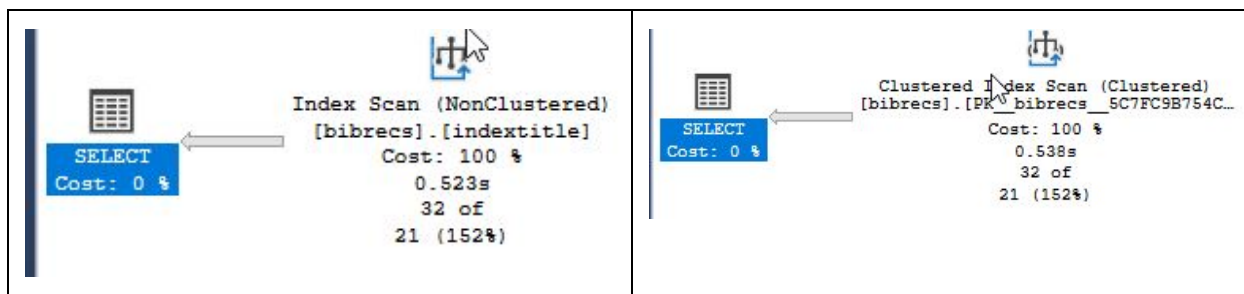
Χρησιμοποιώντας τα εργαλεία που παρέχει Ο MS SERVER MANAGEMENT STUDIO παρατηρούμε ότι το index βοηθάει στην ελαχιστοποίηση του χρόνου I/O εφόσον μειώνονται οι σελίδες που γίνονται load. Παρόλα αυτά, η βελτίωσή δεν είναι πολύ μεγάλη και αυτό οφείλεται στον predicate του where clause. Πιο αναλυτικά επειδή ψάχνουμε οπουδήποτε στον τίτλο την λέξη πληροφορική το DBMS αναγκάζεται να ψαξει περισσότερο μέσα στο b-tree index. Ίσως αν το predicate ήταν μόνο 'Πληροφορική%' το query θα εκανε load πολύ λιγότερες σελίδες.

**Σελίδες που κάνει load:**

	With index	Without index
Logical reads	485	864
Physical reads	1	2
Read-ahead reads	500	867

**Execution plans:**

With index	Without index
------------	---------------



β'.

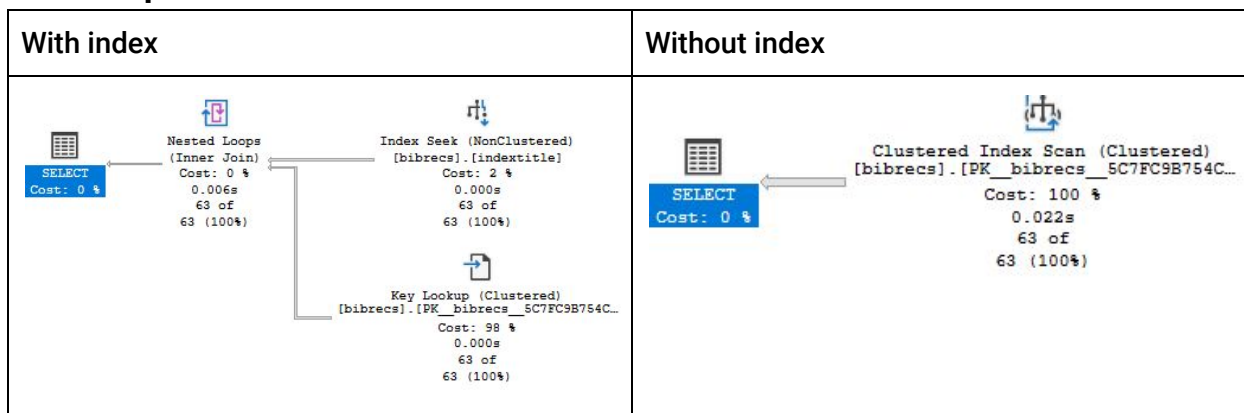
Σε αυτήν την περίπτωση το index βοηθάει λίγο όπως φαίνεται από τις λιγότερες σελίδες που κάνει load. Ωστόσο αν παρατηρήσουμε στο execution plan θα καταλάβουμε ότι το Index του ερωτηματος 1 δεν κουβαλάει μαζί του την στήλη material με αποτέλεσμα το DBMS να ψάχνει να το βρει με key lookup για να βρει το material. Φυσικά αν γράφαμε το παρακάτω index θα είχαμε τρομερές βελτιστοποιήσεις!

`CREATE INDEX index_title ON bibrecs(title) INCLUDE(material)`

**Σελίδες που κάνει load:**

	With index	Without index
Logical reads	356	864
Physical reads	3	2
Read-ahead reads	288	867

**Execution plans:**



Υ'.

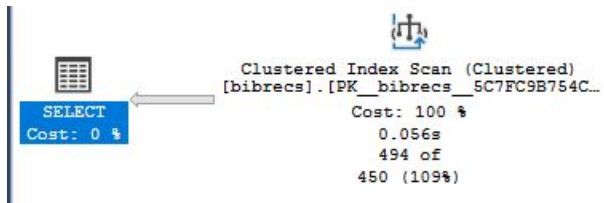
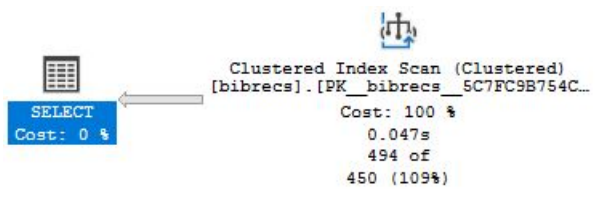
Το dbms εκανε τα στατιστικά του και αποφασισε οτι δεν αξιζει να χρησιμοποιήσει το non-clustered index. Αυτο πιθανων το εκανε διότι το συγκεκριμενο query δινει ενα μεγαλο αριθμο απο rows. Απο την αλλη, ενα index το οποιο θα ηταν πολυ καλο στην περίπτωση αυτη ειναι το παρακάτω το οποιο κατεβάζει τα load pages στα  $9+2+5 = 16!$

```
CREATE INDEX index_title ON bibrecs(title) INCLUDE(material)
```

#### Σελίδες που κανει load:

	With index (ignored)	Without index
Logical reads	864	864
Physical reads	2	2
Read-ahead reads	867	867

#### Execution plans:

With index	Without index
 <p>SELECT Cost: 0 %</p> <p>Clustered Index Scan (Clustered) [bibrecs].[PK_bibrecs_5C7FC9B754C...] Cost: 100 % 0.056s 494 of 450 (109%)</p>	 <p>SELECT Cost: 0 %</p> <p>Clustered Index Scan (Clustered) [bibrecs].[PK_bibrecs_5C7FC9B754C...] Cost: 100 % 0.047s 494 of 450 (109%)</p>

**Συμπέρασμα:** Ισως θα επρεπε να τροποποιήσουμε στην βαση μας το index αφου βλεπουμε οτι δεν καλυπτει ενα μεγαλο ευρος καποιων query που ζητούνται. Αρα να το κανουμε:

```
CREATE NONCLUSTERED INDEX index_title  
ON bibrecs(title)  
INCLUDE(material)
```

## Άσκηση 2

1.

Ακολουθούν queries που απαντούν στα ερωτήματα της εκφώνησης:

α.

```
SELECT bibrecs.title, bibrecs.lang
FROM bibrecs, publishers
WHERE bibrecs.pubid = publishers.pubid
AND publishers.pubname = 'Κλειδάριθμος'
```

β.

```
SELECT departments.depname, count(*)
FROM loanstats, borrowers, departments
WHERE loanstats.loandate >= '2000-01-01'
AND loanstats.loandate <= '2000-12-31'
AND loanstats.bid = borrowers.bid
AND departments.depcode = borrowers.depcode
GROUP BY departments.depname
```

γ.

```
SELECT bibrecs.title, bibrecs.lang, authors.author
FROM bibrecs, bibauthors, authors, bibterms, sterms
WHERE bibrecs.bibno = bibauthors.bibno
AND bibauthors.aid = authors.aid
AND bibterms.bibno = bibrecs.bibno
AND bibterms.tid = sterms.tid
AND sterms.term = 'Databases'
```

## 2.

\*Θεωρώ ότι η εκφώνηση αναφέρετε στην δημιουργία index που να βελτιστοποιεί μεμονωμένα κάθε query.

Τα index που θα χρησιμοποιούσα για την βελτιστοποίηση των παραπάνω queries είναι:

α.

Όπως φαίνεται είναι ξεκάθαρο ότι φορτώνονται λιγότερες σελίδες με τα παρακάτω indexes. Επίσης μπορούμε να δούμε και από το execution plan ότι το DBMS οντως χρησιμοποίησε τα index μας το οποίο δηλώνει ότι θεωρήθηκαν χρήσιμα κατά την διαδικασία εύρεσης καλύτερου πλάνου!

```
CREATE INDEX index_pubid_title_lang  
ON bibrecs(pubid)  
INCLUDE(title, lang)
```

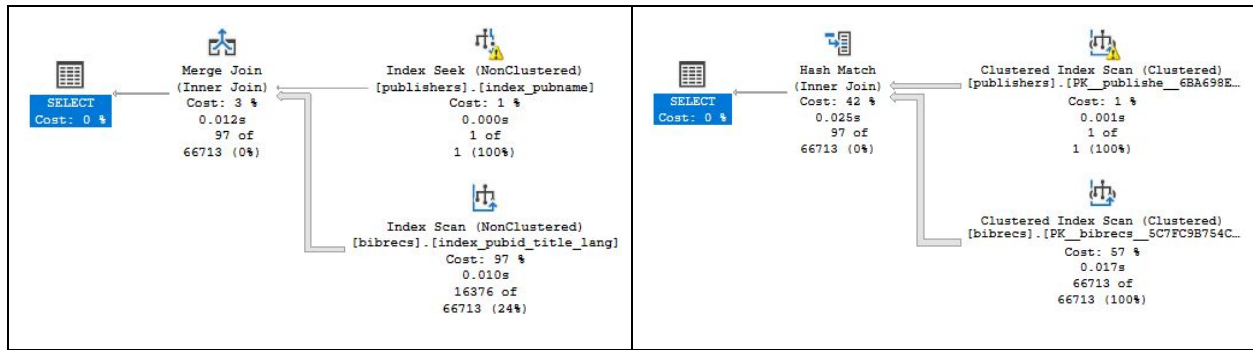
```
CREATE INDEX index_pubname  
ON publishers(pubname)  
INCLUDE(pubid)
```

**Σελίδες που κάνει load(Για κάθε πίνακα):**

	With index	Without index
Logical reads	$136+2 = 138$	$864 + 17 = 881$
Physical reads	$3+2 = 5$	$2 + 1 = 3$
Read-ahead reads	$542+0 = 542$	$867 + 22 = 889$

**Execution plans:**

With index	Without index
------------	---------------



## B.

Τα παρακάτω index βελτιώνουν την εκτέλεση του query. Με κόκκινο βελάκι στο δευτερο execution plan βλέπουμε σε ποιους operators στοχεύουμε να μειώσουμε το κόστος. Πιο αναλυτικά το κόστος απο 41% το πηγαμε 5%.

```
CREATE INDEX index_loanstat_bid
ON loanstats (loandate)
INCLUDE (bid)
```

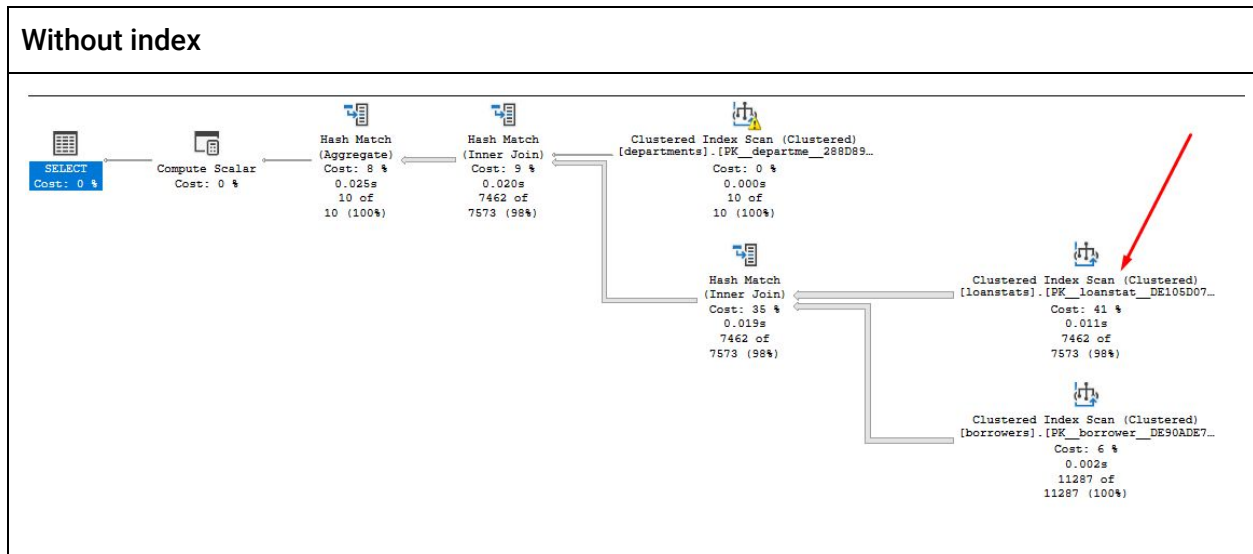
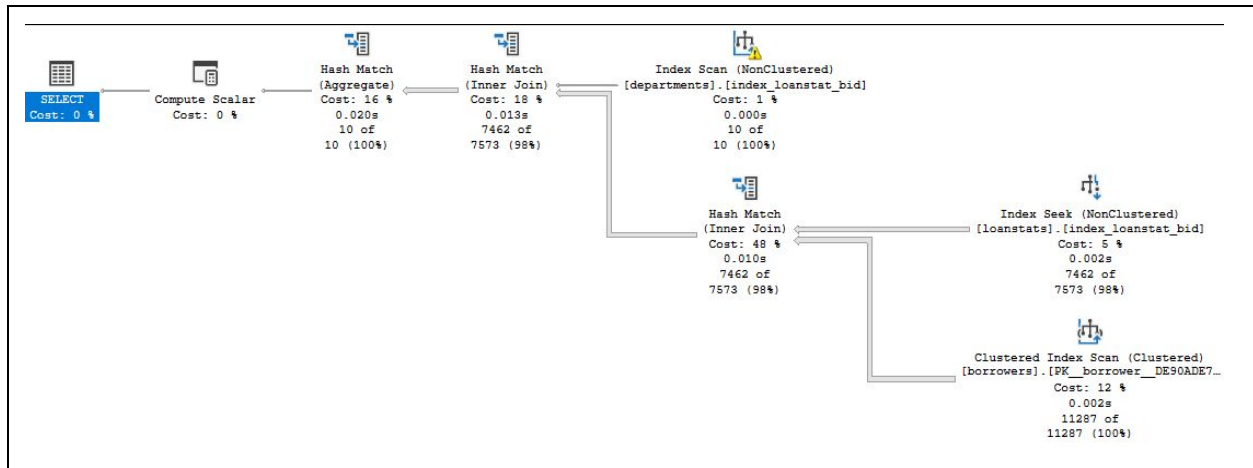
```
CREATE INDEX index_loanstat_bid
ON departments(depname) /** Μπορούμε να το παραλείψουμε αφού είναι
μονο 10 οι εγγραφες (ισως στο μελλον να χρησιμοποιηθεί) */
```

## Σελίδες που κάνει load(Για καθε πινακα):

	With index	Without index
Logical reads	51 + 19 + 2 = 72	51 + 320 + 2 = 373
Physical reads	1 + 1 + 1 = 3	1 + 1 + 1 = 3
Read-ahead reads	56 + 17 + 0 = 73	56 + 325 + 0 = 391

## Execution plans:

With index



Υ.

Όπως παρατηρούμε απο την μείωση σελίδων φόρτωσης το Index βελτιώνει το query. Το συγκεκρικρέμο query ετσι οπως ειναι γραμμενο ισως να μην ειναι το καλυτερο λογικό πλάνο που θα μπορούσαμε να έχουμε. Ισως χρησιμοποιώντας push down selection να εκανε load λιγότερες σελίδες. Ωστόσο κατα την γνώμη μου το query ετσι ειναι πιο readable και αρα πιο maintainable απο τους προγραμματιστες σε αντιθεση με εμφωλευμένα query που ισως να έχουν ελάχιστη καλύτερη απόδοση.

```
CREATE INDEX index_term_tid
ON sterms(term)
INCLUDE(tid)
```



```
CREATE INDEX index_tid_bibno ON bibterms(tid)
```

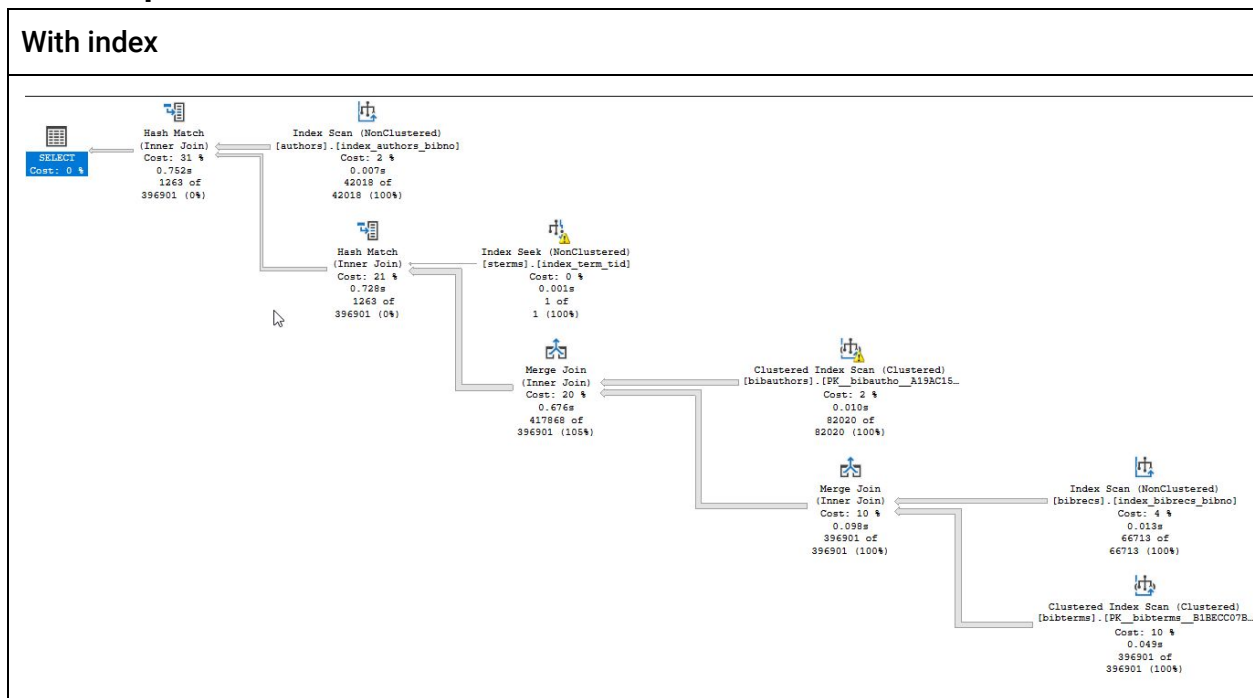
```
CREATE INDEX index_bibreces_bibno
ON bibreces(bibno)
INCLUDE(title, lang)
```

```
CREATE INDEX index_authors_bibno
ON authors(aid)
INCLUDE(author)
```

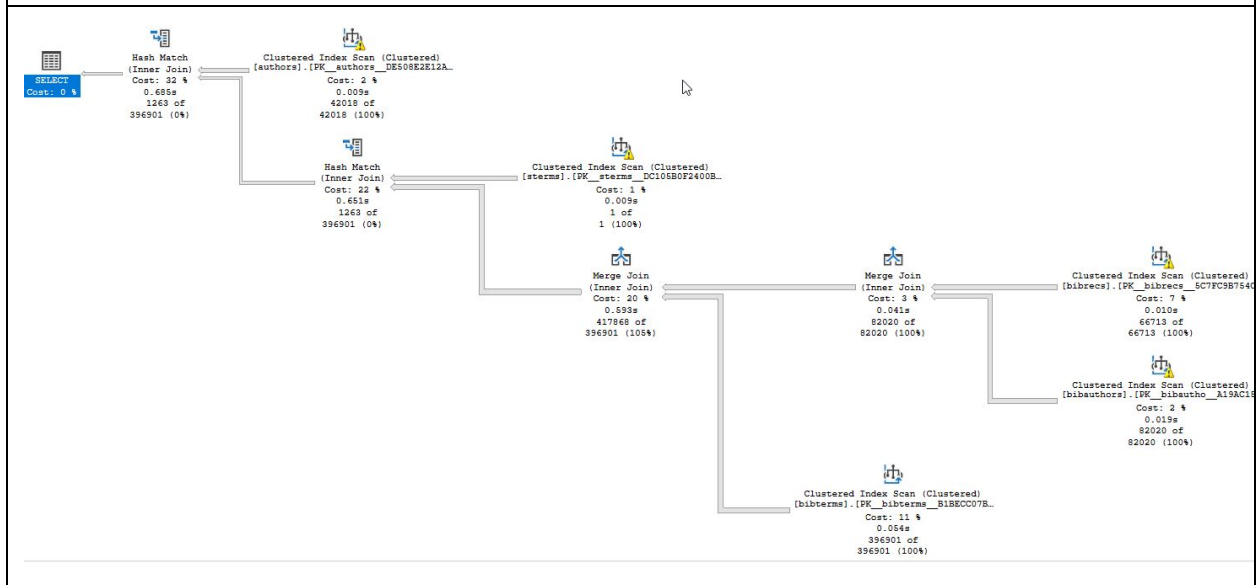
**Σελίδες που κάνει load(Για κάθε πίνακα):**

	With index	Without index
Logical reads	$839 + 502 + 172 + 2 + 160 = 1675$	$839 + 175 + 864 + 79 + 176 = 2133$
Physical reads	$2 + 1 + 1 + 2 + 1 = 7$	$2 + 1 + 2 + 1 + 1 = 7$
Read-ahead reads	$841 + 500 + 180 + 0 + 158 = 1679$	$841 + 180 + 867 + 84 + 181 = 2153$

**Execution plans:**



## Without index



## Άσκηση 3

1.

Ακολουθούν 3 διαφορετικά query που απαντούν στο ερώτημα

No1.

```
SELECT bibrecs.bibno, bibrecs.title
FROM copies, bibrecs
WHERE copyloc = 'OPA'
AND bibrecs.bibno = copies.bibno
INTERSECT
SELECT bibrecs.bibno, bibrecs.title
FROM copies, bibrecs
WHERE copyloc = 'ANA'
AND bibrecs.bibno = copies.bibno
/** Without push down selection*/
```

No2.

```
SELECT bibrecs.bibno, bibrecs.title
FROM (    SELECT bibno
          FROM copies
          WHERE copyloc = 'OPA'
        INTERSECT
          SELECT bibno
          FROM copies
          WHERE copyloc = 'ANA'
        ) AS temp,
      bibrecs
WHERE bibrecs.bibno = temp.bibno
/** With push down selection*/
```

No3.

```
SELECT DISTINCT bibrecs.bibno, bibrecs.title
FROM copies as A, copies AS B, bibrecs
WHERE A.bibno = B.bibno
AND (A.copyloc = 'OPA' OR A.copyloc = 'ANA')
AND ( B.copyloc = 'OPA' OR B.copyloc = 'ANA')
AND A.copyloc != B.copyloc
AND bibrecs.bibno = A.bibno
/** Self-join method*/
```

Το ερώτημα που θα επιλέξουμε είναι αυτό με το καλύτερο λογικό πλάνο.

Ετσι λοιπόν θα επέλεγα το 2ο query. Η επιλογή αυτή έγινε διότι το συγκεκριμένο query κάνει χρήση του push down selection που όπως είδαμε και από τα slides του μαθήματος είναι πολύ χρήσιμο αφού αποσυμφορίζει το κόστος από αργούς operators όπως join κλπ.

Στον παρακάτω πίνακα φαίνεται ξεκάθαρα ότι το 2ο query είναι 'λογικά' καλύτερο από τα άλλα δύο.

	No1. query	No2. query	No3 query
Logical reads	2240	1375	3696
Physical reads	3	3	261
Read-ahead reads	1293	1128	2243

2.

```
CREATE INDEX index_term_tid
ON copies(bibno)
INCLUDE(bibno)
```

Το παραπάνω index επιταχύνει τα εμφωλευμένα queries του 2ου query.

```
CREATE INDEX index_term_tid
ON copies(copyloc)
INCLUDE(title)
```

## Άσκηση 4

1.

```
CREATE TABLE dbo.words(
    wid int IDENTITY(1,1),
    word VARCHAR(100) NOT NULL UNIQUE
    PRIMARY KEY (wid)
)
```

```
CREATE TABLE dbo.bibwords(
    wid int NOT NULL,
    bibno int NOT NULL PRIMARY KEY (wid,bibno),
    FOREIGN KEY (bibno) REFERENCES bibrecs(bibno),
    FOREIGN KEY (wid) REFERENCES words(wid)
)
```

## Schema

words
PK wid
word

bibwords
PK, FK, wid
PK bibno

## 2.

Καταρχάς για να φτιάξουμε index ας δούμε τι query προσπαθούμε να βελτιστοποιήσουμε.

"Εμφάνισε τον κωδικό των βιβλιογραφικών εγγραφών που περιέχουν την λέξη 'οικονομία' **και** την λέξη 'Ευρώπη'"

```
SELECT bibno FROM bibwords WHERE wid IN(
SELECT wid FROM words WHERE word = 'οικονομία')
INTERSECT
SELECT bibno FROM bibwords WHERE wid IN(
SELECT wid FROM words WHERE word = 'Ευρώπη')
```

"Εμφάνισε τον κωδικό των βιβλιογραφικών εγγραφών που περιέχουν την λέξη 'οικονομία' **ή** την λέξη 'Ευρώπη'"

```
SELECT bibno FROM bibwords WHERE wid IN(
SELECT wid FROM words WHERE word = 'οικονομία')
UNION
SELECT bibno FROM bibwords WHERE wid IN(
SELECT wid FROM words WHERE word = 'Ευρώπη')
```

Οπότε βάση αυτών των query μπορούμε να φτιάξουμε το εξής index.  
\*(Το ms server φτιάχνει by default index στο unique columns.)

```
CREATE INDEX words_word_wid ON words(word) INCLUDE (wid)
```

Το παραπάνω index κάνει πιο γρήγορη το εσωτερικό query.

```
CREATE INDEX bibwords_wid_bibno ON bibwords(wid) INCLUDE(bibno)
```

Το παραπάνω query φορτώνει εκ των προτερων στην RAM του DBMS την bibno στήλη μειώνοντας έτσι το IO cost.

## Άσκηση 5

\*(Κανονικά θα έπρεπε να γραφόντουσαν INSERT triggers τα οποία θα γέμιζαν τους παρακάτω πίνακες αυτόματα)

\*(Η παρακάτω λύση στοχεύει στην αποδοτικότητα χρόνου εκτέλεσης του query καθώς αυτο το κομμάτι θεωρείτε πιο σημαντικό. Ας σκεφτούμε αν το search επαιρνε αρκετο χρόνο ποσο κακο user-experience θα ήταν!)

### 1η λύση)

Θα χρησιμοποιήσουμε:

- Relational database (ms server)
- Regular expressions
- Indexes (προεραϊτικά για βελτιστοποίηση χρόνου)
- Load balancing (horizontal scaling)

#### Πλεονεκτήματα:

Με τον τρόπο αυτό θα απαντάμε όχι μονο ερωτήματα της μορφής που αναφέρεται στην άσκηση αλλα πολύ περισσότερα! Αυτο φυσικά οφείλεται στα αγαπημένα και πολυχρησιμοποιημένα regex τα οποία μας δίνουν την δυνατότητα να ελέγχουμε οτιδήποτε πανω σε ενα string. Περισσότερες πληροφορίες εδώ

<https://www.sqlshack.com/t-sql-regex-commands-in-sql-server/>

Επίσης σε περίπτωση που κάνουν πολύ χρήστες ταυτόχρονα search θα κάνουμε load-balancing. Με λίγα λόγια θα υπάρχουν clone της βάσης και σε άλλους server. Έτσι λοιπόν οι χρήστες θα λαμβάνουν την καλύτερη εξυπηρέτηση!

#### Μειονεκτήματα:

Απαιτεί την δημιουργία regular expression για να εκτελούνται ερωτήματα. Κατι το οποιο μπορεί ευκολα για καινούργιους προγραμματιστές να οδηγήσει σε λογικά λάθη. Ομως με την χρήση του <https://regexr.com/> μπορούμε ευκολα να τα ελέγξουμε και να ειμαστε σιγουροι οτι ειναι σωστά!

### Αλλαγές στο σχήμα:

Υπενθυμίζουμε ότι η λύση έχει στόχο την ταχύτερη απόδοση.

- Θα φτιάξουμε έναν νέο πίνακα **section** που θα κρατάει το μέρος που βρέθηκε η λέξη καθώς και το χαρακτηριστικό sid(sector-id).

**sections**

sid	section
1	title
2	author
3	series
...	...

- Στην συνέχεια θα χρησιμοποιήσουμε αυτό το **sid** στον πίνακα bibwords για να ξέρουμε από που προήλθε κάθε λέξη. Φυσικά θα κάνουμε αλλαγή στο primary key του σε (wid, bibno, sid). Για visual αναπαράσταση δείτε παρακάτω:

**bibwords**

wid	bibno	sid
4	32	1
222		2
452	87	1
...	...	...

### Πως το χρησιμοποιούμε:

Ας δούμε το πλάνο εκτέλεσης για να απαντήσουμε στο παρακάτω φυσικό ερώτημα.

"Εμφάνισε τον κωδικό και τον τίτλο των βιβλιογραφικών εγγραφών που περιέχουν στον **τίτλο** την λέξη "οικονομία" στην **σειρά** την λέξη "ελληνική" και στον **συγγραφέα** την λέξη "Οικονόμου".

- Φτιαχνουμε regex = "%Οικονόμου%" Βρίσκουμε τα bibno που έχουν **συγγραφέα** την λέξη "Οικονόμου". Αυτό το έχουμε κάνει στα παραπάνω ερωτήματα
- Βρίσκουμε τα bibno που έχουν την λέξη "ελληνική και "οικονομία". Ακριβώς όπως κάναμε στο προηγούμενο ερώτημα(4)
- Παίρνουμε την τομή των 2 μονόστηλων bibno που βρήκαμε από πάνω.
- Για τα bibno που βρήκαμε, παμε στον πίνακα bibrecs και φιλτραρουμε με το regex = "%οικονομία ελληνική%".

Παίρνουμε τα αποτελέσματα άμεσα. Η συγκεκριμένη σειρά είναι σημαντική γιατί εφαρμόζει push down selection και έτσι έχουμε ταχύτερα αποτελέσματα!

Σε διαφορετικά queries θα χρειαζόταν να αλλάξουμε το regex. Η στρατηγική ωστόσο για να πάρουμε τα αποτελέσματα είναι να χωρίσουμε το ερώτημα σε επιμέρους ερωτήματα στα οποία ίσως χρειαστεί η χρήση regex.

### Έξτρα βελτιστοποιήσεις:

- Να γίνεται paging για την ταχύτερη μεταφορά δεδομένων.
- Τα πιο δημοφιλή βιβλία, που είναι πιο πιθανό να ψαχτούν, να γράφονται σε VIEW έτσι ώστε να έχουμε γρηγορότερα αποτελέσματα.
- Προσθήκη index

```
CREATE INDEX bibrects_title_bibno  
ON bibrecs(title)  
INCLUDE(bibno)
```



```
CREATE INDEX bibwords_bibno  
ON bibrecs(bibno)
```

## 2η λύση)

Θα χρησιμοποιήσουμε:

- ElasticSearch | Sphinx
- Full-text search
- MongoDB

<http://www.elasticsearch.org/>

<http://sphinxsearch.com/>

### Περιγραφή:

Ανάλογα με το εάν έχουμε την ελευθερία να το κάνουμε, θα πρότεινα να χρησιμοποιηθεί κάτι που έχει σχεδιαστεί για αναζήτηση αντί να κάνουμε αναζητήσεις πλήρους κειμένου στη βάση δεδομένων. Οι βάσεις δεδομένων δεν είναι πραγματικά σχεδιασμένες για αναζητήσεις πλήρους κειμένου, επομένως η απόδοση δεν θα είναι η καλύτερη και μπορεί να φορτώσει αρκετά την βάση.

## Ποια απο τις δυο μέθοδοι είναι καλύτερη?)

Αξίζει να ρίξετε μια ματιά στο παρακάτω link που εξηγεί περιγραφικά την διαφορά τους

<https://stackoverflow.com/questions/224714/what-is-full-text-search-vs-like>