Αναφορά Πρότζεκτ

Για τον καλύτερο προσδιορισμό των αποτελεσμάτων έγινε εισαγωγή εκατό χιλιάδων διαφορετικών πλειάδων σε κάθε πίνακα (Student, Professor, Labstaff).

Ερώτημα 2:

Σε όλα τα παρακάτω ερωτήματα όπως και ζητείται γίνεται ταξινόμηση με βάση το επώνυμο και το όνομα.

2.1)

Στο ερώτημα αυτό γίνεται ανάκτηση των στοιχείων ενός φοιτητή με δοσμένο αριθμό μητρώου. Παρακάτω βρίσκεται το αντίστοιχο query:

```
SELECT
 amka,
NAME,
surname,
father_name,
email
FROM "Student"
WHERE am = '2009090013'
ORDER BY surname, name;
```

Η επιλογή του Index με βάση το attribute που βρίσκεται στο selection στην περίπτωσή μας ο αριθμός μητρώου. Από τη θεωρία γνωρίζουμε ότι το κατάλληλο index για ένα τέτοιο point query είναι το hash παρόλαυτά, η επιλογή του έγινε σκόπιμα btree και όχι hash Καθώς η ίδια η postgres δεν υπάρχει clustering σε hash indexes. Στις παρακάτω εικόνες παρουσιάζονται τα αποτελέσματα της explain-analyse με τους αντίστοιχους χρόνους εκτελέσης με ή χωρίς index.

Χωρίς Index.

```
QUERY PLAN
Sort  (cost=3833.39..3833.39 rows=1 width=86) (actual time=21.290..21.291 rows=1 loops=1)
  Sort Key: surname, name
  Sort Method: quicksort  Memory: 25kB
  ->  Seq Scan on "Student"  (cost=0.00..3833.38 rows=1 width=86) (actual time=10.255..21.278 rows=1 loops=1)
        Filter: (am = '2007056816'::bpchar)
        Rows Removed by Filter: 131069
Planning time: 0.133 ms
Execution time: 21.314 ms
```

Με BTree Index πάνω στο attribute am.

```
QUERY PLAN
Sort  (cost=8.45..8.45 rows=1 width=86) (actual time=0.054..0.054 rows=1 loops=1)
  Sort Key: surname, name
  Sort Method: quicksort  Memory: 25kB
  ->  Index Scan using student_am_index1 on "Student"  (cost=0.42..8.44 rows=1 width=86) (actual time=0.043..0.044 rows=1 loops=1)
        Index Cond: (am = '2007056816'::bpchar)
Planning time: 0.227 ms
Execution time: 0.083 ms
```

Η δραματική πτώση στο execution time οφείλεται στην κατάλληλη δεικτοδότηση των απαιτούμενων tuples για το query μέσω του Index στο δίσκο. Για την περαιτέρω πτώση του execution time χρησιμοποιήσαμε clustering με σκοπό την ταξινόμηση των δεδομένων του δίσκου με βάση το btree index που μόλις χρησιμοποιήσαμε

Clustering on btree index

```
QUERY PLAN
Sort  (cost=8.45..8.45 rows=1 width=86) (actual time=0.041..0.041 rows=1 loops=1)
  Sort Key: surname, name
  Sort Method: quicksort  Memory: 25kB
  ->  Index Scan using student_am_index1 on "Student"  (cost=0.42..8.44 rows=1 width=86) (actual time=0.030..0.031 rows=1 loops=1)
        Index Cond: (am = '2007056816'::bpchar)
Planning time: 0.228 ms
Execution time: 0.066 ms
```

2.2)

Στο query αυτό γίνεται ανάκτηση δεδομένων ενός φοιτητή που παρακολουθεί ένα συγκεκριμένο μάθημα στο τωρινό εξάμηνο. Παρακάτω βρίσκεται το query αυτό:

```
SELECT  name,  surname,  am
FROM courserun
 NATURAL JOIN semester
 NATURAL JOIN register
 NATURAL JOIN "Student"
WHERE course_code = 'ΠΛΗ 302' AND semester_status =
                        'present' :: SEMESTER_STATUS_TYPE
ORDER BY surname, NAME;
```

Στο συγκεκριμένο ερώτημα λόγω των διαφόρων natural joins ήταν αναγκαία η δημιουργία μόνο ενός index και αυτό πάνω στο course_code του πίνακα register. Η δημιουργία ενός Index πάνω στο semester status θα άστοχη λόγω του γεγονότος ότι οι δυνατές τιμές είναι μόνο τρεις πράγμα που ο optimizer αγνοεί.

Εκτέλεση χωρίς Index

```
QUERY PLAN
Sort  (cost=873.55..873.61 rows=21 width=44) (actual time=11.397..11.400 rows=50 loops=1)
  Sort Key: "Student".surname, "Student".name
  Sort Method: quicksort  Memory: 29kB
  ->  Nested Loop  (cost=4.65..873.09 rows=21 width=44) (actual time=6.113..11.273 rows=50 loops=1)
        ->  Nested Loop  (cost=4.36..743.42 rows=21 width=4) (actual time=6.095..11.098 rows=50 loops=1)
              Join Filter: (courserun.serial_number = register.serial_number)
              Rows Removed by Join Filter: 375
              ->  Nested Loop  (cost=4.36..17.12 rows=1 width=15) (actual time=0.059..0.068 rows=1 loops=1)
                    Join Filter: (courserun.semester_id = semester.semester_id)
                    Rows Removed by Join Filter: 9
                    ->  Seq Scan on semester  (cost=0.00..1.26 rows=1 width=4) (actual time=0.017..0.019 rows=1 loops=1)
                          Filter: (semester_status = 'present'::semester_status_type)
                          Rows Removed by Filter: 20
                    ->  Bitmap Heap Scan on courserun  (cost=4.36..15.73 rows=10 width=19) (actual time=0.036..0.039 rows=10 loops=1)
                          Recheck Cond: (course_code = 'ΠΛΗ 302'::bpchar)
                          Heap Blocks: exact=1
                          ->  Bitmap Index Scan on courserun_course_code_serial_number_pk  (cost=0.00..4.35 rows=10 width=0) (actual time=0.029..0.029 rows=10 loops=1)
                                Index Cond: (course_code = 'ΠΛΗ 302'::bpchar)
              ->  Seq Scan on register  (cost=0.00..721.12 rows=414 width=19) (actual time=6.013..10.969 rows=425 loops=1)
                    Filter: (course_code = 'ΠΛΗ 302'::bpchar)
                    Rows Removed by Filter: 35825
        ->  Index Scan using student_pkey on "Student"  (cost=0.29..6.17 rows=1 width=48) (actual time=0.003..0.003 rows=1 loops=50)
              Index Cond: (amka = register.amka)
Planning time: 0.877 ms
Execution time: 11.524 ms
```

Εκτέλεση με Btree Index στο register.course_code

```
QUERY PLAN
Sort  (cost=925.21..925.26 rows=21 width=44) (actual time=0.603..0.609 rows=50 loops=1)
  Sort Key: "Student".surname, "Student".name
  Sort Method: quicksort  Memory: 29kB
  ->  Nested Loop  (cost=16.15..924.75 rows=21 width=44) (actual time=0.316..0.496 rows=50 loops=1)
        ->  Nested Loop  (cost=15.85..795.07 rows=21 width=4) (actual time=0.307..0.336 rows=50 loops=1)
              Join Filter: (courserun.serial_number = register.serial_number)
              Rows Removed by Join Filter: 375
              ->  Nested Loop  (cost=4.36..37.51 rows=1 width=15) (actual time=0.062..0.064 rows=1 loops=1)
                    Join Filter: (courserun.semester_id = semester.semester_id)
                    Rows Removed by Join Filter: 9
                    ->  Seq Scan on semester  (cost=0.00..1.26 rows=1 width=4) (actual time=0.010..0.012 rows=1 loops=1)
                          Filter: (semester_status = 'present'::semester_status_type)
                          Rows Removed by Filter: 20
                    ->  Bitmap Heap Scan on courserun  (cost=4.36..36.13 rows=10 width=19) (actual time=0.027..0.045 rows=10 loops=1)
                          Recheck Cond: (course_code = 'ΠΛΗ 302'::bpchar)
                          Heap Blocks: exact=10
                          ->  Bitmap Index Scan on courserun_course_code_serial_number_pk  (cost=0.00..4.35 rows=10 width=0) (actual time=0.022..0.022 rows=10 loops=1)
                                Index Cond: (course_code = 'ΠΛΗ 302'::bpchar)
              ->  Bitmap Heap Scan on register  (cost=11.50..752.38 rows=414 width=19) (actual time=0.075..0.175 rows=425 loops=1)
                    Recheck Cond: (course_code = 'ΠΛΗ 302'::bpchar)
                    Heap Blocks: exact=21
                    ->  Bitmap Index Scan on course_code_idx  (cost=0.00..11.39 rows=414 width=0) (actual time=0.068..0.068 rows=425 loops=1)
                          Index Cond: (course_code = 'ΠΛΗ 302'::bpchar)
        ->  Index Scan using student_pkey on "Student"  (cost=0.29..6.17 rows=1 width=48) (actual time=0.002..0.002 rows=1 loops=50)
              Index Cond: (amka = register.amka)
Planning time: 0.586 ms
Execution time: 0.684 ms
```

Εκτέλεση με Clustering

```
QUERY PLAN
Sort  (cost=445.74..445.79 rows=21 width=44) (actual time=0.317..0.320 rows=50 loops=1)
  Sort Key: "Student".surname, "Student".name
  Sort Method: quicksort  Memory: 29kB
  ->  Nested Loop  (cost=28.92..445.28 rows=21 width=44) (actual time=0.100..0.260 rows=50 loops=1)
        ->  Hash Join  (cost=28.63..315.60 rows=21 width=4) (actual time=0.096..0.178 rows=50 loops=1)
              Hash Cond: (register.serial_number = courserun.serial_number)
              ->  Bitmap Heap Scan on register  (cost=11.50..296.71 rows=414 width=19) (actual time=0.057..0.095 rows=425 loops=1)
                    Recheck Cond: (course_code = 'ΠΛΗ 302'::bpchar)
                    Heap Blocks: exact=5
                    ->  Bitmap Index Scan on course_code_idx  (cost=0.00..11.39 rows=414 width=0) (actual time=0.052..0.052 rows=425 loops=1)
                          Index Cond: (course_code = 'ΠΛΗ 302'::bpchar)
              ->  Hash  (cost=17.12..17.12 rows=1 width=15) (actual time=0.022..0.022 rows=1 loops=1)
                    Buckets: 1024  Batches: 1  Memory Usage: 9kB
                    ->  Nested Loop  (cost=4.36..17.12 rows=1 width=15) (actual time=0.018..0.021 rows=1 loops=1)
                          Join Filter: (courserun.semester_id = semester.semester_id)
                          Rows Removed by Join Filter: 9
                          ->  Seq Scan on semester  (cost=0.00..1.26 rows=1 width=4) (actual time=0.007..0.008 rows=1 loops=1)
                                Filter: (semester_status = 'present'::semester_status_type)
                                Rows Removed by Filter: 20
                          ->  Bitmap Heap Scan on courserun  (cost=4.36..15.73 rows=10 width=19) (actual time=0.008..0.009 rows=10 loops=1)
                                Recheck Cond: (course_code = 'ΠΛΗ 302'::bpchar)
                                Heap Blocks: exact=1
                                ->  Bitmap Index Scan on courserun_course_code_serial_number_pk  (cost=0.00..4.35 rows=10 width=0) (actual time=
                                      Index Cond: (course_code = 'ΠΛΗ 302'::bpchar)
        ->  Index Scan using student_pkey on "Student"  (cost=0.29..6.17 rows=1 width=48) (actual time=0.001..0.001 rows=1 loops=50)
              Index Cond: (amka = register.amka)
Planning time: 0.332 ms
Execution time: 0.361 ms
```

Μπορούμε να παρατηρήσουμε ότι κάνοντας χρήση clustering οι διαφορές είναι υποδιπλάσιας τάξης πράγμα που οφείλεται στην ταξινόμηση των tuples του register στον δίσκο. Μετά από πολλαπλές δοκιμές ενεργοποιώντας και απενεργοποιώντας μεθόδους join (hash join, merge join, nested loop join) παρατηρήσαμε ότι οι επιλογές του optimizer είναι οι βέλτιστες. Ακολουθούν οι παραδοχές που χρησιμοποιούνται οι μεθόδοι merge join και hash join.

Merge Join

```
QUERY PLAN
Sort  (cost=5031.55..5031.61 rows=21 width=44) (actual time=55.486..55.491 rows=50 loops=1)
  Sort Key: "Student".surname, "Student".name
  Sort Method: quicksort  Memory: 29kB
  ->  Merge Join  (cost=4991.26..5031.09 rows=21 width=44) (actual time=55.356..55.394 rows=50 loops=1)
        Merge Cond: ("Student".amka = register.amka)
        ->  Index Scan using student_pkey on "Student"  (cost=0.29..4449.79 rows=100275 width=48) (actual time=0.135..46.824 rows=100226 loops=1)
        ->  Sort  (cost=334.70..334.75 rows=21 width=4) (actual time=0.469..0.475 rows=50 loops=1)
              Sort Key: register.amka
              Sort Method: quicksort  Memory: 27kB
              ->  Merge Join  (cost=331.95..334.24 rows=21 width=4) (actual time=0.440..0.452 rows=50 loops=1)
                    Merge Cond: (courserun.serial_number = register.serial_number)
                    ->  Sort  (cost=17.24..17.25 rows=1 width=15) (actual time=0.081..0.082 rows=1 loops=1)
                          Sort Key: courserun.serial_number
                          Sort Method: quicksort  Memory: 25kB
                          ->  Merge Join  (cost=17.17..17.23 rows=1 width=15) (actual time=0.075..0.076 rows=1 loops=1)
                                Merge Cond: (courserun.semester_id = semester.semester_id)
                                ->  Sort  (cost=15.89..15.92 rows=10 width=19) (actual time=0.044..0.046 rows=10 loops=1)
                                      Sort Key: courserun.semester_id
                                      Sort Method: quicksort  Memory: 25kB
                                      ->  Bitmap Heap Scan on courserun  (cost=4.36..15.73 rows=10 width=19) (actual time=0.027..0.029 rows=10 loo…
                                            Recheck Cond: (course_code = 'ΠΛΗ 302'::bpchar)
                                            Heap Blocks: exact=1
                                            ->  Bitmap Index Scan on courserun_course_code_serial_number_pk  (cost=0.00..4.35 rows=10 width=0) (ac…
                                                  Index Cond: (course_code = 'ΠΛΗ 302'::bpchar)
                                ->  Sort  (cost=1.27..1.28 rows=1 width=4) (actual time=0.028..0.028 rows=1 loops=1)
                                      Sort Key: semester.semester_id
                                      Sort Method: quicksort  Memory: 25kB
                                      ->  Seq Scan on semester  (cost=0.00..1.26 rows=1 width=4) (actual time=0.011..0.012 rows=1 loops=1)
                                            Filter: (semester_status = 'present'::semester_status_type)
                                            Rows Removed by Filter: 20
                    ->  Sort  (cost=314.71..315.74 rows=414 width=19) (actual time=0.302..0.334 rows=425 loops=1)
                          Sort Key: register.serial_number
                          Sort Method: quicksort  Memory: 58kB
                          ->  Bitmap Heap Scan on register  (cost=11.50..296.71 rows=414 width=19) (actual time=0.081..0.199 rows=425 loops=1)
                                Recheck Cond: (course_code = 'ΠΛΗ 302'::bpchar)
                                Heap Blocks: exact=5
                                ->  Bitmap Index Scan on course_code_idx  (cost=0.00..11.39 rows=414 width=0) (actual time=0.075..0.075 rows=425 l…
                                      Index Cond: (course_code = 'ΠΛΗ 302'::bpchar)
Planning time: 0.567 ms
Execution time: 55.595 ms
```

Hash Join

```
QUERY PLAN
Sort  (cost=4506.06..4506.11 rows=21 width=44) (actual time=22.823..22.828 rows=50 loops=1)
  Sort Key: "Student".surname, "Student".name
  Sort Method: quicksort  Memory: 29kB
  ->  Hash Join  (cost=319.51..4505.60 rows=21 width=44) (actual time=22.694..22.736 rows=50 loops=1)
        Hash Cond: ("Student".amka = register.amka)
        ->  Seq Scan on "Student"  (cost=0.00..2681.75 rows=100275 width=48) (actual time=0.006..12.397 rows=100275 loops=1)
        ->  Hash  (cost=319.25..319.25 rows=21 width=4) (actual time=0.297..0.297 rows=50 loops=1)
              Buckets: 1024  Batches: 1  Memory Usage: 10kB
              ->  Hash Join  (cost=32.27..319.25 rows=21 width=4) (actual time=0.124..0.284 rows=50 loops=1)
                    Hash Cond: (register.serial_number = courserun.serial_number)
                    ->  Bitmap Heap Scan on register  (cost=15.21..300.42 rows=414 width=19) (actual time=0.064..0.191 rows=425 loops=1)
                          Recheck Cond: (course_code = 'ΠΛΗ 302'::bpchar)
                          Heap Blocks: exact=5
                          ->  Bitmap Index Scan on course_code_idx  (cost=0.00..15.11 rows=414 width=0) (actual time=0.056..0.056 rows=425 loops=1)
                                Index Cond: (course_code = 'ΠΛΗ 302'::bpchar)
                    ->  Hash  (cost=17.05..17.05 rows=1 width=15) (actual time=0.035..0.035 rows=1 loops=1)
                          Buckets: 1024  Batches: 1  Memory Usage: 9kB
                          ->  Hash Join  (cost=5.63..17.05 rows=1 width=15) (actual time=0.031..0.034 rows=1 loops=1)
                                Hash Cond: (courserun.semester_id = semester.semester_id)
                                ->  Bitmap Heap Scan on courserun  (cost=4.36..15.73 rows=10 width=19) (actual time=0.016..0.018 rows=10 loops=1)
                                      Recheck Cond: (course_code = 'ΠΛΗ 302'::bpchar)
                                      Heap Blocks: exact=1
                                      ->  Bitmap Index Scan on courserun_course_code_serial_number_pk  (cost=0.00..4.35 rows=10 width=0) (actual time=
                                            Index Cond: (course_code = 'ΠΛΗ 302'::bpchar)
                                ->  Hash  (cost=1.26..1.26 rows=1 width=4) (actual time=0.010..0.010 rows=1 loops=1)
                                      Buckets: 1024  Batches: 1  Memory Usage: 9kB
                                      ->  Seq Scan on semester  (cost=0.00..1.26 rows=1 width=4) (actual time=0.007..0.008 rows=1 loops=1)
                                            Filter: (semester_status = 'present'::semester_status_type)
                                            Rows Removed by Filter: 20
Planning time: 0.519 ms
Execution time: 22.896 ms
```

2.3)

Το συγκεκριμένο ερώτημα κάνει την ανάκτηση του ονοματεπωνύμου και του χαρακτηρισμού ενός ατόμου της βάσης δεδομένων. Ακολουθεί το σχετικό query:

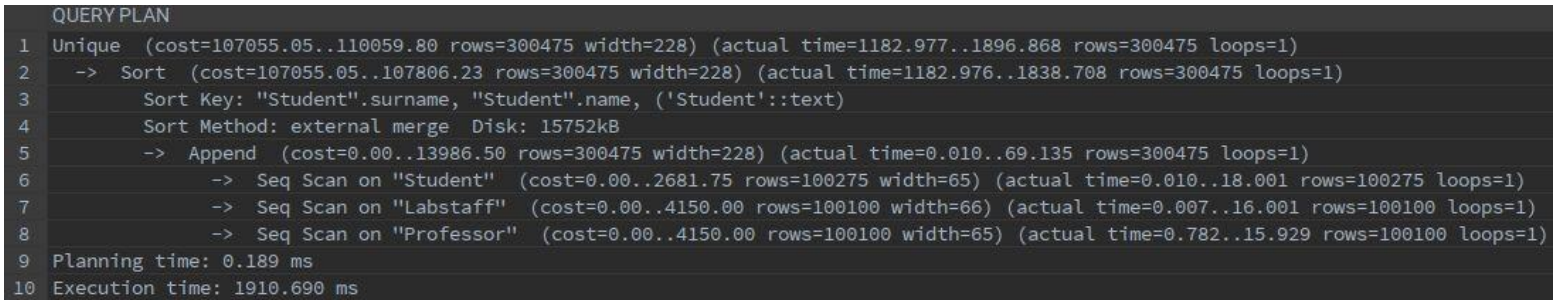
```
SELECT *
FROM (SELECT
      "Student".surname,
      "Student".name,
      'Student' AS status
    FROM "Student"
    UNION
    SELECT
      "Labstaff".surname,
      "Labstaff".name,
      'Labstaff' AS status
    FROM "Labstaff"
    UNION
    SELECT
      "Professor".surname,
      "Professor".name,
      'Professor' AS status
    FROM "Professor") x
ORDER BY surname, name
```

Θέτοντας btree indexes στα surname και name attributes παρατηρήσαμε ότι εν τέλει δεν γίνεται η χρήση τους, πράγμα που είναι λογικό διότι το συγκεκριμένο ερώτημα κάνει ανάκτηση όλων των δεδομένων των πινάκων Student, Professor και Labstaff. Έτσι σε οποιαδήποτε περίπτωση(είτε με χρήση των indexes είτε χωρίς) το execution time παραμένει σταθερό.

Χωρίς Indexes

```
QUERY PLAN
Unique  (cost=107055.05..110059.80 rows=300475 width=228) (actual time=1190.086..1903.317 rows=300475 loops=1)
  ->  Sort  (cost=107055.05..107806.23 rows=300475 width=228) (actual time=1190.084..1845.873 rows=300475 loops=1)
        Sort Key: "Student".surname, "Student".name, ('Student'::text)
        Sort Method: external merge  Disk: 15752kB
        ->  Append  (cost=0.00..13986.50 rows=300475 width=228) (actual time=0.011..73.291 rows=300475 loops=1)
              ->  Seq Scan on "Student"  (cost=0.00..2681.75 rows=100275 width=65) (actual time=0.010..20.327 rows=100275 loops=1)
              ->  Seq Scan on "Labstaff"  (cost=0.00..4150.00 rows=100100 width=66) (actual time=0.008..16.835 rows=100100 loops=1)
              ->  Seq Scan on "Professor"  (cost=0.00..4150.00 rows=100100 width=65) (actual time=0.886..17.208 rows=100100 loops=1)
Planning time: 0.129 ms
Execution time: 1917.216 ms
```

Με Index



```
QUERY PLAN
Unique  (cost=107055.05..110059.80 rows=300475 width=228) (actual time=1182.977..1896.868 rows=300475 loops=1)
  ->  Sort  (cost=107055.05..107806.23 rows=300475 width=228) (actual time=1182.976..1838.708 rows=300475 loops=1)
        Sort Key: "Student".surname, "Student".name, ('Student'::text)
        Sort Method: external merge  Disk: 15752kB
        ->  Append  (cost=0.00..13986.50 rows=300475 width=228) (actual time=0.010..69.135 rows=300475 loops=1)
              ->  Seq Scan on "Student"  (cost=0.00..2681.75 rows=100275 width=65) (actual time=0.010..18.001 rows=100275 loops=1)
              ->  Seq Scan on "Labstaff"  (cost=0.00..4150.00 rows=100100 width=66) (actual time=0.007..16.001 rows=100100 loops=1)
              ->  Seq Scan on "Professor"  (cost=0.00..4150.00 rows=100100 width=65) (actual time=0.782..15.929 rows=100100 loops=1)
Planning time: 0.189 ms
Execution time: 1910.690 ms
```

Ερώτημα 3:

Στο συγκεκριμένο ερώτημα ζητείται η δημιουργία και η σύγκριση των χρόνων εκτέλεσης του ίδιου query σε virtual view και σε materialized view. Η ζητούμενη SQL ερώτηση είναι η εξής:

Εύρεση και παρουσίαση των στοιχείων των φοιτητών που το επίθετό τους περιέχει την συμβολοσειρά “αλ”, τον αριθμό των μαθημάτων για τα οποία έχει εγκριθεί η εγγραφή τους και τον συνολικό αριθμό διδακτικών μονάδων.

Παρακάτω παρουσιάζεται το σχετικό query: SELECT

```
 s.amka,
 s.surname,
 s.father_name,
 s.name,
 s.entry_date,
 s.email,
 s.am,
count(r.course_code),
 CASE WHEN sum(r.units) IS NULL
   THEN 0
 ELSE sum(r.units) END
FROM "Student" s
 LEFT OUTER JOIN ((SELECT *
                   FROM register
                   WHERE register_status =
                       'approved' :: REGISTER_STATUS_TYPE) r1
   NATURAL JOIN "Course") r ON S.amka = r.amka
WHERE surname LIKE '%ΑΛ%'
GROUP BY S.amka;
```

Παρακάτω παρουσιάζονται οι εκτελέσεις των δύο όψεων:

Virtual View

```
QUERY PLAN
HashAggregate  (cost=4066.16..4228.23 rows=16206 width=115) (actual time=48.700..51.473 rows=10741 loops=1)
  Group Key: s.amka
  ->  Hash Right Join  (cost=3156.60..3904.11 rows=16206 width=112) (actual time=34.392..42.890 rows=10937 loops=1)
        Hash Cond: (register.amka = s.amka)
        ->  Hash Join  (cost=21.59..761.69 rows=1380 width=17) (actual time=0.209..4.966 rows=1400 loops=1)
              Hash Cond: (register.course_code = "Course".course_code)
              ->  Seq Scan on register  (cost=0.00..721.12 rows=1380 width=15) (actual time=0.065..4.410 rows=1400 loops=1)
                    Filter: (register_status = 'approved'::register_status_type)
                    Rows Removed by Filter: 34850
              ->  Hash  (cost=20.15..20.15 rows=115 width=13) (actual time=0.117..0.117 rows=115 loops=1)
                    Buckets: 1024  Batches: 1  Memory Usage: 14kB
                    ->  Seq Scan on "Course"  (cost=0.00..20.15 rows=115 width=13) (actual time=0.010..0.088 rows=115 loops=1)
        ->  Hash  (cost=2932.44..2932.44 rows=16206 width=99) (actual time=34.151..34.151 rows=10741 loops=1)
              Buckets: 16384  Batches: 1  Memory Usage: 1579kB
              ->  Seq Scan on "Student" s  (cost=0.00..2932.44 rows=16206 width=99) (actual time=0.025..30.855 rows=10741 loops=1)
                    Filter: ((surname)::text ~~ '%ΑΛ%'::text)
                    Rows Removed by Filter: 89534
Planning time: 0.685 ms
Execution time: 52.085 ms
```

Materialized View

```
QUERY PLAN
Seq Scan on student_details_materialized_view  (cost=0.00..312.41 rows=10741 width=117) (actual time=0.007..1.043 rows=10741 loops=1)
Planning time: 0.030 ms
Execution time: 1.434 ms
```

Από τα παραπάνω μπορούμε να παρατηρήσουμε ότι ο χρόνος εκτέλεσης των δύο όψεων είναι δραματικά καλύτερος στην περίπτωση του materialized view. Αυτό οφείλεται στο γεγονός ότι το materialized view έχει ήδη εκτελεσθεί και αποθηκευτεί στην βάση με την μορφή πίνακα με απότοκο κατά την εκτέλεση να γίνεται η απευθείας ανάκτηση των δεδομένων εν αντιθέσει με το virtual view το οποίο εκτελείται realtime.

Ερώτημα 4:

Μετά από την εκτέλεση των δύο εισαγωγών στους πίνακες Student, Professor και Labstaff τόσο με indexes όσο και χωρίς οι χρόνοι εκτέλεσης που παρατηρήθηκαν έχουν ελάχιστες διαφορές πράγμα που οφείλεται στο γεγονός ότι τα χρησιμοποιούμενα indexes είναι σχετικά λίγα για να γίνουν εμφανείς οι διαφορές. Παρόλαυτα μπορούμε να παρατηρήσουμε μία ελαφριά αύξηση του χρόνου εκτέλεσης στην περίπτωση όπου γίνεται η χρήση των indexes. Παρακάτω παρατίθενται οι σχετικοί χρόνοι εκτέλεσης.

Με χρήση Indexes

```
QUERY PLAN
Result  (cost=0.00..0.26 rows=1 width=4) (actual time=10538.044..10538.045 rows=1 loops=1)
Planning time: 0.015 ms
Execution time: 10538.053 ms
```

Χωρίς Χρήση Indexes

```
QUERY PLAN
Result  (cost=0.00..0.26 rows=1 width=4) (actual time=8863.569..8863.570 rows=1 loops=1)
Planning time: 0.027 ms
Execution time: 8863.586 ms
```