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 B+-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 .

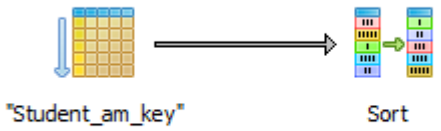
Οπότε με χρήση της εντολής EXPLAIN της Postgres μελετήσαμε τα ακόλουθα πλάνα εκτέλεσης :

1. Εκτέλεση του ερωτήματος χωρίς δημιουργία ευρετηρίων ή συσταδοποίηση

	QUERY PLAN text
1	Sort (cost=8.45..8.45 rows=1 width=84) (actual time=0.025..0.025 rows=1 loops=1)
2	Sort Key: surname, name
3	Sort Method: quicksort Memory: 25kB
4	-> Index Scan using "Student am key" on "Student" (cost=0.42..8.44 rows=1 width=84) (actual time=0.015..0.016 rows=1 loops=1)
5	Index Cond: (am = 2016083621)
6	Planning time: 0.111 ms
7	Execution time: 0.051 ms

Παρατηρούμε ότι ο 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 . Το 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 :

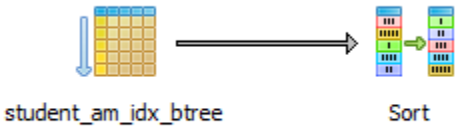


2. Εκτέλεση του ερωτήματος με χρήση B+-Tree ευρετηρίου χωρίς συσταδοποίηση

	QUERY PLAN text
1	Sort (cost=8.45..8.45 rows=1 width=84) (actual time=0.023..0.023 rows=1 loops=1)
2	Sort Key: surname, name
3	Sort Method: quicksort Memory: 25kB
4	-> Index Scan using student surname idx btree on "Student" (cost=0.42..8.44 rows=1 width=84) (actual time=0.015..0.016 rows=1 loops=1)
5	Index Cond: (am = 2016083621)
6	Planning time: 0.124 ms
7	Execution time: 0.051 ms

Παρατηρούμε ότι ο 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 :



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

	QUERY PLAN text
1	Sort (cost=8.45..8.45 rows=1 width=84) (actual time=0.028..0.028 rows=1 loops=1)
2	Sort Key: surname, name
3	Sort Method: quicksort Memory: 25kB
4	-> Index Scan using student surname idx btree on "Student" (cost=0.42..8.44 rows=1 width=84) (actual time=0.015..0.016 rows=1 loops=1)
5	Index Cond: (am = 2016083621)
6	Planning time: 0.122 ms
7	Execution time: 0.054 ms

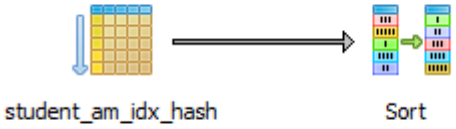
Παρατηρούμε ότι ο βελτιστοποιητής (optimizer) χρησιμοποιεί Index Scan χρησιμοποιώντας το ευρετήριο που ορίσαμε. Η ύπαρξη ευρετηρίου σε συνδυασμό με την απλότητα του ερωτήματος , οδηγεί σε εξαιρετικά μικρό χρόνο εκτέλεσης . Οι χρόνοι είναι ήδη πολύ μικροί , οπότε δεν παρατηρούμε σημαντικές αποκλίσεις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

4. Εκτέλεση του ερωτήματος με χρήση ευρετηρίου κατακερματισμού χωρίς συσταδοποίηση

	QUERY PLAN text
1	Sort (cost=8.03..8.03 rows=1 width=84) (actual time=0.025..0.025 rows=1 loops=1)
2	Sort Key: surname, name
3	Sort Method: quicksort Memory: 25kB
4	-> Index Scan using student surname idx on "Student" (cost=0.00..8.02 rows=1 width=84) (actual time=0.017..0.018 rows=1 loops=1)
5	Index Cond: (am = 2016083621)
6	Planning time: 0.126 ms
7	Execution time: 0.053 ms

Παρατηρούμε ότι ο 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 :

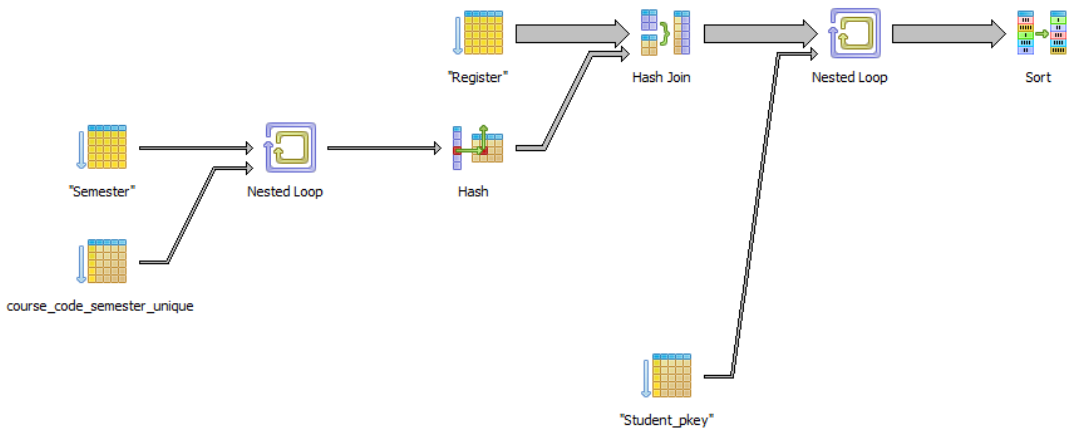
Ο κώδικας SQL του ερωτήματος είναι ο εξής :

```
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 :

	QUERY PLAN text
1	Sort (cost=3253.41..3253.41 rows=1 width=42) (actual time=29.618..29.618 rows=1 loops=1)
2	Sort Key: "Student".surname, "Student".name
3	Sort Method: quicksort Memory: 25kB
4	-> Nested Loop (cost=9.83..3253.40 rows=1 width=42) (actual time=1.599..29.609 rows=1 loops=1)
5	-> Hash Join (cost=9.53..3253.01 rows=1 width=4) (actual time=1.584..29.591 rows=1 loops=1)
6	Hash Cond: ("Register".serial_number = "CourseRun".serial_number)
7	-> Seq Scan on "Register" (cost=0.00..2856.20 rows=103270 width=12) (actual time=0.032..16.101 rows=103306 loops=1)
8	Filter: (register_status = 'approved'::register_status_type)
9	Rows Removed by Filter: 230
10	-> Hash (cost=9.52..9.52 rows=1 width=8) (actual time=0.034..0.034 rows=1 loops=1)
11	Buckets: 1024 Batches: 1 Memory Usage: 9kB
12	-> Nested Loop (cost=0.42..9.52 rows=1 width=8) (actual time=0.030..0.032 rows=1 loops=1)
13	-> Seq Scan on "Semester" (cost=0.00..1.08 rows=1 width=4) (actual time=0.002..0.003 rows=1 loops=1)
14	Filter: (semester_status = 'present'::semester_status_type)
15	Rows Removed by Filter: 5
16	-> Index Scan using course_code_semester_unique on "CourseRun" (cost=0.42..8.44 rows=1 width=12) (actual time=0.026..0.026 rows=1 loops=1)
17	Index Cond: ((semester_id = "Semester".semester_id) AND ((course_code)::text = 'MA8 411'::text))
18	-> Index Scan using "Student_pkey" on "Student" (cost=0.29..0.38 rows=1 width=46) (actual time=0.010..0.010 rows=1 loops=1)
19	Index Cond: (amka = "Register"."Student_amka")
20	Planning time: 1.019 ms
21	Execution time: 29.700 ms

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

	QUERY PLAN text
1	Sort (cost=3253.28..3253.29 rows=1 width=42) (actual time=30.439..30.440 rows=1 loops=1)
2	Sort Key: "Student".surname, "Student".name
3	Sort Method: quicksort Memory: 25kB
4	-> Nested Loop (cost=8.62..3253.27 rows=1 width=42) (actual time=3.216..30.267 rows=1 loops=1)
5	Join Filter: ("CourseRun".semester id = "Semester".semester id)
6	Rows Removed by Join Filter: 20
7	-> Nested Loop (cost=8.62..3252.18 rows=1 width=46) (actual time=0.756..29.896 rows=21 loops=1)
8	-> Hash Join (cost=8.32..3251.80 rows=1 width=8) (actual time=0.745..29.547 rows=21 loops=1)
9	Hash Cond: ("Register".serial number = "CourseRun".serial number)
10	-> Seq Scan on "Register" (cost=0.00..2856.20 rows=103270 width=12) (actual time=0.028..16.115 rows=103306 loops=1)
11	Filter: (register status = 'approved'::register status type)
12	Rows Removed by Filter: 230
13	-> Hash (cost=8.31..8.31 rows=1 width=12) (actual time=0.028..0.028 rows=2 loops=1)
14	Buckets: 1024 Batches: 1 Memory Usage: 9kB
15	-> Index Scan using course code cr b tree idx on "CourseRun" (cost=0.29..8.31 rows=1 width=12) (actual time=0.021..0.024 rows=2 loops=1)
16	Index Cond: ((course code)::text = 'MA8 411'::text)
17	-> Index Scan using "Student pkey" on "Student" (cost=0.29..0.38 rows=1 width=46) (actual time=0.013..0.014 rows=1 loops=21)
18	Index Cond: (amka = "Register"."Student amka")
19	-> Seq Scan on "Semester" (cost=0.00..1.08 rows=1 width=4) (actual time=0.003..0.004 rows=1 loops=21)
20	Filter: (semester status = 'present'::semester status type)
21	Rows Removed by Filter: 5
22	Planning time: 0.822 ms
23	Execution time: 30.512 ms

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

2. Ευρετήριο στο register_status χωρίς συσταδοποιήση

	QUERY PLAN text
1	Sort (cost=3253.41..3253.41 rows=1 width=42) (actual time=29.145..29.145 rows=1 loops=1)
2	Sort Key: "Student".surname, "Student".name
3	Sort Method: quicksort Memory: 25kB
4	-> Nested Loop (cost=9.83..3253.40 rows=1 width=42) (actual time=2.323..29.130 rows=1 loops=1)
5	-> Hash Join (cost=9.53..3253.01 rows=1 width=4) (actual time=2.297..29.103 rows=1 loops=1)
6	Hash Cond: ("Register".serial number = "CourseRun".serial number)
7	-> Seq Scan on "Register" (cost=0.00..2856.20 rows=103270 width=12) (actual time=0.044..15.795 rows=103306 loops=1)
8	Filter: (register status = 'approved'::register status type)
9	Rows Removed by Filter: 230
10	-> Hash (cost=9.52..9.52 rows=1 width=8) (actual time=0.050..0.050 rows=1 loops=1)
11	Buckets: 1024 Batches: 1 Memory Usage: 9kB
12	-> Nested Loop (cost=0.42..9.52 rows=1 width=8) (actual time=0.044..0.047 rows=1 loops=1)
13	-> Seq Scan on "Semester" (cost=0.00..1.08 rows=1 width=4) (actual time=0.005..0.006 rows=1 loops=1)
14	Filter: (semester status = 'present'::semester status type)
15	Rows Removed by Filter: 5
16	-> Index Scan using course code semester unique on "CourseRun" (cost=0.42..8.44 rows=1 width=12) (actual time=0.036..0.037 rows=1 loops=1)
17	Index Cond: ((semester id = "Semester".semester id) AND ((course code)::text = 'MA0 411'::text))
18	-> Index Scan using "Student pkey" on "Student" (cost=0.29..0.38 rows=1 width=46) (actual time=0.016..0.017 rows=1 loops=1)
19	Index Cond: (amka = "Register"."Student amka")
20	Planning time: 2.023 ms
21	Execution time: 29.265 ms

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

3. Ευρετήριο στο “Student_amka” χωρίς συσταδοποιήση

	QUERY PLAN text
1	Sort (cost=3253.41..3253.41 rows=1 width=42) (actual time=28.855..28.856 rows=1 loops=1)
2	Sort Key: "Student".surname, "Student".name
3	Sort Method: quicksort Memory: 25kB
4	-> Nested Loop (cost=9.83..3253.40 rows=1 width=42) (actual time=2.752..28.841 rows=1 loops=1)
5	-> Hash Join (cost=9.53..3253.01 rows=1 width=4) (actual time=2.714..28.802 rows=1 loops=1)
6	Hash Cond: ("Register".serial number = "CourseRun".serial number)
7	-> Seq Scan on "Register" (cost=0.00..2856.20 rows=103270 width=12) (actual time=0.043..15.820 rows=103306 loops=1)
8	Filter: (register status = 'approved'::register status type)
9	Rows Removed by Filter: 230
10	-> Hash (cost=9.52..9.52 rows=1 width=8) (actual time=0.048..0.048 rows=1 loops=1)
11	Buckets: 1024 Batches: 1 Memory Usage: 9kB
12	-> Nested Loop (cost=0.42..9.52 rows=1 width=8) (actual time=0.042..0.044 rows=1 loops=1)
13	-> Seq Scan on "Semester" (cost=0.00..1.08 rows=1 width=4) (actual time=0.005..0.006 rows=1 loops=1)
14	Filter: (semester status = 'present'::semester status type)
15	Rows Removed by Filter: 5
16	-> Index Scan using course code semester unique on "CourseRun" (cost=0.42..8.44 rows=1 width=12) (actual time=0.034..0.034 rows=1 loops=1)
17	Index Cond: ((semester id = "Semester".semester id) AND ((course code)::text = 'MA0 411'::text))
18	-> Index Scan using "Student pkey" on "Student" (cost=0.29..0.38 rows=1 width=46) (actual time=0.025..0.026 rows=1 loops=1)
19	Index Cond: (amka = "Register"."Student amka")
20	Planning time: 1.118 ms
21	Execution time: 28.982 ms

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.

4. Ευρετήριο στο amka του πίνακα Student χωρίς συσταδοποιήση

	QUERY PLAN text
1	Sort (cost=3253.41..3253.41 rows=1 width=42) (actual time=28.693..28.693 rows=1 loops=1)
2	Sort Key: "Student".surname, "Student".name
3	Sort Method: quicksort Memory: 25kB
4	-> Nested Loop (cost=9.83..3253.40 rows=1 width=42) (actual time=1.489..28.685 rows=1 loops=1)
5	-> Hash Join (cost=9.53..3253.01 rows=1 width=4) (actual time=1.478..28.673 rows=1 loops=1)
6	Hash Cond: ("Register".serial number = "CourseRun".serial number)
7	-> Seq Scan on "Register" (cost=0.00..2856.20 rows=103270 width=12) (actual time=0.039..15.435 rows=103306 loops=1)
8	Filter: (register status = 'approved'::register status type)
9	Rows Removed by Filter: 230
10	-> Hash (cost=9.52..9.52 rows=1 width=8) (actual time=0.039..0.039 rows=1 loops=1)
11	Buckets: 1024 Batches: 1 Memory Usage: 9kB
12	-> Nested Loop (cost=0.42..9.52 rows=1 width=8) (actual time=0.034..0.036 rows=1 loops=1)
13	-> Seq Scan on "Semester" (cost=0.00..1.08 rows=1 width=4) (actual time=0.004..0.005 rows=1 loops=1)
14	Filter: (semester status = 'present'::semester status type)
15	Rows Removed by Filter: 5
16	-> Index Scan using course code semester unique on "CourseRun" (cost=0.42..8.44 rows=1 width=12) (actual time=0.028..0.029 rows=1 loops=1)
17	Index Cond: ((semester id = "Semester".semester id) AND ((course code)::text = 'MA0 411'::text))
18	-> Index Scan using student amka idx on "Student" (cost=0.29..0.38 rows=1 width=46) (actual time=0.007..0.007 rows=1 loops=1)
19	Index Cond: (amka = "Register"."Student amka")
20	Planning time: 0.734 ms
21	Execution time: 28.773 ms

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 .

5. Ευρετήριο στο serial_number του πίνακα “Register” χωρίς συσταδοποίηση

	QUERY PLAN text
1	Sort (cost=18.24..18.25 rows=1 width=42) (actual time=0.068..0.069 rows=1 loops=1)
2	Sort Key: "Student".surname, "Student".name
3	Sort Method: quicksort Memory: 25kB
4	-> Nested Loop (cost=1.00..18.23 rows=1 width=42) (actual time=0.057..0.059 rows=1 loops=1)
5	-> Nested Loop (cost=0.71..17.85 rows=1 width=4) (actual time=0.052..0.053 rows=1 loops=1)
6	-> Nested Loop (cost=0.42..9.52 rows=1 width=8) (actual time=0.035..0.036 rows=1 loops=1)
7	-> Seq Scan on "Semester" (cost=0.00..1.08 rows=1 width=4) (actual time=0.006..0.007 rows=1 loops=1)
8	Filter: (semester status = 'present'::semester status type)
9	Rows Removed by Filter: 5
10	-> Index Scan using course code semester unique on "CourseRun" (cost=0.42..8.44 rows=1 width=12) (actual time=0.025..0.026 rows=1 loops=1)
11	Index Cond: ((semester id = "Semester".semester id) AND ((course code)::text = 'MA0 411'::text))
12	-> Index Scan using serial number reg idx on "Register" (cost=0.29..8.31 rows=1 width=12) (actual time=0.016..0.016 rows=1 loops=1)
13	Index Cond: (serial number = "CourseRun".serial number)
14	Filter: (register status = 'approved'::register status type)
15	-> Index Scan using "Student pkey" on "Student" (cost=0.29..0.38 rows=1 width=46) (actual time=0.003..0.004 rows=1 loops=1)
16	Index Cond: (amka = "Register"."Student amka")
17	Planning time: 1.201 ms
18	Execution time: 0.125 ms

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. Η 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 :

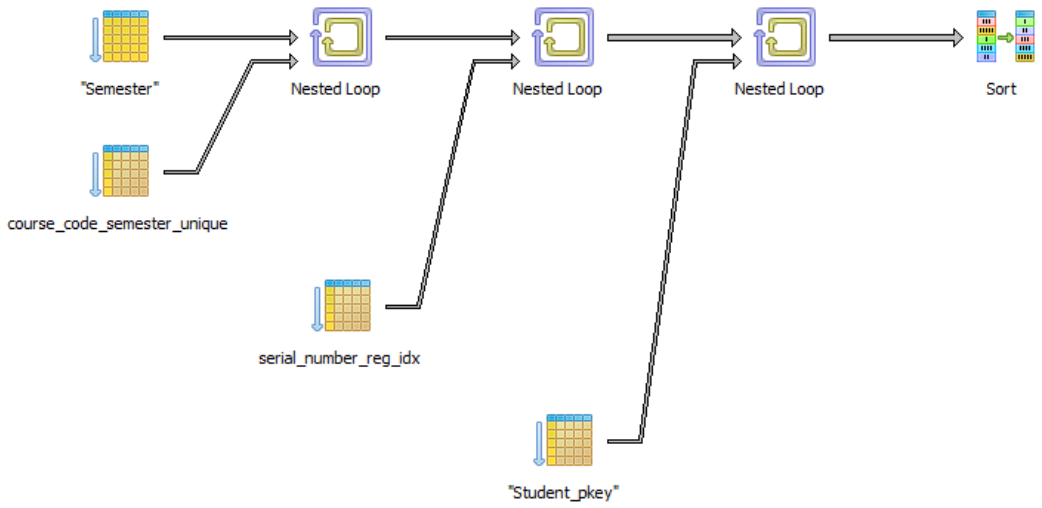
1. B+-Tree index στο serial_number χωρίς συσταδοποιηση

	QUERY PLAN text
1	Sort (cost=18.24..18.25 rows=1 width=42) (actual time=0.046..0.047 rows=1 loops=1)
2	Sort Key: "Student".surname, "Student".name
3	Sort Method: quicksort Memory: 25kB
4	-> Nested Loop (cost=1.00..18.23 rows=1 width=42) (actual time=0.036..0.039 rows=1 loops=1)
5	-> Nested Loop (cost=0.71..17.85 rows=1 width=4) (actual time=0.032..0.034 rows=1 loops=1)
6	-> Nested Loop (cost=0.42..9.52 rows=1 width=8) (actual time=0.027..0.028 rows=1 loops=1)
7	-> Seq Scan on "Semester" (cost=0.00..1.08 rows=1 width=4) (actual time=0.005..0.006 rows=1 loops=1)
8	Filter: (semester status = 'present'::semester status type)
9	Rows Removed by Filter: 5
10	-> Index Scan using course code semester unique on "CourseRun" (cost=0.42..8.44 rows=1 width=12) (actual time=0.020..0.020 rows=1 loops=1)
11	Index Cond: ((semester id = "Semester".semester id) AND ((course code)::text = 'MA8 411'::text))
12	-> Index Scan using serial number reg idx on "Register" (cost=0.29..8.31 rows=1 width=12) (actual time=0.004..0.004 rows=1 loops=1)
13	Index Cond: (serial number = "CourseRun".serial number)
14	Filter: (register status = 'approved'::register status type)
15	-> Index Scan using "Student pkey" on "Student" (cost=0.29..0.38 rows=1 width=46) (actual time=0.002..0.003 rows=1 loops=1)
16	Index Cond: (amka = "Register"."Student amka")
17	Planning time: 0.738 ms
18	Execution time: 0.100 ms

2. B+-Tree index στο serial_number με συσταδοποιηση

	QUERY PLAN text
1	Sort (cost=18.24..18.25 rows=1 width=42) (actual time=0.046..0.046 rows=1 loops=1)
2	Sort Key: "Student".surname, "Student".name
3	Sort Method: quicksort Memory: 25kB
4	-> Nested Loop (cost=1.00..18.23 rows=1 width=42) (actual time=0.037..0.039 rows=1 loops=1)
5	-> Nested Loop (cost=0.71..17.85 rows=1 width=4) (actual time=0.034..0.035 rows=1 loops=1)
6	-> Nested Loop (cost=0.42..9.52 rows=1 width=8) (actual time=0.028..0.029 rows=1 loops=1)
7	-> Seq Scan on "Semester" (cost=0.00..1.08 rows=1 width=4) (actual time=0.005..0.006 rows=1 loops=1)
8	Filter: (semester status = 'present'::semester status type)
9	Rows Removed by Filter: 5
10	-> Index Scan using course code semester unique on "CourseRun" (cost=0.42..8.44 rows=1 width=12) (actual time=0.020..0.020 rows=1 loops=1)
11	Index Cond: ((semester id = "Semester".semester id) AND ((course code)::text = 'MA8 411'::text))
12	-> Index Scan using serial number reg idx on "Register" (cost=0.29..8.31 rows=1 width=12) (actual time=0.005..0.005 rows=1 loops=1)
13	Index Cond: (serial number = "CourseRun".serial number)
14	Filter: (register status = 'approved'::register status type)
15	-> Index Scan using "Student pkey" on "Student" (cost=0.29..0.38 rows=1 width=46) (actual time=0.002..0.003 rows=1 loops=1)
16	Index Cond: (amka = "Register"."Student amka")
17	Planning time: 0.999 ms
18	Execution time: 0.103 ms

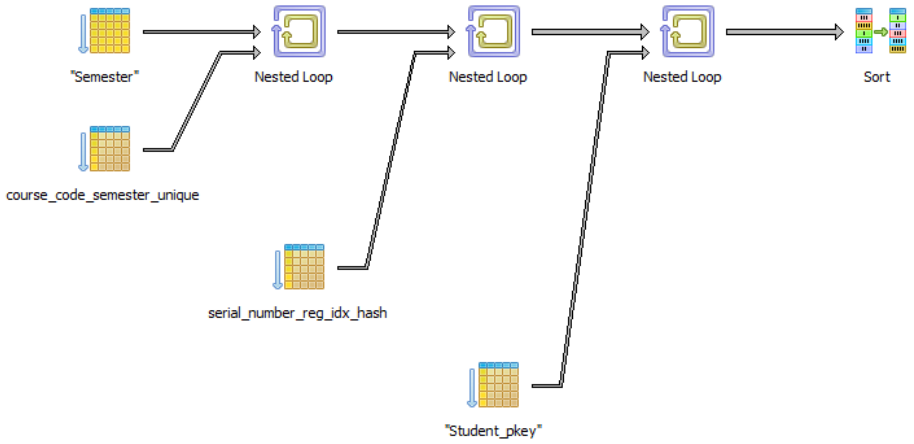
Η αντίστοιχη γραφική αναπαράσταση του EXPLAIN :



3. Hash index στο serial_number

	QUERY PLAN text
1	Sort (cost=18.24..18.25 rows=1 width=42) (actual time=0.046..0.046 rows=1 loops=1)
2	Sort Key: "Student".surname, "Student".name
3	Sort Method: quicksort Memory: 25kB
4	-> Nested Loop (cost=1.00..18.23 rows=1 width=42) (actual time=0.037..0.039 rows=1 loops=1)
5	-> Nested Loop (cost=0.71..17.85 rows=1 width=4) (actual time=0.034..0.035 rows=1 loops=1)
6	-> Nested Loop (cost=0.42..9.52 rows=1 width=8) (actual time=0.028..0.029 rows=1 loops=1)
7	-> Seq Scan on "Semester" (cost=0.00..1.08 rows=1 width=4) (actual time=0.005..0.006 rows=1 loops=1)
8	Filter: (semester status = 'present'::semester status type)
9	Rows Removed by Filter: 5
10	-> Index Scan using course code semester unique on "CourseRun" (cost=0.42..8.44 rows=1 width=12) (actual time=0.020..0.020 rows=1 loops=1)
11	Index Cond: ((semester id = "Semester".semester id) AND ((course code)::text = 'MA0 411'::text))
12	-> Index Scan using serial number reg idx on "Register" (cost=0.29..8.31 rows=1 width=12) (actual time=0.005..0.005 rows=1 loops=1)
13	Index Cond: (serial number = "CourseRun".serial number)
14	Filter: (register status = 'approved'::register status type)
15	-> Index Scan using "Student pkey" on "Student" (cost=0.29..0.38 rows=1 width=46) (actual time=0.002..0.003 rows=1 loops=1)
16	Index Cond: (amka = "Register"."Student amka")
17	Planning time: 0.999 ms
18	Execution time: 0.103 ms

Η γραφική αναπαράσταση του EXPLAIN :



Έπειτα δοκιμάσαμε την απόδοση αλλάζοντας την σειρά των joins :

B+-Tree index και Clustering :

	QUERY PLAN text
1	Sort (cost=18.24..18.25 rows=1 width=42) (actual time=0.058..0.059 rows=1 loops=1)
2	Sort Key: "Student".surname , "Student".name
3	Sort Method: quicksort Memory: 25kB
4	-> Nested Loop (cost=1.00..18.23 rows=1 width=42) (actual time=0.048..0.050 rows=1 loops=1)
5	-> Nested Loop (cost=0.71..17.85 rows=1 width=4) (actual time=0.042..0.044 rows=1 loops=1)
6	-> Nested Loop (cost=0.42..9.52 rows=1 width=8) (actual time=0.033..0.034 rows=1 loops=1)
7	-> Seq Scan on "Semester" (cost=0.00..1.08 rows=1 width=4) (actual time=0.006..0.007 rows=1 loops=1)
8	Filter: (semester status = 'present'::semester status type)
9	Rows Removed by Filter: 5
10	-> Index Scan using course code semester unique on "CourseRun" (cost=0.42..8.44 rows=1 width=12) (actual time=0.025..0.025 rows=1 loops=1)
11	Index Cond: ((semester id = "Semester".semester id) AND ((course code)::text = 'MA0 411'::text))
12	-> Index Scan using serial number reg idx on "Register" (cost=0.29..8.31 rows=1 width=12) (actual time=0.007..0.008 rows=1 loops=1)
13	Index Cond: (serial number = "CourseRun".serial number)
14	Filter: (register status = 'approved'::register status type)
15	-> Index Scan using "Student pkey" on "Student" (cost=0.29..0.38 rows=1 width=46) (actual time=0.005..0.005 rows=1 loops=1)
16	Index Cond: (amka = "Register"."Student amka")
17	Planning time: 0.841 ms
18	Execution time: 0.117 ms

Hash index :

	QUERY PLAN text
1	Sort (cost=17.95..17.96 rows=1 width=42) (actual time=0.057..0.057 rows=1 loops=1)
2	Sort Key: "Student".surname, "Student".name
3	Sort Method: quicksort Memory: 25kB
4	-> Nested Loop (cost=0.71..17.94 rows=1 width=42) (actual time=0.046..0.049 rows=1 loops=1)
5	-> Nested Loop (cost=0.42..17.55 rows=1 width=4) (actual time=0.042..0.044 rows=1 loops=1)
6	-> Nested Loop (cost=0.42..9.52 rows=1 width=8) (actual time=0.032..0.032 rows=1 loops=1)
7	-> Seq Scan on "Semester" (cost=0.00..1.08 rows=1 width=4) (actual time=0.006..0.007 rows=1 loops=1)
8	Filter: (semester status = 'present'::semester status type)
9	Rows Removed by Filter: 5
10	-> Index Scan using course code semester unique on "CourseRun" (cost=0.42..8.44 rows=1 width=12) (actual time=0.023..0.023 rows=1 loops=1)
11	Index Cond: ((semester id = "Semester".semester id) AND ((course code)::text = 'MA0 411'::text))
12	-> Index Scan using serial number reg idx on "Register" (cost=0.00..8.02 rows=1 width=12) (actual time=0.009..0.010 rows=1 loops=1)
13	Index Cond: (serial number = "CourseRun".serial number)
14	Filter: (register status = 'approved'::register status type)
15	-> Index Scan using "Student pkey" on "Student" (cost=0.29..0.38 rows=1 width=46) (actual time=0.003..0.004 rows=1 loops=1)
16	Index Cond: (amka = "Register"."Student amka")
17	Planning time: 0.784 ms
18	Execution time: 0.118 ms

Η διαφορές στους χρόνους είναι ελάχιστες .

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

- SET enable_hashjoin = off
- SET enable_mergejoin = off
- SET 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 :

	QUERY PLAN text
1	Sort (cost=7525.34..7525.34 rows=1 width=42) (actual time=72.812..72.812 rows=1 loops=1)
2	Sort Key: "Student".surname, "Student".name
3	Sort Method: quicksort Memory: 25kB
4	-> Hash Join (cost=4631.46..7525.33 rows=1 width=42) (actual time=49.505..72.798 rows=1 loops=1)
5	Hash Cond: ("CourseRun".semester id = "Semester".semester id)
6	-> Hash Join (cost=4630.37..7524.22 rows=1 width=46) (actual time=49.414..72.771 rows=21 loops=1)
7	Hash Cond: ("Student".amka = "Register"."Student amka")
8	-> Seq Scan on "Student" (cost=0.00..2516.43 rows=100643 width=46) (actual time=0.003..8.638 rows=100643 loops=1)
9	-> Hash (cost=4630.36..4630.36 rows=1 width=8) (actual time=49.397..49.397 rows=21 loops=1)
10	Buckets: 1024 Batches: 1 Memory Usage: 9kB
11	-> Hash Join (cost=2189.89..4630.36 rows=1 width=8) (actual time=21.044..49.376 rows=21 loops=1)
12	Hash Cond: ("Register".serial number = "CourseRun".serial number)
13	-> Seq Scan on "Register" (cost=0.00..2053.20 rows=103270 width=12) (actual time=0.014..15.031 rows=103306 loops=1)
14	Filter: (register status = 'approved'::register status type)
15	Rows Removed by Filter: 230
16	-> Hash (cost=2189.88..2189.88 rows=1 width=12) (actual time=20.933..20.933 rows=2 loops=1)
17	Buckets: 1024 Batches: 1 Memory Usage: 9kB
18	-> Seq Scan on "CourseRun" (cost=0.00..2189.88 rows=1 width=12) (actual time=0.026..20.930 rows=2 loops=1)
19	Filter: ((course code)::text = 'MA0 411'::text)
20	Rows Removed by Filter: 100228
21	-> Hash (cost=1.08..1.08 rows=1 width=4) (actual time=0.014..0.014 rows=1 loops=1)
22	Buckets: 1024 Batches: 1 Memory Usage: 9kB
23	-> Seq Scan on "Semester" (cost=0.00..1.08 rows=1 width=4) (actual time=0.007..0.009 rows=1 loops=1)
24	Filter: (semester status = 'present'::semester status type)
25	Rows Removed by Filter: 5
26	Planning time: 0.892 ms
27	Execution time: 72.906 ms

Με hash ευρετήριο :

	QUERY PLAN text
1	Sort (cost=7525.34..7525.34 rows=1 width=42) (actual time=75.394..75.395 rows=1 loops=1)
2	Sort Key: "Student".surname, "Student".name
3	Sort Method: quicksort Memory: 25kB
4	-> Hash Join (cost=4631.46..7525.33 rows=1 width=42) (actual time=54.212..75.380 rows=1 loops=1)
5	Hash Cond: ("CourseRun".semester id = "Semester".semester id)
6	-> Hash Join (cost=4630.37..7524.22 rows=1 width=46) (actual time=54.134..75.358 rows=21 loops=1)
7	Hash Cond: ("Student".amka = "Register"."Student amka")
8	-> Seq Scan on "Student" (cost=0.00..2516.43 rows=100643 width=46) (actual time=0.004..7.671 rows=100643 loops=1)
9	-> Hash (cost=4630.36..4630.36 rows=1 width=8) (actual time=54.120..54.120 rows=21 loops=1)
10	Buckets: 1024 Batches: 1 Memory Usage: 9kB
11	-> Hash Join (cost=2189.89..4630.36 rows=1 width=8) (actual time=18.066..54.088 rows=21 loops=1)
12	Hash Cond: ("Register".serial number = "CourseRun".serial number)
13	-> Seq Scan on "Register" (cost=0.00..2053.20 rows=103270 width=12) (actual time=0.023..19.140 rows=103306 loops=1)
14	Filter: (register status = 'approved'::register status type)
15	Rows Removed by Filter: 230
16	-> Hash (cost=2189.88..2189.88 rows=1 width=12) (actual time=17.975..17.975 rows=2 loops=1)
17	Buckets: 1024 Batches: 1 Memory Usage: 9kB
18	-> Seq Scan on "CourseRun" (cost=0.00..2189.88 rows=1 width=12) (actual time=0.039..17.972 rows=2 loops=1)
19	Filter: ((course code)::text = 'MA0 411'::text)
20	Rows Removed by Filter: 100228
21	-> Hash (cost=1.08..1.08 rows=1 width=4) (actual time=0.013..0.013 rows=1 loops=1)
22	Buckets: 1024 Batches: 1 Memory Usage: 9kB
23	-> Seq Scan on "Semester" (cost=0.00..1.08 rows=1 width=4) (actual time=0.008..0.009 rows=1 loops=1)
24	Filter: (semester status = 'present'::semester status type)
25	Rows Removed by Filter: 5
26	Planning time: 1.253 ms
27	Execution time: 75.512 ms

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 :

	QUERY PLAN text
1	Sort (cost=10519.32..10519.33 rows=1 width=42) (actual time=31.810..31.810 rows=1 loops=1)
2	Sort Key: "Student".surname, "Student".name
3	Sort Method: quicksort Memory: 25kB
4	-> Merge Join (cost=6163.78..10519.31 rows=1 width=42) (actual time=31.785..31.786 rows=1 loops=1)
5	Merge Cond: ("Student".amka = "Register"."Student amka")
6	-> Index Scan using "Student pkey" on "Student" (cost=0.29..4130.94 rows=100643 width=46) (actual time=0.020..0.100 rows=231 loops=1)
7	-> Sort (cost=6163.48..6163.49 rows=1 width=4) (actual time=31.636..31.637 rows=1 loops=1)
8	Sort Key: "Register"."Student amka"
9	Sort Method: quicksort Memory: 25kB
10	-> Merge Join (cost=2191.30..6163.47 rows=1 width=4) (actual time=31.615..31.619 rows=1 loops=1)
11	Merge Cond: ("Register".serial number = "CourseRun".serial number)
12	-> Index Scan using serial number reg idx on "Register" (cost=0.29..3714.28 rows=103270 width=12) (actual time=0.018..3.096 rows=3516 loops=1)
13	Filter: (register status = 'approved'::register status type)
14	Rows Removed by Filter: 230
15	-> Sort (cost=2191.01..2191.01 rows=1 width=8) (actual time=28.084..28.086 rows=1 loops=1)
16	Sort Key: "CourseRun".serial number
17	Sort Method: quicksort Memory: 25kB
18	-> Merge Join (cost=2190.97..2191.00 rows=1 width=8) (actual time=27.909..27.910 rows=1 loops=1)
19	Merge Cond: ("CourseRun".semester id = "Semester".semester id)
20	-> Sort (cost=2189.89..2189.89 rows=1 width=12) (actual time=27.784..27.784 rows=2 loops=1)
21	Sort Key: "CourseRun".semester id
22	Sort Method: quicksort Memory: 25kB
23	-> Seq Scan on "CourseRun" (cost=0.00..2189.88 rows=1 width=12) (actual time=0.043..27.769 rows=2 loops=1)
24	Filter: ((course code)::text = 'MA0 411'::text)
25	Rows Removed by Filter: 100228
26	-> Sort (cost=1.09..1.09 rows=1 width=4) (actual time=0.117..0.117 rows=1 loops=1)
27	Sort Key: "Semester".semester id
28	Sort Method: quicksort Memory: 25kB
29	-> Seq Scan on "Semester" (cost=0.00..1.08 rows=1 width=4) (actual time=0.011..0.011 rows=1 loops=1)
30	Filter: (semester status = 'present'::semester status type)
31	Rows Removed by Filter: 5
32	Planning time: 1.491 ms
33	Execution time: 32.036 ms

Με hash ευρετήριο :

	QUERY PLAN text
1	Sort (cost=17716.78..17716.78 rows=1 width=42) (actual time=124.651..124.651 rows=1 loops=1)
2	Sort Key: "Student".surname, "Student".name
3	Sort Method: quicksort Memory: 25kB
4	-> Merge Join (cost=13361.23..17716.77 rows=1 width=42) (actual time=124.638..124.638 rows=1 loops=1)
5	Merge Cond: ("Student".amka = "Register"."Student amka")
6	-> Index Scan using "Student pkey" on "Student" (cost=0.29..4130.94 rows=100643 width=46) (actual time=0.007..0.054 rows=231 loops=1)
7	-> Sort (cost=13360.94..13360.94 rows=1 width=4) (actual time=124.557..124.557 rows=1 loops=1)
8	Sort Key: "Register"."Student amka"
9	Sort Method: quicksort Memory: 25kB
10	-> Merge Join (cost=12844.61..13360.93 rows=1 width=4) (actual time=124.550..124.551 rows=1 loops=1)
11	Merge Cond: ("CourseRun".serial number = "Register".serial number)
12	-> Sort (cost=2191.01..2191.01 rows=1 width=8) (actual time=18.853..18.853 rows=1 loops=1)
13	Sort Key: "CourseRun".serial number
14	Sort Method: quicksort Memory: 25kB
15	-> Merge Join (cost=2190.97..2191.00 rows=1 width=8) (actual time=18.849..18.850 rows=1 loops=1)
16	Merge Cond: ("CourseRun".semester id = "Semester".semester id)
17	-> Sort (cost=2189.89..2189.89 rows=1 width=12) (actual time=18.792..18.792 rows=2 loops=1)
18	Sort Key: "CourseRun".semester id
19	Sort Method: quicksort Memory: 25kB
20	-> Seq Scan on "CourseRun" (cost=0.00..2189.88 rows=1 width=12) (actual time=0.026..18.786 rows=2 loops=1)
21	Filter: ((course code)::text = 'MA0 411'::text)
22	Rows Removed by Filter: 100228
23	-> Sort (cost=1.09..1.09 rows=1 width=4) (actual time=0.048..0.049 rows=1 loops=1)
24	Sort Key: "Semester".semester id
25	Sort Method: quicksort Memory: 25kB
26	-> Seq Scan on "Semester" (cost=0.00..1.08 rows=1 width=4) (actual time=0.015..0.016 rows=1 loops=1)
27	Filter: (semester status = 'present'::semester status type)
28	Rows Removed by Filter: 5
29	-> Sort (cost=10653.56..10911.73 rows=103270 width=12) (actual time=105.112..105.479 rows=3516 loops=1)
30	Sort Key: "Register".serial number
31	Sort Method: external sort Disk: 2624kB
32	-> Seq Scan on "Register" (cost=0.00..2053.20 rows=103270 width=12) (actual time=0.035..24.410 rows=103306 loops=1)
33	Filter: (register status = 'approved'::register status type)
34	Rows Removed by Filter: 230
35	Planning time: 0.836 ms
36	Execution time: 126.349 ms

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 ο 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 (Nested 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

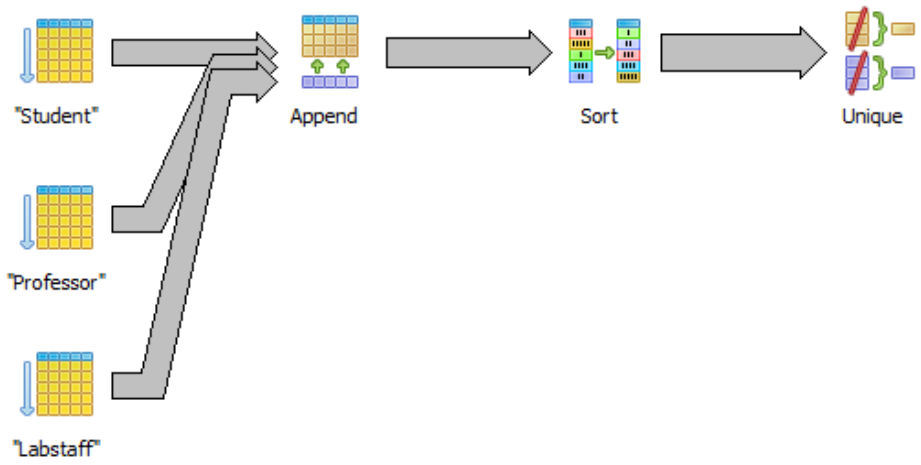
Παρατηρούμε ότι στο παραπάνω ερώτημα , δεν έχουμε ούτε join ούτε κάποιο ερώτημα σε WHERE . Επομένως ο μοναδικός τρόπος να επιταχύνουμε το ερώτημα είναι με δημιουργία ευρετηρίων B+-Trees στα select attributes ,του κάθε πίνακα και με ομαδοποίηση των πλειάδων χρησιμοποιώντας αυτά.

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

	QUERY PLAN text
1	Unique (cost=70523.68..75039.71 rows=451603 width=34) (actual time=4145.674..5349.169 rows=451035 loops=1)
2	-> Sort (cost=70523.68..71652.69 rows=451603 width=34) (actual time=4145.673..5240.991 rows=451603 loops=1)
3	Sort Key: "Student".surname, "Student".name, ('Student'::text)
4	Sort Method: external merge Disk: 23736kB
5	-> Append (cost=0.00..15756.06 rows=451603 width=34) (actual time=0.012..149.298 rows=451603 loops=1)
6	-> Seq Scan on "Student" (cost=0.00..3766.43 rows=150643 width=34) (actual time=0.011..42.060 rows=150643 loops=1)
7	-> Seq Scan on "Professor" (cost=0.00..3733.19 rows=150319 width=34) (actual time=0.003..42.307 rows=150319 loops=1)
8	-> Seq Scan on "Labstaff" (cost=0.00..3740.41 rows=150641 width=34) (actual time=0.003..41.118 rows=150641 loops=1)
9	Planning time: 0.185 ms
10	Execution time: 5368.187 ms

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 :



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

	QUERY PLAN text
1	Unique (cost=70276.68..74792.71 rows=451603 width=34) (actual time=3931.627..5122.776 rows=451035 loops=1)
2	-> Sort (cost=70276.68..71405.69 rows=451603 width=34) (actual time=3931.626..5015.825 rows=451603 loops=1)
3	Sort Key: "Student".surname, "Student".name, ('Student'::text)
4	Sort Method: external merge Disk: 23736kB
5	-> Append (cost=0.00..15509.06 rows=451603 width=34) (actual time=0.009..124.702 rows=451603 loops=1)
6	-> Seq Scan on "Student" (cost=0.00..3766.43 rows=150643 width=34) (actual time=0.009..33.452 rows=150643 loops=
7	-> Seq Scan on "Professor" (cost=0.00..3609.19 rows=150319 width=34) (actual time=0.003..34.270 rows=150319 loop
8	-> Seq Scan on "Labstaff" (cost=0.00..3617.41 rows=150641 width=34) (actual time=0.005..34.273 rows=150641 loops
9	Planning time: 0.161 ms
10	Execution time: 5142.154 ms

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

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

	QUERY PLAN text
1	Unique (cost=70276.68..74792.71 rows=451603 width=34) (actual time=3948.818..5166.418 rows=451035 loops=1)
2	-> Sort (cost=70276.68..71405.69 rows=451603 width=34) (actual time=3948.817..5056.716 rows=451603 loops=1)
3	Sort Key: "Student".surname, "Student".name, ('Student'::text)
4	Sort Method: external merge Disk: 23736kB
5	-> Append (cost=0.00..15509.06 rows=451603 width=34) (actual time=0.010..124.783 rows=451603 loops=1)
6	-> Seq Scan on "Student" (cost=0.00..3766.43 rows=150643 width=34) (actual time=0.010..33.322 rows=150643 loops=1)
7	-> Seq Scan on "Professor" (cost=0.00..3609.19 rows=150319 width=34) (actual time=0.004..34.644 rows=150319 loops=1)
8	-> Seq Scan on "Labstaff" (cost=0.00..3617.41 rows=150641 width=34) (actual time=0.004..34.246 rows=150641 loops=1)
9	Planning time: 0.178 ms
10	Execution time: 5185.349 ms

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 .)

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

	QUERY PLAN text
1	Unique (cost=70275.68..74791.71 rows=451603 width=34) (actual time=4039.773..5610.404 rows=451035 loops=1)
2	-> Sort (cost=70275.68..71404.69 rows=451603 width=34) (actual time=4039.771..5463.931 rows=451603 loops=1)
3	Sort Key: "Student".surname, "Student".name, ('Student'::text)
4	Sort Method: external merge Disk: 23736kB
5	-> Append (cost=0.00..15508.06 rows=451603 width=34) (actual time=0.011..130.324 rows=451603 loops=1)
6	-> Seq Scan on "Student" (cost=0.00..3766.43 rows=150643 width=34) (actual time=0.011..33.747 rows=150643 loops=1)
7	-> Seq Scan on "Professor" (cost=0.00..3609.19 rows=150319 width=34) (actual time=0.005..36.634 rows=150319 loops=1)
8	-> Seq Scan on "Labstaff" (cost=0.00..3616.41 rows=150641 width=34) (actual time=0.005..36.832 rows=150641 loops=1)
9	Planning time: 0.230 ms
10	Execution time: 5634.013 ms

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

	QUERY PLAN text
1	HashAggregate (cost=2378.10..2378.23 rows=13 width=86) (actual time=38.987..38.993 rows=16 loops=1)
2	Group Key: st.amka, st.am, st.name, st.surname, st.father name, st.email, st.entry date
3	Buffers: shared hit=15742
4	-> Nested Loop (cost=5.88..2377.81 rows=13 width=86) (actual time=0.144..38.687 rows=189 loops=1)
5	Buffers: shared hit=15742
6	-> Nested Loop (cost=5.59..2331.19 rows=119 width=6) (actual time=0.073..28.405 rows=3536 loops=1)
7	Buffers: shared hit=5134
8	-> Hash Join (cost=5.59..2321.90 rows=115 width=10) (actual time=0.061..23.206 rows=230 loops=1)
9	Hash Cond: ((cr.course code)::bpchar = c.course code)
10	Buffers: shared hit=940
11	-> Seq Scan on "CourseRun" cr (cost=0.00..1939.30 rows=100230 width=13) (actual time=0.003..5.926 rows=100230 loops=1)
12	Buffers: shared hit=937
13	-> Hash (cost=4.15..4.15 rows=115 width=12) (actual time=0.049..0.049 rows=115 loops=1)
14	Buckets: 1024 Batches: 1 Memory Usage: 14kB
15	Buffers: shared hit=3
16	-> Seq Scan on "Course" c (cost=0.00..4.15 rows=115 width=12) (actual time=0.002..0.027 rows=115 loops=1)
17	Buffers: shared hit=3
18	-> Index Scan using serial number reg idx on "Register" r (cost=0.00..0.07 rows=1 width=12) (actual time=0.003..0.019 rows=15 loops=230)
19	Index Cond: (serial number = cr.serial number)
20	Filter: (register status = 'approved'::register status type)
21	Rows Removed by Filter: 1
22	Buffers: shared hit=4194
23	-> Index Scan using "Student pkey" on "Student" st (cost=0.29..0.38 rows=1 width=84) (actual time=0.002..0.003 rows=0 loops=3536)
24	Index Cond: (amka = r."Student amka")
25	Filter: ((surname)::text ~~ '%ΑΛ%'::text)
26	Rows Removed by Filter: 1
27	Buffers: shared hit=10608
28	Planning time: 0.837 ms
29	Execution time: 39.120 ms

2. Virtual View

	QUERY PLAN text
1	HashAggregate (cost=2378.10..2378.23 rows=13 width=86) (actual time=35.724..35.730 rows=16 loops=1)
2	Group Key: st.amka, st.am, st.name, st.surname, st.father name, st.email, st.entry date
3	Buffers: shared hit=15742
4	-> Nested Loop (cost=5.88..2377.81 rows=13 width=86) (actual time=0.206..35.486 rows=189 loops=1)
5	Buffers: shared hit=15742
6	-> Nested Loop (cost=5.59..2331.19 rows=119 width=6) (actual time=0.083..27.192 rows=3536 loops=1)
7	Buffers: shared hit=5134
8	-> Hash Join (cost=5.59..2321.90 rows=115 width=10) (actual time=0.069..22.818 rows=230 loops=1)
9	Hash Cond: ((cr.course code)::bpchar = c.course code)
10	Buffers: shared hit=940
11	-> Seq Scan on "CourseRun" cr (cost=0.00..1939.30 rows=100230 width=13) (actual time=0.003..5.924 rows=100230 loops=1)
12	Buffers: shared hit=937
13	-> Hash (cost=4.15..4.15 rows=115 width=12) (actual time=0.057..0.057 rows=115 loops=1)
14	Buckets: 1024 Batches: 1 Memory Usage: 14kB
15	Buffers: shared hit=3
16	-> Seq Scan on "Course" c (cost=0.00..4.15 rows=115 width=12) (actual time=0.003..0.028 rows=115 loops=1)
17	Buffers: shared hit=3
18	-> Index Scan using serial number reg idx on "Register" r (cost=0.00..0.07 rows=1 width=12) (actual time=0.003..0.016 rows=15 loops=230)
19	Index Cond: (serial number = cr.serial number)
20	Filter: (register status = 'approved'::register status type)
21	Rows Removed by Filter: 1
22	Buffers: shared hit=4194
23	-> Index Scan using "Student pkey" on "Student" st (cost=0.29..0.38 rows=1 width=84) (actual time=0.002..0.002 rows=0 loops=3536)
24	Index Cond: (amka = r."Student amka")
25	Filter: ((surname)::text ~~ '%ΑΛ%'::text)
26	Rows Removed by Filter: 1
27	Buffers: shared hit=10608
28	Planning time: 0.847 ms
29	Execution time: 35.838 ms

2. 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

Hash

	QUERY PLAN text
1	Result (cost=0.00..0.26 rows=1 width=0) (actual time=41777.551..41777.551 rows=1 loops=1)
2	Planning time: 0.027 ms
3	Execution time: 41777.569 ms

B+-Tree

	QUERY PLAN text
1	Result (cost=0.00..0.26 rows=1 width=0) (actual time=42066.805..42066.805 rows=1 loops=1)
2	Planning time: 0.014 ms
3	Execution time: 42066.814 ms

2. Εισαγωγή χωρίς ευρετήρια

	QUERY PLAN text
1	Result (cost=0.00..0.26 rows=1 width=0) (actual time=40728.383..40728.383 rows=1 loops=1)
2	Planning time: 0.126 ms
3	Execution time: 40728.400 ms

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 .