# Αναφορά Β Φάσης Εργαστηριακής Εργασίας του μαθήματος «Βάσεις Δεδομένων»

Ομάδα (LAB30232464):

|  |  |
| --- | --- |
| Χρήστος Ζαχαριουδάκης | 2014030056 |
| Ιωάννης Μισόκαλος | 2014030138 |

## Ζήτημα 2 :

Για την καλύτερη μελέτη της απόδοσης των ερωτήσεων ανάκτησης 2.1 , 2.2 και 2.3 με χρήση ευρετηρίων (indices) και ομαδοποίησης (clustering) , προσθέσαμε 100000 εγγραφές στους πίνακες “Student” , “Labstaff” , “Professor” , “Register” και “CourseRun”. Για να διευκολυνθούμε στις εισαγωγές και να αποφύγουμε τυχών διαγραφές απενεργοποιήσαμε προσωρινά τα εναύσματα (triggers) των πινάκων “Register” και “CourseRun” , καθώς και τους περιορισμούς ξένου κλειδιού των attributes course\_code του “CourseRun” και Student\_amka , serial\_number του “Register” .

### Ερώτημα 2.1 :

Ο κώδικας SQL του ερωτήματος είναι ο εξής :

SELECT "Student".name, "Student".email, "Student".father\_name, "Student".surname, "Student".entry\_date, "Student".am, "Student".amka

FROM "Student"

**WHERE "Student".am = am\_param**

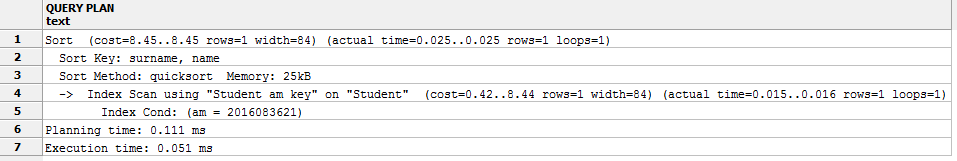
ORDER by surname, name

Όπως βλέπουμε , έχουμε ένα ερώτημα στο WHERE και συγκεκριμένα ένα ερώτημα ισότητας . Επομένως για την επιτάχυνση του ερωτήματος , το βέλτιστο που μπορούμε να κάνουμε είναι να δημιουργήσουμε **ένα ευρετήριο κατακερματισμού (hash index)** στο attribute am .

Ωστόσο , επειδή θέλουμε να μελετήσουμε τη περίπτωση που έχουμε ομαδοποιήσει τις πλειάδες ως προς ένα ευρετήριο , δοκιμάζουμε και την περίπτωση των B+-Tree ευρετηρίων , καθώς η Postgresql δεν επιτρέπει ομαδοποίηση με hash ευρετήρια . Αξίζει να σημειωθεί ότι τα Β+-Tree ευρετήρια λειτουργούν καλά στα ερωτήματα ισότητας αλλά όχι τόσο καλά όσο το ευρετήριο κατακερματισμού . Το B-Tree index μπορεί να χρησιμοποιηθεί σε ερωτήματα εύρους , ενώ το hash index δεν μπορεί .

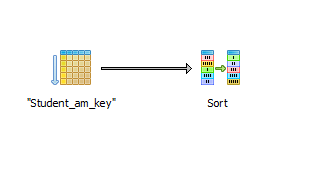
Οπότε με χρήση της εντολής ΕΧPLAIN της Postgres μελετήσαμε τα ακόλουθα πλάνα εκτέλεσης :

1. Εκτέλεση του ερωτήματος χωρίς δημιουργία ευρετηρίων ή συσταδοποίηση

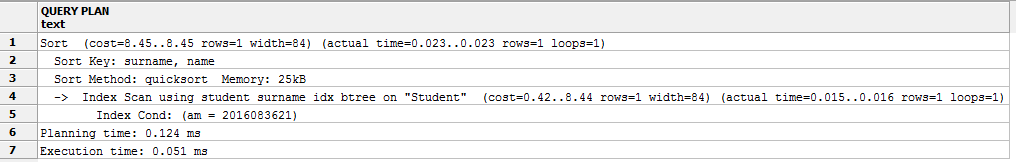


Παρατηρούμε ότι o βελτιστοποιητής (optimizer) χρησιμοποιεί Index Scan στο am , παρόλο που δεν έχουμε ορίσει ευρετήρια . Αυτό συμβαίνει επειδή το am είναι κλειδί του πίνακα “Student” και συγκεκριμένα υποψήφιο κλειδί (Candidate key) , επειδή το έχουμε ορίσει ως UNIQUE . To σύστημα της Postgres δημιουργεί αυτόματα ευρετήρια για τα κλειδιά των πινάκων. Η ύπαρξη ευρετηρίου σε συνδυασμό με την απλότητα του ερωτήματος , οδηγεί σε εξαιρετικά μικρό χρόνο εκτέλεσης .

Γραφικά το πλάνο εκτέλεσης είναι το εξής :

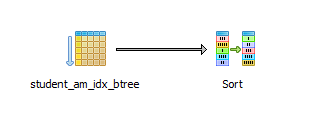


1. Εκτέλεση του ερωτήματος με χρήση B+-Tree ευρετηρίου χωρίς συσταδοποίηση

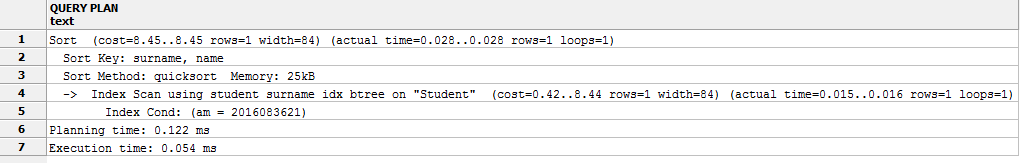


Παρατηρούμε ότι o βελτιστοποιητής (optimizer) χρησιμοποιεί Index Scan χρησιμοποιώντας το ευρετήριο που ορίσαμε. Η ύπαρξη ευρετηρίου σε συνδυασμό με την απλότητα του ερωτήματος , οδηγεί σε εξαιρετικά μικρό χρόνο εκτέλεσης . Οι χρόνοι είναι ήδη πολύ μικροί , οπότε δεν παρατηρούμε σημαντικές αποκλίσεις ανάμεσα στις περιπτώσεις .

Γραφικά το πλάνο εκτέλεσης είναι το εξής :

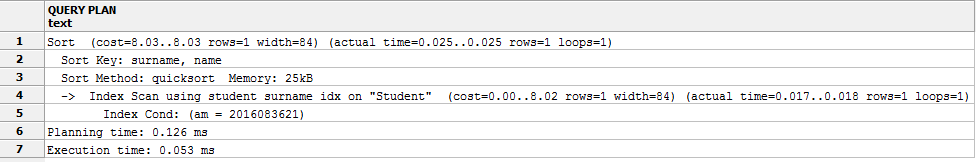


1. Εκτέλεση του ερωτήματος με χρήση B+-Tree ευρετηρίου με συσταδοποίηση



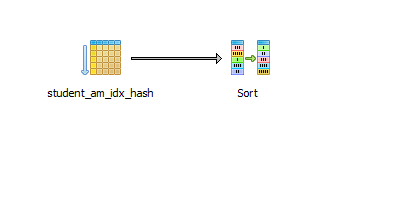
Παρατηρούμε ότι o βελτιστοποιητής (optimizer) χρησιμοποιεί Index Scan χρησιμοποιώντας το ευρετήριο που ορίσαμε. Η ύπαρξη ευρετηρίου σε συνδυασμό με την απλότητα του ερωτήματος , οδηγεί σε εξαιρετικά μικρό χρόνο εκτέλεσης . Οι χρόνοι είναι ήδη πολύ μικροί , οπότε δεν παρατηρούμε σημαντικές αποκλίσεις ανάμεσα στις περιπτώσεις . Για αυτό το λόγο , παρόλο που μετά την συσταδοποίηση αναμέναμε μικρότερους χρόνους , επειδή τα δεδομένα (πλειάδες) με το ίδιο κλειδί (am) είναι συγκεντρωμένα πλέον στην ίδια σελίδα/ες στον δίσκο , ο χρόνος εκτέλεσης παραμένει λίγο πολύ ο ίδιος .

1. Εκτέλεση του ερωτήματος με χρήση ευρετηρίου κατακερματισμού χωρίς συσταδοποίηση



Παρατηρούμε ότι o βελτιστοποιητής (optimizer) χρησιμοποιεί Index Scan χρησιμοποιώντας το ευρετήριο που ορίσαμε. Η ύπαρξη ευρετηρίου σε συνδυασμό με την απλότητα του ερωτήματος , οδηγεί σε εξαιρετικά μικρό χρόνο εκτέλεσης . Οι χρόνοι είναι ήδη πολύ μικροί , οπότε δεν παρατηρούμε σημαντικές αποκλίσεις ανάμεσα στις περιπτώσεις . Για αυτό το λόγο , παρόλο που μετά την χρήση του ευρετηρίου κατακερματισμού αναμέναμε μικρότερους χρόνους επειδή έχουμε ερώτημα ισότητας , ο χρόνος εκτέλεσης παραμένει περίπου ο ίδιος .

Γραφικά το πλάνο εκτέλεσης είναι το εξής :



### Ερώτημα 2.2 :

Ο κώδικας SQL του ερωτήματος είναι ο εξής :

SELECT "Student".name, "Student".surname, "Student".am

FROM "Student", "Register", "CourseRun", "Semester"

WHERE "CourseRun".course\_code = tempcode

AND "CourseRun".semester\_id = "Semester".semester\_id

AND "Semester".semester\_status = 'present'

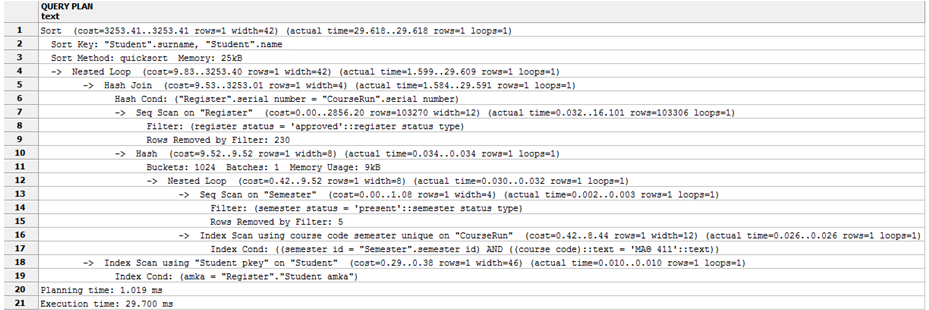
AND "CourseRun".serial\_number = "Register".serial\_number

AND "Register"."Student\_amka" = "Student".amka

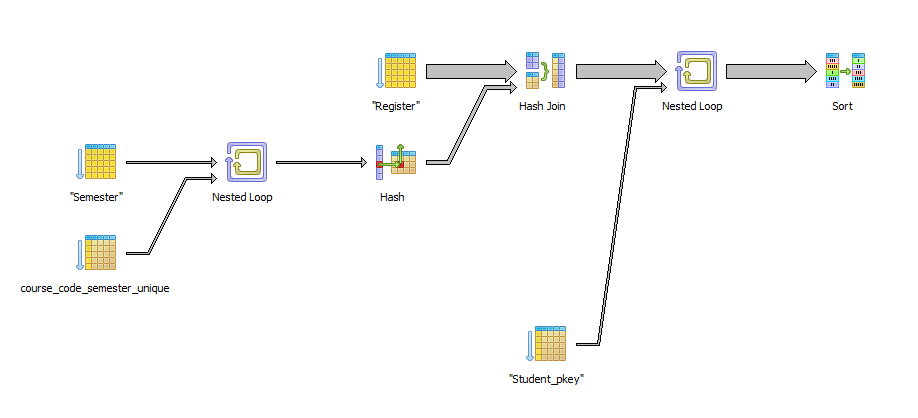
AND "Register".register\_status = 'approved'

ORDER BY surname , name

Στο ερώτημα αυτό παρατηρούμε ότι για να λάβουμε την επιθυμητή πληροφορία έχουμε join μεταξύ των πινάκων "Student", "Register", "CourseRun" και "Semester" χρησιμοποιώντας ως join attributes το semester\_id , Student\_amka και serial\_number. Για να βελτιώσουμε την απόδοση του ερωτήματος έχει νόημα να δημιουργήσουμε ευρετήρια και να κάνουμε clustering σε αυτά τα attributes . Αν εκτελέσουμε το ερώτημα χωρίς ευρετήρια ή συσταδοποίηση έχουμε :

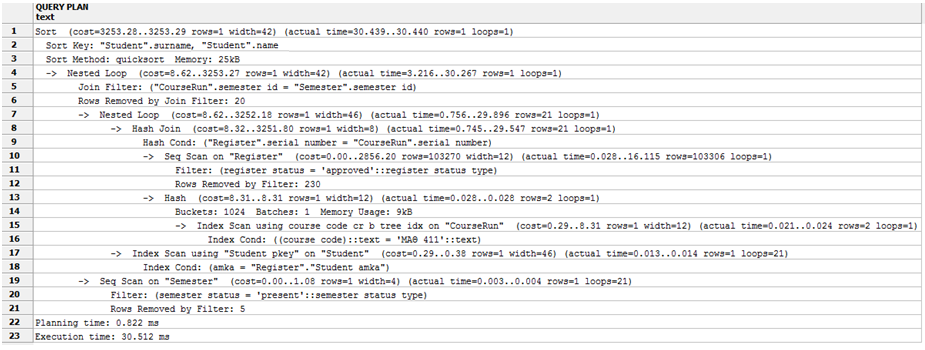


Η αντίστοιχη γραφική αναπαράσταση είναι :



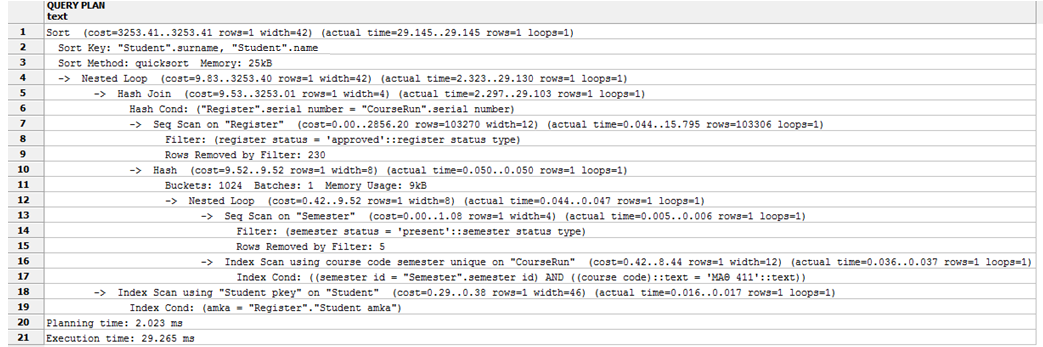
Επομένως δοκιμάζουμε κάθε φορά ένα ευρετήριο σε καθένα από τα παραπάνω καθώς και σε μερικά άλλα attributes που συμμετέχουν στο ερώτημα (course\_code , register\_status) attributes για να δούμε ποιο έχει την μεγαλύτερη επίδραση στην απόδοση . Προτιμήσαμε B+-Tree , ώστε να μπορούμε να κάνουμε συσταδοποίηση με το ευρετήριο που θα προκύψει γρηγορότερο . Συγκεκριμένα :

1. Ευρετήριο στο course\_code χωρίς συσταδοποιήση



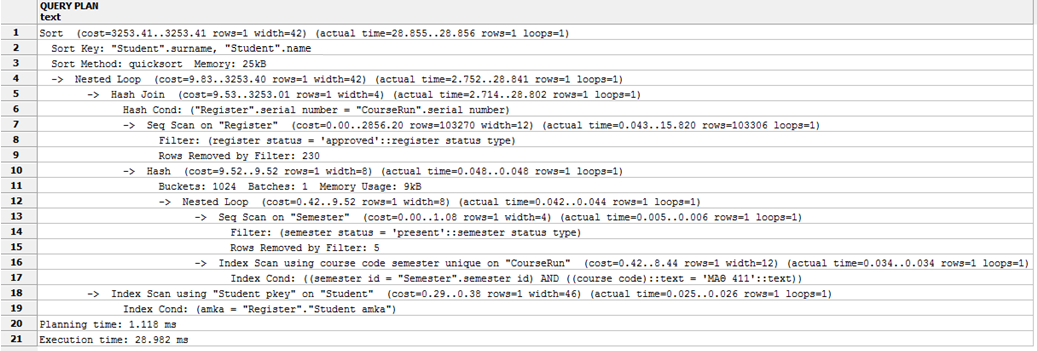
Παρατηρούμε περίπου τον ίδιο χρόνο . Αυτό είναι αναμενόμενο καθώς το course\_code λαμβάνει συγκεκριμένη τιμή στο ερώτημα και δεν είναι join attribute .

1. Ευρετήριο στο register\_status χωρίς συσταδοποιήση



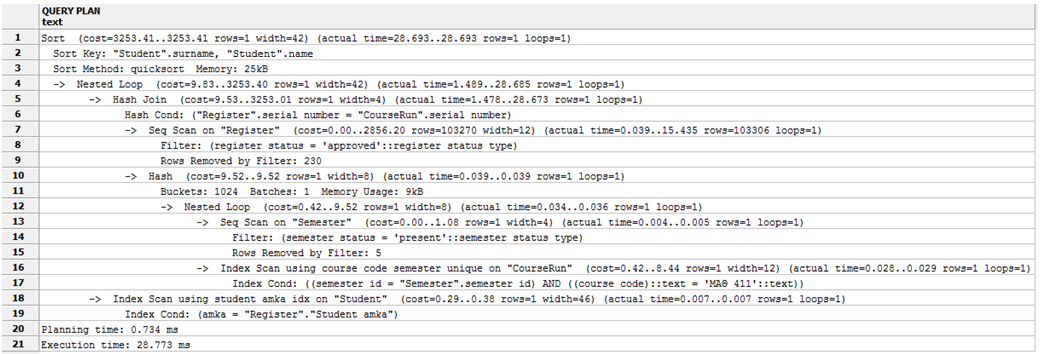
Παρατηρούμε περίπου τον ίδιο χρόνο . Αυτό είναι αναμενόμενο , καθώς το register\_status λαμβάνει συγκεκριμένη τιμή στο ερώτημα και δεν είναι join attribute.

1. Ευρετήριο στο “Student\_amka” χωρίς συσταδοποιήση



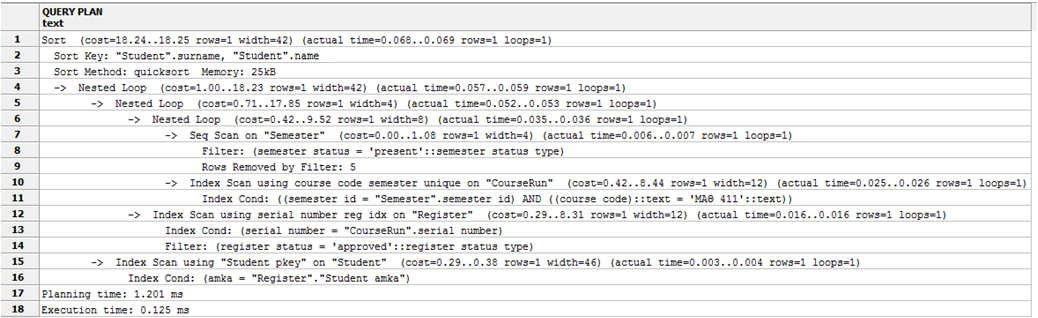
Παρατηρούμε ελάχιστα καλύτερο τον ίδιο χρόνο . Αυτό είναι αναμενόμενο , καθώς το “Student\_amka” είναι join attribute , αλλά δεν περιορίζει σημαντικά τον αριθμό των πλειάδων στο Register που λαμβάνει το ερώτημα , καθώς ένας μαθητής μπορεί να έχει εγγραφεί πολλές φορές σε διαφορετικά (ή και σε ίδια αν έχει αποτύχει) μαθήματα.

1. Ευρετήριο στο amka του πίνακα Student χωρίς συσταδοποιήση



Παρατηρούμε τον ίδιο χρόνο με την προηγούμενη δοκιμή και παρόμοιο χρόνο με τα υπόλοιπα . Αυτό είναι αναμενόμενο , καθώς το amka είναι join attribute , που περιορίζει σημαντικά τον αριθμό των πλειάδων στο Student που λαμβάνει το ερώτημα, αλλά αποτελεί και κλειδί του πίνακα , το οποίο σημαίνει ότι το ίδιο το σύστημα βάσεων δεδομένων δημιούργησε index για το κλειδί , όταν το ορίσαμε , το οποίο χρησιμοποιεί χωρίς εισάγουμε εμείς κάποιο ευρετήριο . Παρόμοια αποτελέσματα είδαμε όταν φτιάξαμε ευρετήριο στο serial\_number του CourseRun επειδή είναι κλειδί (μέρος του κλειδιού) και UNIQUE .

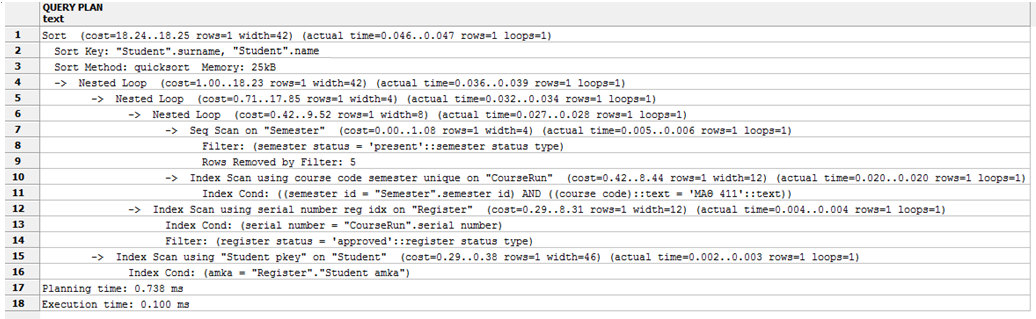
1. Ευρετήριο στο serial\_number του πίνακα “Register” χωρίς συσταδοποίηση



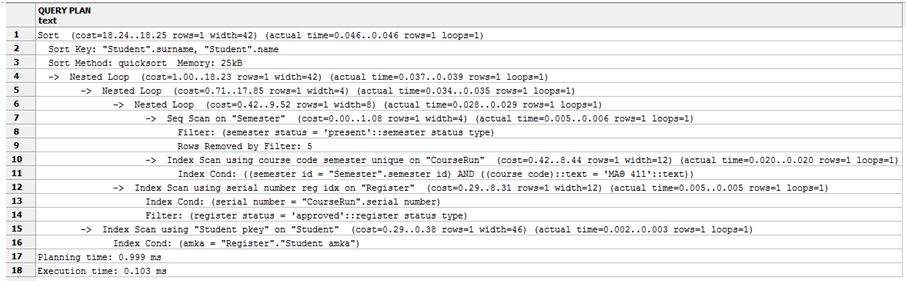
Παρατηρούμε εξαιρετικά μικρότερο χρόνο σε σχέση με τις υπόλοιπες δοκιμές . Αυτό είναι αναμενόμενο , καθώς το serial\_number είναι join attribute , που περιορίζει σημαντικά τον αριθμό των πλειάδων στο Register που λαμβάνει το ερώτημα , καθώς μας επιστρέφει τις αιτήσεις ενός συγκεκριμένου CourseRun του παρόντος εξαμήνου , οι οποίες είναι πολύ λιγότερες σε σχέση με το σύνολο των πλειάδων του Register. H ιδιότητα αυτή ονομάζεται Selectivity. Επομένως για τις επόμενες αναλύσεις , θα χρησιμοποιήσουμε ευρετήρια στο serial\_number .

Έχοντας διαπιστώσει ότι το γρηγορότερο ευρετήριο είναι στο serial\_number του Register , δοκιμάζουμε τα εξής για να βελτιώσουμε την απόδοση :

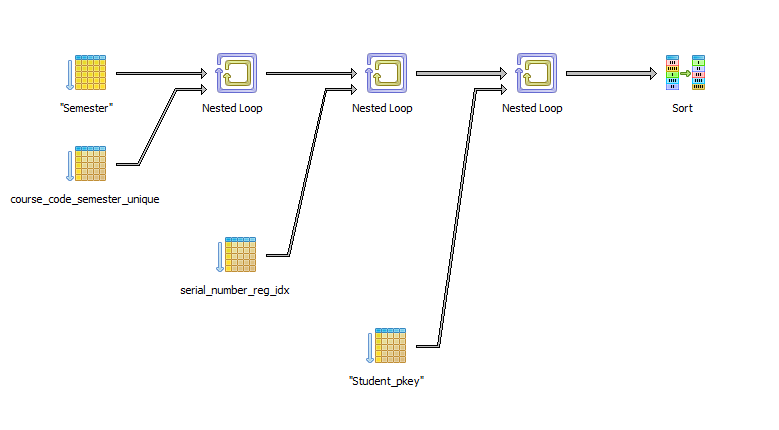
1. B+-Tree index στο serial\_number χωρίς συσταδοποιηση



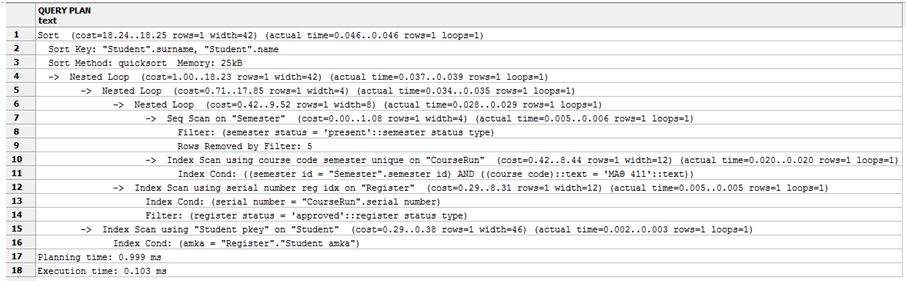
1. B+-Tree index στο serial\_number με συσταδοποιηση



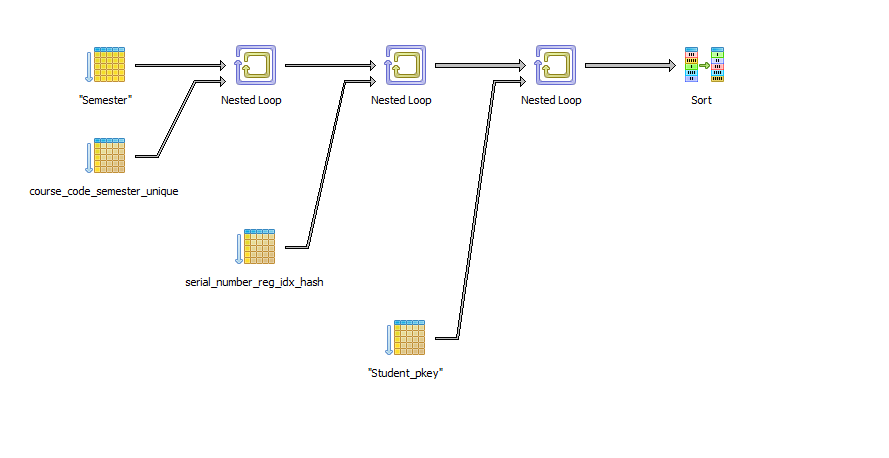
Η αντίστοιχη γραφική αναπαράσταση του EXPLAIN :



1. Hash index στο serial\_number

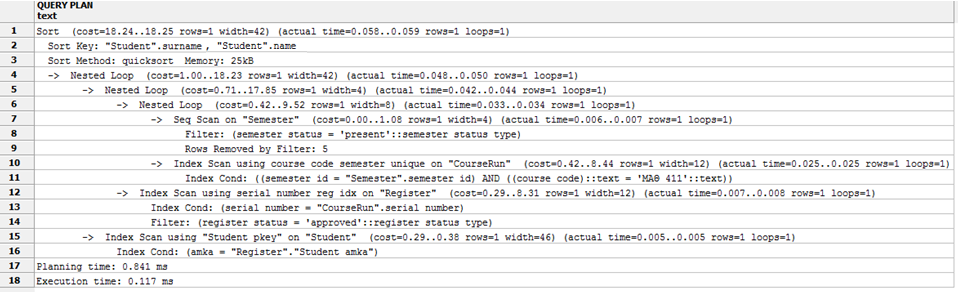


Η γραφική αναπαράσταση του EXPLAIN :

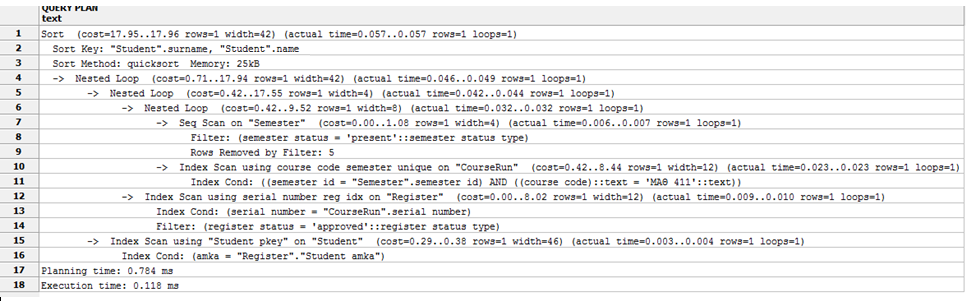


### Έπειτα δοκιμάσαμε την απόδοση αλλάζοντας την σειρά των joins :

Β+-Tree index και Clustering :



Hash index :



Η διαφορές στους χρόνους είναι ελάχιστες .

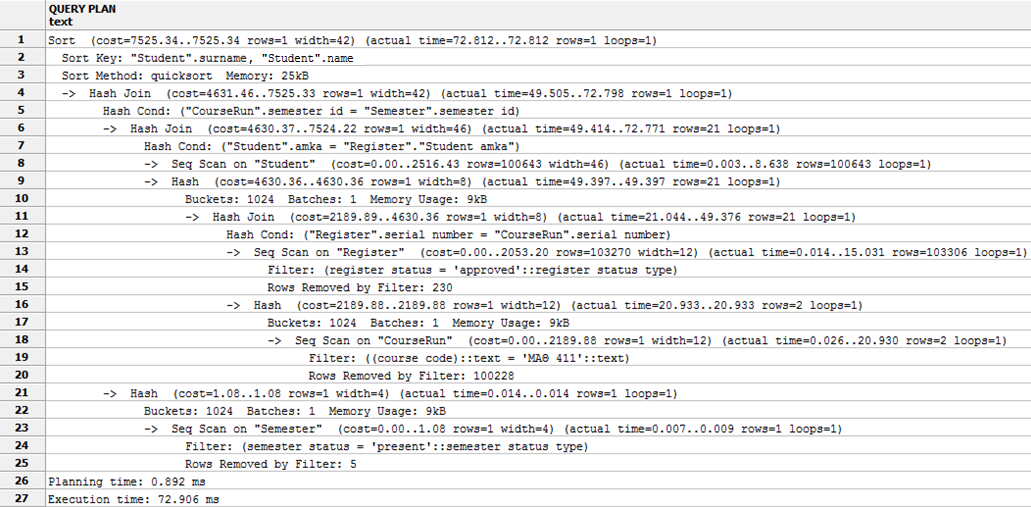
Τέλος δοκιμάζουμε την απόδοση του ερωτήματος απενεργοποιώντας ορισμένους αλγορίθμους join με τις εντολές :

* SΕΤ enable\_hashjoin = off
* SΕΤ enable\_mergejoin = off
* SΕΤ enable\_nestloop = off

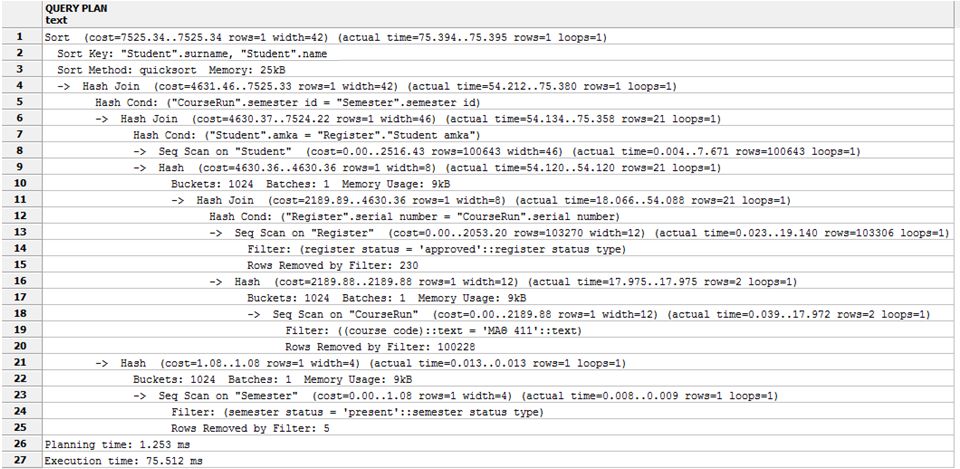
για τους αλγορίθμους hash join , Sort-Merge join και nested loop join αντίστοιχα . Η ανάλυση αυτή είναι χρήσιμη καθώς ο optimizer επιλέγει συνήθως τον βέλτιστο αλγόριθμο συνένωσης , αλλά όχι πάντα . Με την παρακάτω μελέτη , θα διαπιστώσουμε αν ο βελτιστοποιητής έκανε την καλύτερη επιλογή .

Παρατηρούμε ότι στην εκτέλεση του ερωτήματος χρησιμοποιεί nested loop join . Επομένως για την πρώτη δοκιμή απενεργοποιούμε το nested loop join και εκτελούμε το ερώτημα :

Με B+-Tree ευρετήριο :

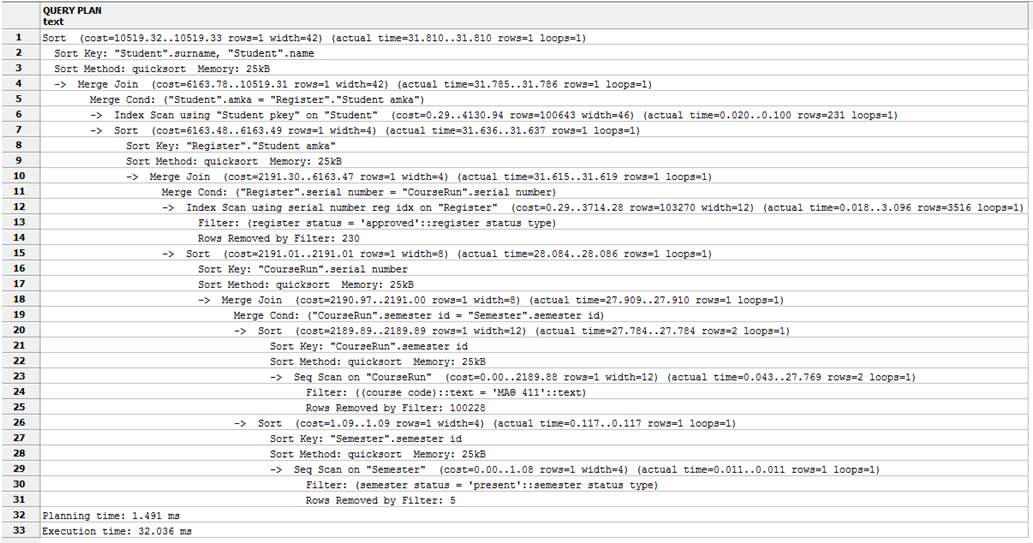


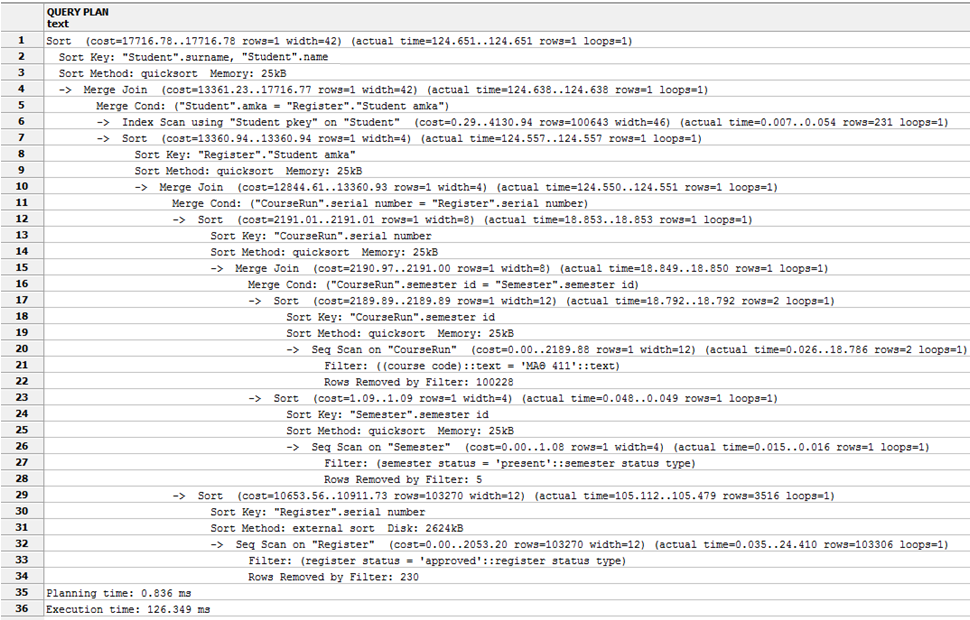
Με hash ευρετήριο :



Παρατηρούμε ότι ο optimizer χρησιμοποιεί τον αλγόριθμο hash join . Ωστόσο η απόδοση έχει χειροτερεύσει καθώς με χρήση του αλγορίθμου nested loop join ο χρόνος ήταν κοντά στα 0,100 ms . Οι χρόνοι με τα δυο διαφορετικά ευρετήρια είναι οι ίδιοι . Για την τελευταία δοκιμή απενεργοποιούμε το hash join και εκτελούμε το ερώτημα :

Με B+-Tree ευρετήριο :



Με hash ευρετήριο :

Παρατηρούμε ότι ο optimizer χρησιμοποιεί τον αλγόριθμο merge join . Ωστόσο η απόδοση έχει χειροτερεύσει καθώς με χρήση του αλγορίθμου nested loop join ο χρόνος ήταν κοντά στα 0,100 ms . Οι χρόνοι με τα δυο διαφορετικά ευρετήρια όμως διαφέρουν , καθώς με B+-Tree o χρόνος είναι πολύ καλύτερος . Αυτό πιθανόν συμβαίνει επειδή ο αλγόριθμος Sort-Merge απαιτεί ταξινόμηση των δεδομένων , η οποία δύσκολα γίνεται με hash , αλλά ιδανικά γίνεται με B+-Tree το οποίο είναι βέλτιστο σε ερωτήματα εύρους .

Τελικά διαπιστώσαμε ότι ο optimizer έκανε την βέλτιστη επιλογή αλγορίθμου συνένωσης (Νested Loop Join) .

### Ερώτημα 2.3 :

### 

Ο κώδικας SQL του ερωτήματος είναι ο εξής :

((SELECT "Student".surname, "Student".name, 'Student'

FROM "Student")

UNION

(SELECT "Professor".surname ,"Professor".name , 'Professor'

FROM "Professor")

UNION

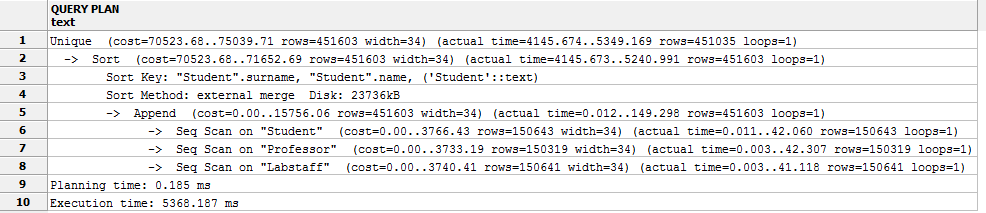
(SELECT "Labstaff".surname ,"Labstaff".name, 'Labstaff'

FROM "Labstaff"))

ORDER BY surname , name

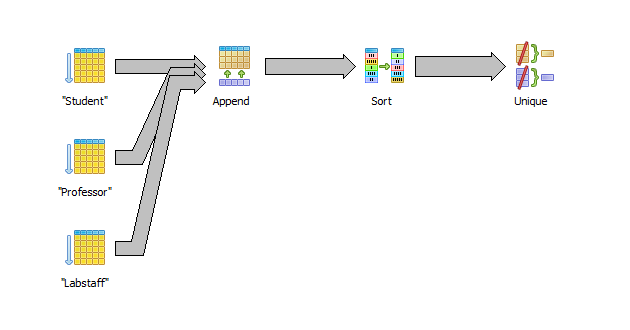
Παρατηρούμε ότι στο παραπάνω ερώτημα , δεν έχουμε ούτε join ούτε κάποιο ερώτημα σε WHERE . Επομένως ο μοναδικός τρόπος να επιταχύνουμε το ερώτημα είναι με δημιουργία ευρετηρίων Β+-Trees στα select attributes ,του κάθε πίνακα και με ομαδοποίηση των πλειάδων χρησιμοποιώντας αυτά.

1. Εκτέλεση του ερωτήματος χωρίς χρήση ευρετηρίου ή συσταδοποίησης (και στους τρείς πίνακες)

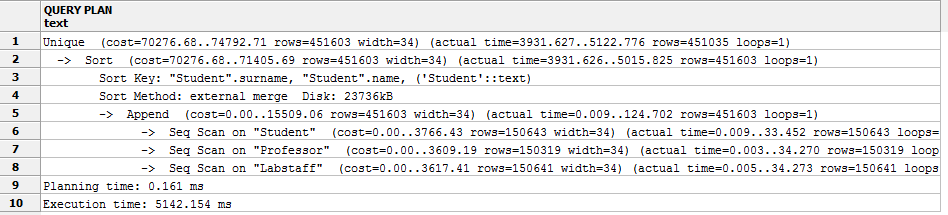


Παρατηρούμε ότι έχουμε εξαιρετικά μεγάλους χρόνους σε σχέση με τα άλλα ερωτήματα . Αυτό συμβαίνει λόγω της εκτύπωσης όλων των πλειάδων των πινάκων “Labstaff” , “Student” και “Professor” , οι οποίες είναι τουλάχιστον 300000 με χρήση σειριακής αναζήτησης (Sequential scan) .

Γραφικά το πλάνο εκτέλεσης είναι το εξής :

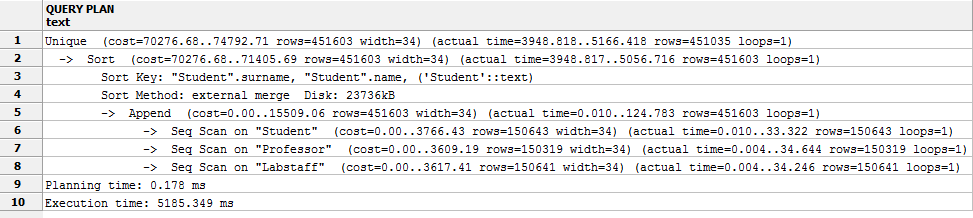


1. Εκτέλεση του ερωτήματος με χρήση ευρετηρίου στο name με συσταδοποίηση (και στους τρείς πίνακες)



Παρατηρούμε μείωση του χρόνου , την οποία αναμέναμε λόγω του clustering .

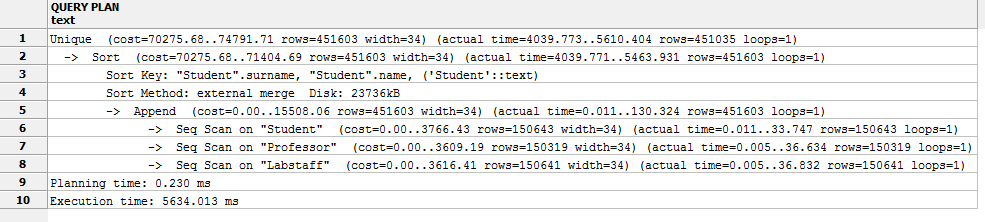
1. Εκτέλεση του ερωτήματος με χρήση ευρετηρίου στο surname με συσταδοποίηση (και στους τρείς πίνακες)



Παρατηρούμε μείωση του χρόνου σε σχέση με το 1 , την οποία αναμέναμε λόγω του clustering . Ο χρόνος είναι περίπου ίδιος με την περίπτωση 2 .

(Μικρό Selectivity του name , δηλαδή υπάρχουν πολλές πλειάδες με την ιδιά τιμή name, άρα δεν μειώνει σημαντικά τον αριθμό των πλειάδων που επιστρέφει η αναζήτηση . )

1. Εκτέλεση του ερωτήματος με χρήση σύνθετου ευρετηρίου στο (name,surname) με συσταδοποίηση (και στους τρείς πίνακες)



Παρατηρούμε μια αύξηση στον χρόνο εκτέλεσης σε σχέση με τις περιπτώσεις 2 και 3 .

### Ζήτημα 3 :

Ο κώδικας SQL του ερωτήματος είναι ο εξής :

(SELECT St.amka ,St.am , St.name , St.surname , St.father\_name , St.email , St.entry\_date , count(\*) as numOfLessons ,sum(units) as TotalUnits

FROM "Student" St , "Register" R, "CourseRun" CR , "Course" C

WHERE St.surname like '%ΑΛ%'

AND R."Student\_amka" = St.amka

AND CR.course\_code = C.course\_code

AND R.serial\_number = CR.serial\_number

AND R.register\_status = 'approved'

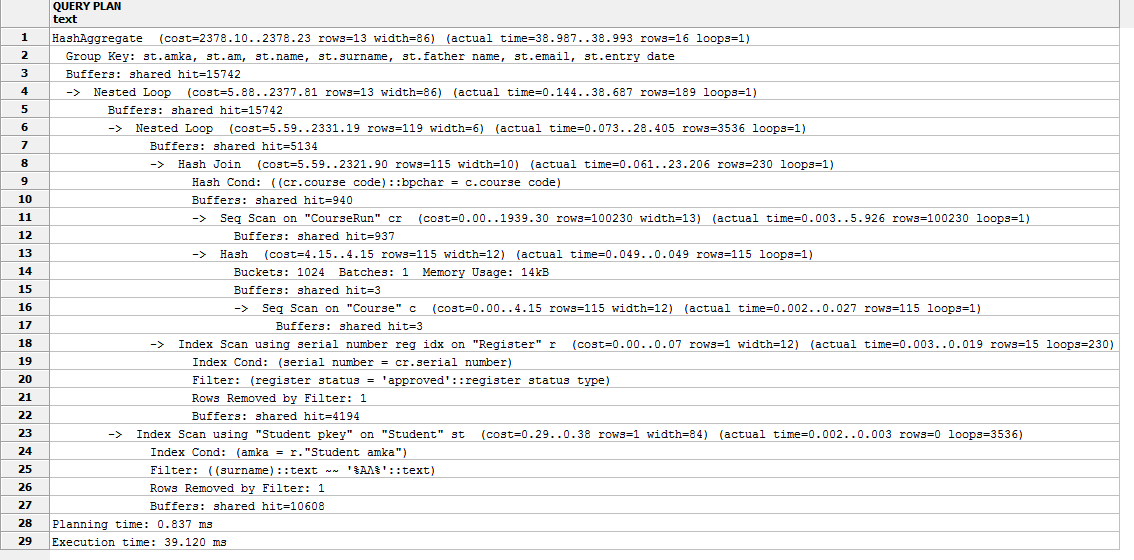
GROUP BY St.amka ,St.am , St.name , St.surname , St.father\_name , St.email , St.entry\_date )

### Εκτελούμε αυτό το ερώτημα με τρείς τρόπους :

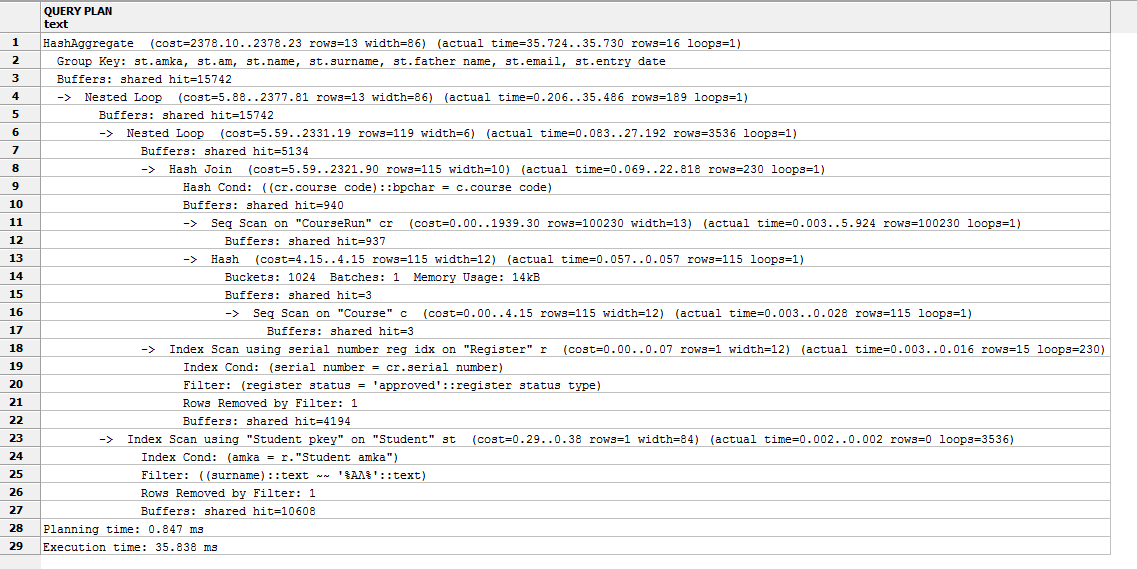
* Ως ερώτημα της SQL
* Ως virtual view
* Ως materialized view

με χρήση της εντολής EXPLAIN (ANALYZE,BUFFERS) και λαμβάνουμε τα εξής αποτελέσματα:

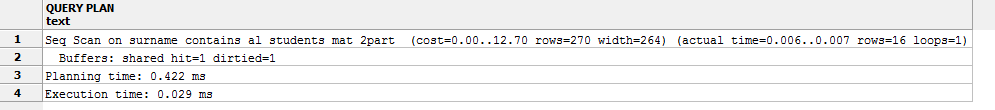
1. Ερώτημα SQL



1. Virtual View



1. Materialized View

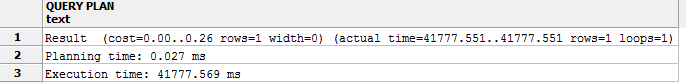


Οι χρόνοι των δύο πρώτων περιπτώσεων είναι σχεδόν ίδιοι , καθώς εκτελούν το ίδιο ερώτημα κάθε φορά που καλούνται . Αντίθετα , στη τελευταία περίπτωση , δηλαδή στη περίπτωση του materialized view παρατηρούμε εξαιρετικά μικρό χρόνο εκτέλεσης , παρόλο που εκτελεί το ίδιο ερώτημα . Αυτό συμβαίνει επειδή το materialized view αποθηκεύεται μόνιμα ως πίνακας στην βάση όταν δημιουργηθεί , ενώ το ερώτημα και το virtual view εκτελούνται κάθε φορά από την αρχή , σε κάθε χρήση τους (view data) . Η ιδιότητα αυτή του materialized view το καθιστά εξαιρετικά γρήγορο . Όμως έχει το μειονέκτημα ότι δεν είναι πάντα ενημερωμένο με τα πιο πρόσφατα δεδομένα , ενώ το Virtual View είναι πάντα ενημερωμένο , αλλά καθυστερεί να εκτελεστεί σε σχέση με το materialized.

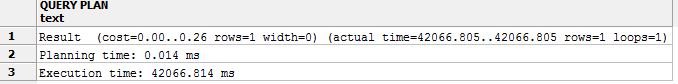
### Ζήτημα 4 :

1. Εισαγωγή με ευρετήρια

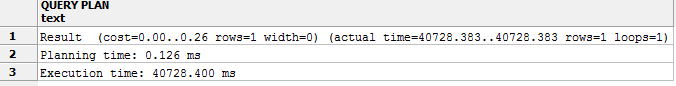
Ηash



B+-Tree



1. Εισαγωγή χωρίς ευρετήρια



Στη μελέτη της επίδρασης των ευρετηρίων πάνω στην επίδοση των λειτουργιών εισαγωγής τα αποτελέσματα που πήραμε ήταν αυτά που περιμέναμε έχοντας διαβάσει τη θεωρία του μαθήματος πάνω στα ευρετήρια. Τα ευρετήρια σε μεγάλες σχέσεις μπορούν να δώσουν πιο γρήγορες απαντήσεις σε ερωτήματα αναζήτησης , αλλά σε μαζικές εισαγωγές (ή ακόμη και σε διαγραφές ή ενημερώσεις) , όπως και σε αυτή του ερωτήματος μπορούν να επιβαρύνουν την απόδοση του συστήματος διαχείρισης βάσεων δεδομένων. Έτσι στις δύο μεθόδους εισαγωγών ήταν ξεκάθαρο ότι η εισαγωγή χωρίς ευρετήρια ήταν πιο γρήγορη, αλλά δεν παρατηρήσαμε ουσιαστικές διαφορές, λόγω του ήδη μεγάλου χρόνου διεκπεραίωσης της “μαζικής” εισαγωγής . Εισήγαμε 10000 στοιχεία στον κάθε πίνακα (Student, Labstaff, Professor), παρόλα αυτά όσο αυξάνεται το ποσό των στοιχείων εισαγωγής θα παρατηρούμε μεγαλύτερη διαφορά στις επιδόσεις του ερωτήματος 1.1.

Γενικά η εντολή insert δεν μπορεί να ωφεληθεί από τα indexes, λόγω της έλλειψης where clause. Κατά την εισαγωγή ενός νέου στοιχείου, η insert απλά του καταχωρεί ένα table block με αρκετό χώρο, αλλά όταν υπάρχουν indexes πρέπει η insert να κάνει εφικτή την εύρεση του στοιχείου μέσω των indexes που υπάρχουν, οπότε προσθέτει τη νέα εισαγωγή σε κάθε index , δηλαδή τα ανανεώνει. Ο αριθμός των index λοιπών λειτουργεί πολλαπλασιαστικά για το κόστος του insert. Επίσης, στην περίπτωση των ευρετηρίων δεν αρκεί η καταγραφή του νέου στοιχείου σε οποιοδήποτε κενό block βρεθεί πρώτο, αλλά πρέπει να βρεθεί το σωστό leaf node, που απαιτεί περισσότερο χρόνο. Καταλαβαίνουμε λοιπόν, ότι για μεγάλο αριθμό εισαγωγών η διαφορά θα είναι αντιληπτή.

Τέλος παρατηρούμε ότι οι χρόνοι εκτέλεσης με ευρετήρια hash και B+-Tree είναι περίπου οι ίδιοι , αλλά μεγαλύτεροι από τον χρόνο εισαγωγής χωρίς ευρετήρια .