ΠΛΗ302 Βάσεις Δεδομένων
Αναφορά Δεύτερης Φάσης Εργαστηριακής Άσκησης

Πανταζής Θεόδωρος ΑΜ 2005030004
Παντουράκης Μιχαήλ ΑΜ 2015030185
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Πολυτεχνείο Κρήτης

12 Μαΐου 2017

1 Ζητούμενο 2

Στόχος αυτού του ζητούμενου είναι η κατανόηση της ανάλυσης απόδοσης SQL ερωτημάτων, καθώς και η χρήση ευρετηρίων και συσταδοποίησης για την περαιτέρω βελτίωσής τους. Οι εκτιμήσεις κόστους και οι πραγματικοί χρόνοι που αναφέρονται στις επόμενες υποενότητες έχουν όλες προκύψει μετά από χρήση της εντολής VACUUM ANALYZE και επανάληψη κάθε ερωτήματος δύο φορές (δηλαδή γίνεται χρήση της cache στους αναφερόμενους πραγματικούς χρόνους).

1.1 Βελτίωση απόδοσης ερωτήματος 2.1

Το ερώτημα 2.1 αφορά την εύρεση των στοιχείων ενός φοιτητή δίνοντας ως είσοδο το ΑΜ του. Το ερώτημα αυτό είναι πολύ απλό καθώς απαιτεί μόνο την ανάγνωση του πίνακα "Student", αποκλειστικά με βάση την στήλη am. Με χρήση της εντολής explain analyze για ένα τυχαίο ΑΜ βλέπουμε ότι το ερώτημα είναι ήδη αρκετά γρήγορο καθώς χρησιμοποιείται ευρετήριο για τη στήλη am (Παράρτημα 4.1). Το ευρετήριο αυτό έχει δημιουργηθεί αυτόματα από το σύστημα, καθώς η postgres δημιουργεί αυτόματα b-tree ευρετήριο (πέρα των πρωτευόντων κλειδιών) στα unique χαρακτηριστικά.

Επιπλέον μικρή μείωση του κόστους μπορεί να επιτευχθεί με χρήση ευρετηρίου κατακερματισμού (Παράρτημα 4.2), καθώς αυτό το είδος ευρετηρίου είναι ιδανικό για την εύρεση μικρού πλήθους τυχαίων τιμών. Παρ΄ όλα αυτά, στην παραδοτέα βάση δεν διατηρούμε το ευρετήριο αυτό, καθώς θεωρούμε τη βελτίωση μη σημαντική ενώ πιθανώς να επιβαρυνθεί δυσανάλογα η απόδοση εισαγωγών/διαγραφών στον πίνακα.

Τέλος, στο ερώτημα αυτό δεν ασχολούμαστε με συσταδοποίηση του πίνακα και ταξινόμηση των αποτελεσμάτων (δηλαδή δεν κάνουμε χρήση της εντολής order by), καθώς πάντα επιστρέφεται το πολύ μία σειρά.

1.2 Βελτίωση απόδοσης ερωτήματος 2.2

Στο ερώτημα 2.2 αναζητούμε τους φοιτητές που έχουν εγγραφεί σε μαθήματα του τρέχοντος εξαμήνου. Έτσι από το σχήμα της βάσης συμμετείχαν σε αυτό το ερώτημα τέσσερις σχέσεις, "CourseRun", "Register", "Student", "Semester" με πλήθος πλειάδων 920, 177013, 150400 και 15000 αντίστοιχα. Στην συνέχεια παρουσιάζουμε τα βήματα βελτίωσης απόδοσης.

1) Ξεκινώντας με το αρχικό πλάνο εκτέλεσης του ερωτήματος 2.2 (Παράρτημα 4.3) παρατηρούμε ότι πραγματοποιούνται δύο nested loop joins, τα οποία και παρουσιάζουν σημαντική καθυστέρηση. Ως πρώτη επέμβασή μας για να βελτιώσουμε το χρόνο εκτέλεσης των joins δημιουργήσαμε b-tree index πάνω στη στήλη serial_number του "Register". Το ευρετήριο αυτό μειώνει τόσο το χρόνο του join μεταξύ "CourseRun"-"Register", αλλά επίσης επιτρέπει στο βελτιστοποιητή να χρησιμοποιήσει τον αλγόριθμο sort-merge join μεταξύ "CourseRun" και "Student" (χρησιμοποιώντας τα υπάρχοντα primary key indices). Αυτή η προσθήκη έφερε και την πιο σημαντική βελτίωση, καθώς το εκτιμώμενο κόστος (και αντίστοιχα ο πραγματικός χρόνος) έπεσε κατά περίπου 10 φορές!

2) Στη συνέχεια προχωράμε στη βελτίωση του χρόνου εκτέλεσης του εναπομένοντος nested loop. Προσθέσαμε λοιπόν b-tree index στο course_code του "Courserun", ο οποίος και έριξε το συνολικό κόστος του nested loop από 37,55 σε 28,26 (λόγω χρήσης bitmap heap scan αντί για σειριακή σάρωση). Σε πραγματικό χρόνο η βελτίωση αυτή δεν ήταν ιδιαίτερα μεγάλη, καθώς ο αριθμός μαθημάτων ανά εξάμηνο περιορίζεται έτσι και αλλιώς από τα αρχικά δεδομένα.

3) Το επόμενο σημείο καθυστέρησης εντοπίστηκε στην εύρεση του τρέχοντος εξαμήνου, καθώς είχαμε εισάγει για λόγους δοκιμής σημαντικό πλήθος μελλοντικών εξαμήνων. Όπως αναμενόταν, για την ταχεία αναζήτηση του τωρινού (present) εξαμήνου (που είναι πάντα μοναδικό) η δημιουργία hash index βάσει της αντίστοιχης στήλης είναι η ιδανική (με δέκα φορές μείωση του εκτιμώμενου κόστους). Επίσης, η μείωση σε πραγματικό χρόνο της εύρεσης του τρέχοντος εξαμήνου αναμενόμενα βελτίωσε και το χρόνο εύρεσης των μαθημάτων του εξαμήνου.

4) Τέλος, μικρές ακόμη βελτιώσεις είχαμε με την αντικατάσταση των προαναφερθέντων ευρετηρίων b-tree(serial_number), b-tree(course_code) με τα αντίστοιχα ζεύγη b-tree(serial_number, course_code), b-tree(course_code, semester_id), τα οποία και χρησιμοποιούνται για την αποφυγή των bitmap heap scans με τη χρήση άμεσων ευρετηρίων πολλαπλών χαρακτηριστικών.

5) Μιας και στο επόμενο ερώτημα θα δούμε ότι είναι απαραίτητη η χρήση clustering με βάση το ονοματεπώνυμο (για γρηγορότερη ταξινόμηση), αυτή χρησιμοποιείται και εδώ, χωρίς ωστόσο η διαφορά στην απόδοση να είναι σημαντική. Η τελική μορφή του πλάνου φαίνεται στο Παράρτημα 4.4.

1.3 Βελτίωση απόδοσης ερωτήματος 2.3

Ξεκινώντας την ανάλυση του ερωτήματος 2.3 εντοπίσαμε (Παράρτημα 4.5) ότι η εντολή union επιφέρει μεγάλη καθυστέρηση (περίπου διπλάσιο κόστος), καθώς εκτελείται η εντολή unique για έλεγχο και αφαίρεση διπλοεγγραφών (δηλαδή συνωνυμίες μεταξύ φοιτητών, καθηγητών ή εργαστηριακού προσωπικού). Μιας και η εκφώνηση της Α΄ φάσης δεν μας περιορίζει, προχωρήσαμε στην αντικατάσταση του union με union all, βελτιώνοντας σημαντικά το χρόνο εκτέλεσης του ερωτήματος.

Στη συνέχεια, μιας και απαιτείται η ανάσυρση από τη μνήμη όλων των σειρών των πινάκων "Student", "Professor", "Labstaff", κατασκευάζουμε ευρετήρια b-trees για τα ζητούμενα χαρακτηριστικά (surname, name), τα οποία και χρησιμοποιούμε για τη συσταδοποίηση των πινάκων με βάση τη ζητούμενη αλφαβητική σειρά. Με αυτό τον τρόπο, χρησιμοποιείται merge append αντί για απλό append, αποφεύγοντας έτσι το κόστος της εκ νέου ταξινόμησης των στοιχείων μετά το append (Παράρτημα 4.6).

2 Ζητούμενο 3

Στόχοι αυτού του ζητούμενου είναι τόσο η κατανόηση της χρήσης buffers (cache memory) για τη γρήγορη εκτέλεση ερωτημάτων που επαναλαμβάνονται συχνά, όσο και η κατανόηση του ρόλου των views και materialized views σε μία βάση.

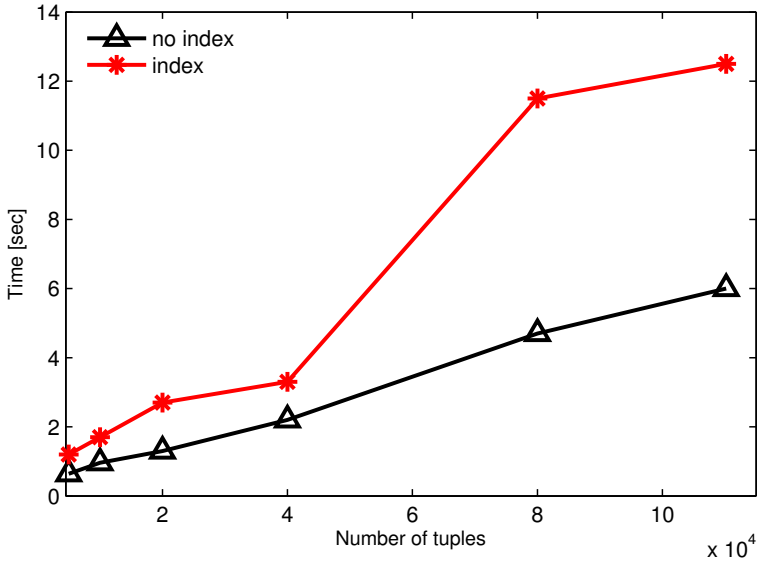
Καταρχάς, παρατηρούμε ότι η view του ερωτήματος λειτουργεί ακριβώς απλά σαν ένα αποθηκευμένο query που μπορεί να καλείται από πολλές συναρτήσεις, δίνοντας ακριβώς το ίδιο πλάνο εκτέλεση (εξαιρώντας διακυμάνσεις λόγω συστήματος και ίδιους χρόνους εκτέλεσης).

Παρατηρούμε επίσης ότι κατά την πρώτη εκτέλεση του ερωτήματος (Παράρτημα 4.7) υπάρχει πολύ μεγαλύτερη καθυστέρηση λόγω ανάγνωσης πληροφορίας και από το δίσκο (1555 σελίδες από τη μνήμη, 390 από το δίσκο). Στη δεύτερη εκτέλεση (Παράρτημα 4.8) όλη η πληροφορία αντλείται από τα buffers πλέον, για αυτό και ο χρόνος εκτέλεσης μειώνεται σημαντικά (το ίδιο ισχύει προφανώς και για το view και materialized view).

Τέλος, όσον αφορά το κόστος εκτέλεσης του materialized view, είναι φανερό (Παράρτημα 4.9) ότι είναι κατά πολύ μικρότερο από αυτό των προηγούμενων δύο περιπτώσεων, καθώς κοστίζει μόνο όσο η ανάγνωση 2 σελίδων από το δίσκο (ή από την cache). Φυσικά αυτό συμβαίνει διότι το αποτέλεσμα του view αποθηκεύεται στο δίσκο κατά τη δημιουργία (ή ανανέωση) του materialized view.

3 Ζητούμενο 4

Τελευταίος στόχος της παρούσας άσκησης αποτελεί η κατανόηση της επίδρασης των λειτουργιών εισαγωγής/διαγραφής εγγραφών από την ύπαρξη ευρετηρίων. Για το σκοπό αυτό χρησιμοποιήσαμε την



Σχήμα 1: Πραγματικός χρόνος εκτέλεσης (μέτρηση pgAdmin) εισαγωγής εγγραφών στον πίνακα "Labstaff" με την ύπαρξη ή όχι του ευρετηρίου b-tree(surname, name).

συνάρτηση εισαγωγής προσώπων του ερωτήματος 1.1 της Α΄ φάσης της άσκησης.

Προτού χρησιμοποιήσουμε τη συνάρτηση αυτή, προχωρήσαμε σε βελτίωση του κώδικά της, καθώς η χρήση for loop σε procedural γλώσσα με εισαγωγή μίας-μίας τυχαίων εγγραφών ήταν μία πολύ αργή διαδικασία. Έτσι για να το αποφύγουμε αυτό το πρόβλημα, προσθέσαμε την υπορουτίνα get_random_fullname, η οποία και δίνει τώρα τη δυνατότητα μαζικής εισαγωγής έως 110220 φοιτητών, καθηγητών ή εργαστηριακών ανά κλήση (δηλαδή όσο το μέγεθος του πίνακα "Surname" που μας δόθηκε).

Με τη νέα αυτή υλοποίηση, προχωρήσαμε σε χρονομέτρηση της εισαγωγής αρκετών χιλιάδων κάθε φορά στον πίνακα "Labstaff", πραγματοποιώντας τις μετρήσεις δύο φορές, μία έχοντας το ευρετήριο b-tree(surname, name)που δημιουργήσαμε στο ερώτημα 2.3, και μία διαγράφοντας το.

Τα αποτελέσματα απεικονίζονται στο Σχήμα 1. Οι μετρήσεις αφορούν την εισαγωγή 5000, 10000, 20000, 40000, 80000, 110220 εγγραφών με απενεργοποιημένο το trigger του πίνακα. Βλέπουμε λοιπόν καθαρά την αρνητική επίδραση του ευρετηρίου στους χρόνους εισαγωγής νέων στοιχείων, κάτι που είναι λογικό γιατί πέρα από την εισαγωγή στον ίδιο τον πίνακα πληρώνουμε και το κόστος εισαγωγής στη δομή του ευρετηρίου.

4 Παράρτημα - Πλάνα Ερωτημάτων

4.1 Ερώτημα 2.1 - Χρήση υπάρχοντος b-tree index

```
"Index Scan using "Student_am_key" on "Student" (cost=0.42..8.44 rows=1 width=197)
(actual time=0.031..0.032 rows=1 loops=1)"
" Index Cond: (am = '2018001035'::bpchar)"
"Planning time: 0.240 ms"
"Execution time: 0.060 ms"
```

4.2 Ερώτημα 2.1 - Προσθήκη hash index

```
"Index Scan using hash_am on "Student" (cost=0.00..8.02 rows=1 width=197) (actual time=0.016..0.017 rows=1 loops
    =1)"
" Index Cond: (am = '2018001035'::bpchar)"
"Planning time: 0.266 ms"
"Execution time: 0.044 ms"
```

4.3 Ερώτημα 2.2 - Αρχικό

```
"Sort (cost=4387.89..4387.90 rows=1 width=108) (actual time=23.321..23.325 rows=123 loops=1)"
" Sort Key: ""Student"".surname, ""Student"".name"
" Sort Method: quicksort Memory: 44kB"
" InitPlan 1 (returns $0)"
```

```
" -> Seq Scan on ""Semester"" (cost=0.00..320.50 rows=8 width=12) (actual time=0.005..1.779 rows=1 loops=1)"
" Filter: (semester_status = 'present'::semester_status_type)"
" Rows Removed by Filter: 14999"
" -> Nested Loop (cost=0.42..4067.38 rows=1 width=108) (actual time=19.871..22.835 rows=123 loops=1)"
" -> Nested Loop (cost=0.00..4059.62 rows=1 width=12) (actual time=19.836..21.808 rows=123 loops=1)"
" Join Filter: (""CourseRun"".serial_number = ""Register"".serial_number)"
" Rows Removed by Join Filter: 709"
" -> Seq Scan on ""CourseRun"" (cost=0.00..23.80 rows=1 width=15) (actual time=1.886..1.892 rows=1 loops=1)"
" Filter: ((course_code = 'ΠΛΗ 102'::bpchar) AND (semester_id = $0))"
" Rows Removed by Filter: 919"
" -> Seq Scan on ""Register"" (cost=0.00..4033.19 rows=210 width=27) (actual time=0.349..19.853 rows=832 loops=1)
    "
" Filter: ((register_status <> 'rejected'::register_status_type) AND (course_code = 'ΠΛΗ 102'::bpchar))"
" Rows Removed by Filter: 176181"
" -> Index Scan using ""Student_pkey"" on ""Student"" (cost=0.42..7.75 rows=1 width=120) (actual time
    =0.007..0.008 rows=1 loops=123)"
" Index Cond: (amka = ""Register"".amka)"
"Planning time: 1.297 ms"
"Execution time: 23.395 ms"
```

4.4 Ερώτημα 2.2 - Τελικό

```
"Sort (cost=47.63..47.64 rows=1 width=108) (actual time=1.278..1.280 rows=123 loops=1)"
" Sort Key: ""Student"".surname, ""Student"".name"
" Sort Method: quicksort Memory: 44kB"
" InitPlan 1 (returns $0)"
" -> Bitmap Heap Scan on ""Semester"" (cost=4.06..30.28 rows=8 width=12) (actual time=0.013..0.013 rows=1 loops
    =1)"
" Recheck Cond: (semester_status = 'present'::semester_status_type)"
" Heap Blocks: exact=1"
" -> Bitmap Index Scan on semester_status_ind (cost=0.00..4.06 rows=8 width=0) (actual time=0.007..0.007 rows=1
    loops=1)"
" Index Cond: (semester_status = 'present'::semester_status_type)"
" -> Merge Join (cost=17.18..17.34 rows=1 width=108) (actual time=0.155..0.868 rows=123 loops=1)"
" Merge Cond: (""Student"".amka = ""Register"".amka)"
" -> Index Scan using ""Student_pkey"" on ""Student"" (cost=0.42..22072.07 rows=150400 width=120) (actual time
    =0.005..0.400 rows=401 loops=1)"
" -> Sort (cost=16.76..16.76 rows=1 width=12) (actual time=0.135..0.144 rows=123 loops=1)"
" Sort Key: ""Register"".amka"
" Sort Method: quicksort Memory: 30kB"
" -> Nested Loop (cost=0.70..16.75 rows=1 width=12) (actual time=0.044..0.083 rows=123 loops=1)"
" -> Index Scan using courserun_code_ind on ""CourseRun"" (cost=0.28..8.29 rows=1 width=15) (actual time
    =0.030..0.030 rows=1 loops=1)"
" Index Cond: ((course_code = 'ΠΛΗ 102'::bpchar) AND (semester_id = $0))"
" -> Index Scan using register_serial_ind on ""Register"" (cost=0.42..8.44 rows=1 width=27) (actual time
    =0.012..0.039 rows=123 loops=1)"
" Index Cond: ((serial_number = ""CourseRun"".serial_number) AND (course_code = 'ΠΛΗ 102'::bpchar))"
" Filter: (register_status <> 'rejected'::register_status_type)"
"Planning time: 1.383 ms"
"Execution time: 1.347 ms"
```

4.5 Ερώτημα 2.3 - Αρχικό

```
"Sort (cost=161823.16..162949.41 rows=450500 width=96) (actual time=13919.026..13998.182 rows=450281 loops=1)"
" Sort Key: ""Student"".surname, ""Student"".name"
" Sort Method: external sort Disk: 51648kB"
" -> Unique (cost=87408.58..91913.58 rows=450500 width=97) (actual time=5330.048..7293.073 rows=450281 loops=1)"
" -> Sort (cost=87408.58..88534.83 rows=450500 width=97) (actual time=5330.046..7134.622 rows=450500 loops=1)"
" Sort Key: ""Student"".surname, ""Student"".name, ('Student'::text)"
" Sort Method: external merge Disk: 51584kB"
" -> Append (cost=0.00..20464.00 rows=450500 width=97) (actual time=0.012..188.917 rows=450500 loops=1)"
" -> Seq Scan on ""Student"" (cost=0.00..5892.00 rows=150400 width=97) (actual time=0.010..65.222 rows=150400
    loops=1)"
" -> Seq Scan on ""Professor"" (cost=0.00..5073.20 rows=150020 width=97) (actual time=0.003..50.264 rows=150020
    loops=1)"
" -> Seq Scan on ""Labstaff"" (cost=0.00..4993.80 rows=150080 width=97) (actual time=0.002..49.633 rows=150080
    loops=1)"
"Planning time: 0.586 ms"
"Execution time: 14022.034 ms"
```

4.6 Ερώτημα 2.3 - Τελικό

```
"Merge Append (cost=1.28..43893.29 rows=450500 width=129) (actual time=0.045..678.549 rows=450500 loops=1)"
" Sort Key: ""Student"".surname, ""Student"".name"
" -> Index Only Scan using student_surname_ind on ""Student"" (cost=0.42..11896.42 rows=150400 width=129) (actual time=0.018..36.616 rows=150400 loops=1)"
" Heap Fetches: 0"
" -> Index Only Scan using professor_surname_ind on ""Professor"" (cost=0.42..11862.72 rows=150020 width=129) (actual time=0.007..36.261 rows=150020 loops=1)"
" Heap Fetches: 0"
" -> Index Only Scan using labstaff_surname_ind on ""Labstaff"" (cost=0.42..11867.62 rows=150080 width=129) (actual time=0.006..37.540 rows=150080 loops=1)"
" Heap Fetches: 0"
"Planning time: 0.438 ms"
"Execution time: 695.493 ms"
```

4.7 Ερώτημα 3 - Διάβασμα από δίσκο

```
"Nested Loop (cost=3817.98..6919.66 rows=57 width=213) (actual time=38.038..38.335 rows=34 loops=1)"
" Buffers: shared hit=1555 read=390"
" -> HashAggregate (cost=3817.56..3821.86 rows=430 width=14) (actual time=38.010..38.020 rows=34 loops=1)"
" Group Key: ""Student_1"".amka"
" Buffers: shared hit=1418 read=390"
" -> Hash Join (cost=3786.85..3814.34 rows=430 width=14) (actual time=32.793..37.897 rows=265 loops=1)"
" Hash Cond: (""Register"".course_code = ""Course"".course_code)"
" Buffers: shared hit=1418 read=390"
" -> Merge Join (cost=3781.26..3802.84 rows=430 width=23) (actual time=32.737..37.753 rows=265 loops=1)"
" Merge Cond: (""Student_1"".amka = ""Register"".amka)"
" Buffers: shared hit=1415 read=390"
" -> Index Scan using ""Student_pkey"" on ""Student"" ""Student_1"" (cost=0.42..22448.35 rows=19848 width=12) (actual time=0.081..3.794 rows=37 loops=1)"
" Filter: (surname ~~ '%ΑΛ%'::text)"
" Rows Removed by Filter: 386"
" Buffers: shared hit=37 read=390"
" -> Sort (cost=3780.70..3788.84 rows=3257 width=23) (actual time=32.621..32.716 rows=3147 loops=1)"
" Sort Key: ""Register"".amka"
" Sort Method: quicksort Memory: 342kB"
" Buffers: shared hit=1378"
" -> Seq Scan on ""Register"" (cost=0.00..3590.66 rows=3257 width=23) (actual time=15.793..24.469 rows=3147 loops=1)"
" Filter: (register_status = 'approved'::register_status_type)"
" Rows Removed by Filter: 173866"
" Buffers: shared hit=1378"
" -> Hash (cost=4.15..4.15 rows=115 width=13) (actual time=0.043..0.043 rows=115 loops=1)"
" Buckets: 1024 Batches: 1 Memory Usage: 14kB"
" Buffers: shared hit=3"
" -> Seq Scan on ""Course"" (cost=0.00..4.15 rows=115 width=13) (actual time=0.004..0.022 rows=115 loops=1)"
" Buffers: shared hit=3"
" -> Index Scan using ""Student_pkey"" on ""Student"" (cost=0.42..7.18 rows=1 width=197) (actual time=0.008..0.008 rows=1 loops=34)"
" Index Cond: (amka = ""Student_1"".amka)"
" Filter: (surname ~~ '%ΑΛ%'::text)"
" Buffers: shared hit=137"
"Planning time: 7.761 ms"
"Execution time: 38.545 ms"
```

4.8 Ερώτημα 3 - Διάβασμα από cache

```
"Nested Loop (cost=3817.98..6919.66 rows=57 width=213) (actual time=27.143..27.425 rows=34 loops=1)"
" Buffers: shared hit=1945"
" -> HashAggregate (cost=3817.56..3821.86 rows=430 width=14) (actual time=27.121..27.129 rows=34 loops=1)"
" Group Key: ""Student_1"".amka"
" Buffers: shared hit=1808"
" -> Hash Join (cost=3786.85..3814.34 rows=430 width=14) (actual time=25.436..27.032 rows=265 loops=1)"
" Hash Cond: (""Register"".course_code = ""Course"".course_code)"
" Buffers: shared hit=1808"
" -> Merge Join (cost=3781.26..3802.84 rows=430 width=23) (actual time=25.373..26.892 rows=265 loops=1)"
" Merge Cond: (""Student_1"".amka = ""Register"".amka)"
" Buffers: shared hit=1805"
" -> Index Scan using ""Student_pkey"" on ""Student"" ""Student_1"" (cost=0.42..22448.35 rows=19848 width=12) (actual time=0.016..0.388 rows=37 loops=1)"
" Filter: (surname ~~ '%ΑΛ%'::text)"
" Rows Removed by Filter: 386"
" Buffers: shared hit=427"
" -> Sort (cost=3780.70..3788.84 rows=3257 width=23) (actual time=25.322..25.420 rows=3147 loops=1)"
```

```
" Sort Key: ""Register"".amka"
" Sort Method: quicksort Memory: 342kB"
" Buffers: shared hit=1378"
" -> Seq Scan on ""Register"" (cost=0.00..3590.66 rows=3257 width=23) (actual time=15.509..17.410 rows=3147 loops=1)"
" Filter: (register_status = 'approved'::register_status_type)"
" Rows Removed by Filter: 173866"
" Buffers: shared hit=1378"
" -> Hash (cost=4.15..4.15 rows=115 width=13) (actual time=0.046..0.046 rows=115 loops=1)"
" Buckets: 1024 Batches: 1 Memory Usage: 14kB"
" Buffers: shared hit=3"
" -> Seq Scan on ""Course"" (cost=0.00..4.15 rows=115 width=13) (actual time=0.005..0.023 rows=115 loops=1)"
" Buffers: shared hit=3"
" -> Index Scan using ""Student_pkey"" on ""Student"" (cost=0.42..7.18 rows=1 width=197) (actual time=0.008..0.008 rows=1 loops=34)"
" Index Cond: (amka = ""Student_1"".amka)"
" Filter: (surname ~~ '%ΑΛ%'::text)"
" Buffers: shared hit=137"
"Planning time: 1.221 ms"
"Execution time: 27.565 ms"
```

4.9 Ερώτημα 3 - Materialized View

```
"Seq Scan on ""StudentsWithAL_Q3_mater"" (cost=0.00..11.20 rows=120 width=636) (actual time=0.011..0.014 rows=34 loops=1)"
" Buffers: shared hit=2"
"Planning time: 0.062 ms"
"Execution time: 0.046 ms"
```