Βάσεις Δεδομένων - Αναφορά Β' Φάσης

Μελέτη απόδοσης ερωτήσεων – Φυσικός Σχεδιασμός

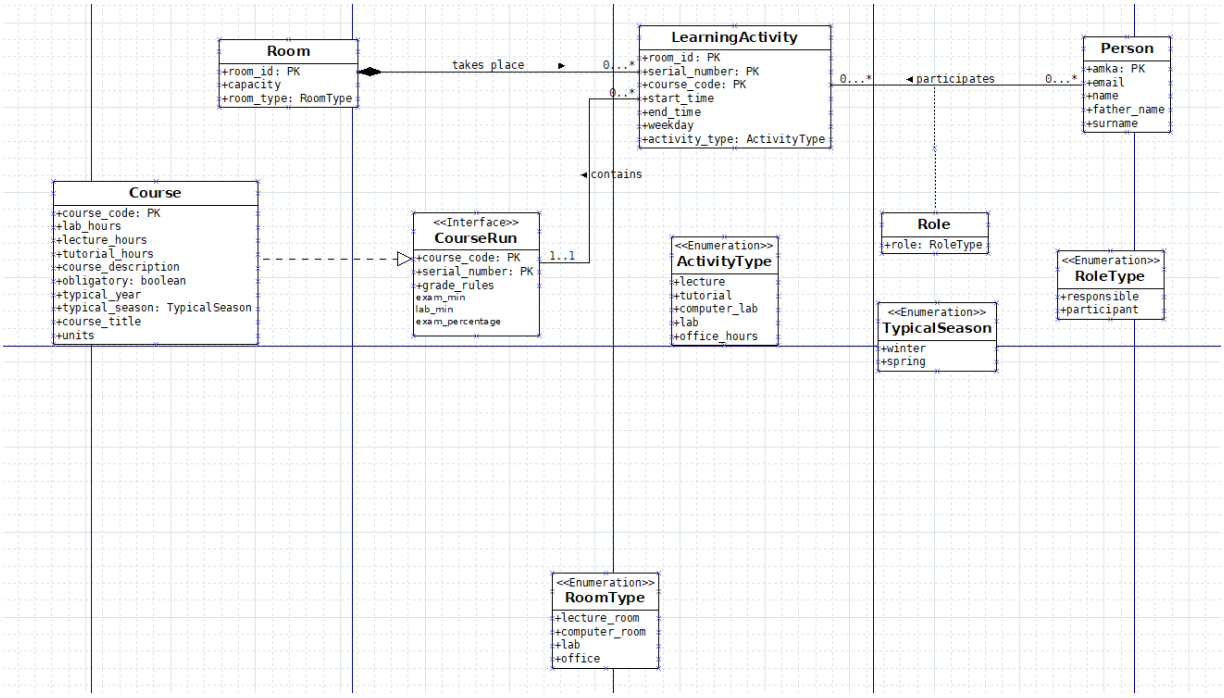
Ομάδα Εργασίας : LAB30244929

Μπαντουράκη Μαρία ΑΜ : 2017030095

Αμπλιανίτης Κωνσταντίνος ΑΜ : 2017030014

Α'Φαση

UML Διάγραμμα



Αντιστοίχιση συναρτήσεων με τα ερωτήματα της πρώτης φάσης

Ερώτημα 3.1:

Δημιουργία n αριθμού καθηγητών με τυχαίο τρόπο → q3_1_professors(n integer)

Δημιουργία n αριθμού εργαστηριακού προσωπικού → q3_1_insert_labstaff(n integer)

Δημιουργία n αριθμού φοιτητών → q3_1_students_insert(year integer, startval integer, num integer)

Ερώτημα 3.2:

Χρησιμοποιήσαμε την συνάρτηση:
q3_2_gradesupdateonspecificsemester(typicalseason semester_season_type, typicalyear integer)

Ερώτημα 4:

Ερώτημα 4.1 → q4_1_capacity_greater_than30 ()

Ερώτημα 4.2 → q4_2_display_office_hours ()

Ερώτημα 4.3 → q4_3_calculate_max_grade4(typ_year integer, typ_season semester_season_type, grade_type varchar)

Ερώτημα 4.4 → q4_4_check_computer_room ()

Ερώτημα 4.5 → q4_5_check_if_late_activity()

Ερώτημα 4.6 → q4_6_no_lab_room()

Ερώτημα 4.7 → q4_7_labstaff_busyness()

Ερώτημα 4.8 → q_4_8_roomsWithMostDifferentCourses()

Ερώτημα 4.9 → q4_9_roombusynessperday ()

Ερώτημα 4.10 → q4_10_amkaofresponsibles (mincap integer, maxcap integer)

Ερώτημα 5:

Ερώτημα 5.1.1 → q5_1_1_checkifavl ()

Ερώτημα 5.2.1 → q5_2_1_validtime(), q5_2_1_validweekday()

Ερώτημα 5.2.2 → q5_2_2_roomavl()

Ερώτημα 5.3→ q5_3_semesterqualities(), q5_3_newsemester()

Ερώτημα 6

6.1 → q6_1_goodstudentsinlabs

6.2 → q6_2_timetable

Α) Το αίτημα του πρώτου ερωτήματος ήταν :Βρες τους φοιτητές που έχουν ένα ορισμένο πατρώνυμο.

Αρχικά, ελέγξαμε πόσοι φοιτητές ήταν καταγεγραμμένοι στον πίνακα Student, μέσω του ερωτήματος :

select count(*)
from "Student" , το οποίο μας έδειξε ότι στη βάση μας υπήρχαν 210 φοιτητές.

Εκτελέσαμε το query για την αναζήτηση φοιτητών με συγκεκριμένο πατρώνυμο, αρχικά χωρίς ευρετήρια και χρησιμοποιώντας την εντολή explain analyze, ώστε να μελετήσουμε το πλάνο και το χρόνο εκτέλεσης.

Ο μέσος χρόνος εκτέλεσης ήταν 0,140 ms και το ερώτημα υλοποιήθηκε ως εξής:

explain analyze
select *
from "Student"
where father_name='ΖΗΣΗΣ'

	QUERY PLAN text
1	Seq Scan on "Student" (cost=0.00..8.85 rows=4 width=165) (actual time=0.035..0.098 rows=3 loops=1)
2	Filter: (father_name = 'ΖΗΣΗΣ'::bpchar)
3	Rows Removed by Filter: 207
4	Planning Time: 0.140 ms
5	Execution Time: 0.121 ms

Στη συνέχεια, τρέξαμε ξανά το ίδιο ερώτημα, χρησιμοποιώντας B+tree index, το οποίο δημιουργήσαμε στον πίνακα Student και στη στήλη father_name με την εξής εντολή :

create index student_index on "Student"(father_name)

Ο μέσος χρόνος εκτέλεσης προέκυψε 0,064 ms και το query plan διαμορφώθηκε έτσι :

	QUERY PLAN text
1	Seq Scan on "Student" (cost=0.00..8.63 rows=3 width=165) (actual time=0.022..0.051 rows=3 loops=1)
2	Filter: (father_name = 'ΖΗΣΗΣ'::bpchar)
3	Rows Removed by Filter: 207
4	Planning Time: 0.171 ms
5	Execution Time: 0.065 ms

Έπειτα, ακολουθήσαμε και πάλι την ίδια διαδικασία, χρησιμοποιώντας hash-index, το οποίο δημιουργήσαμε ως εξής :

create index hash_student_index on "Student" using hash (father_name)

Ο μέσος χρόνος εκτέλεσης ήταν 0,056 ms και το query plan έδωσε αυτά τα αποτελέσματα :

	QUERY PLAN text
1	Seq Scan on "Student" (cost=0.00..8.63 rows=3 width=165) (actual time=0.022..0.050 rows=3 loops=1)
2	Filter: (father_name = 'ΖΗΣΗΣ'::bpchar)
3	Rows Removed by Filter: 207
4	Planning Time: 0.112 ms
5	Execution Time: 0.062 ms

Παρατηρείται ότι με τη χρήση ευρετηρίων μειώνεται περίπου στο μισό ο χρόνος εκτέλεσης του ερωτήματος, ενώ ο hash index είναι ελάχιστα πιο γρήγορος από τον B+tree index.

Τέλος, δοκιμάσαμε τη μέθοδο της συσταδοποίησης (clustering) για το ίδιο ερώτημα, με την ακόλουθη εντολή :

cluster "Student" using student_index;

Ο μέσος χρόνος εκτέλεσης αυτή τη φόρα έγινε 0,051 ms, ενώ το query plan διαμορφώθηκε ως εξής :

	QUERY PLAN text
1	Seq Scan on "Student" (cost=0.00..8.63 rows=3 width=165) (actual time=0.025..0.051 rows=3 loops=1)
2	Filter: (father_name = 'ΖΗΣΗΣ'::bpchar)
3	Rows Removed by Filter: 207
4	Planning Time: 0.577 ms
5	Execution Time: 0.063 ms

Η μέθοδος της συσταδοποίησης βελτίωσε για κάποια ms την αναζήτηση μέσω B+tree index.

Στη συνέχεια χρειάστηκε να επαναλάβουμε όλα τα προηγούμενα βήματα για μερικές χιλιάδες φοιτητών. Δημιουργήσαμε πολλούς ακόμα φοιτητές με τη χρήση της δοθείσας συνάρτησης :

```
CREATE OR REPLACE FUNCTION public.insert_1000_students_per_year(
    yearstart integer,
    yearfinish integer)
  RETURNS void AS
$BODY$
DECLARE
        year integer;
        round integer;
BEGIN
        FOR year in yearstart..yearfinish LOOP
                FOR round in 1..5 LOOP
                        PERFORM insert_students(year,200,200*(round-1)+1);
                END LOOP;
        END LOOP;
END;
$BODY$
  LANGUAGE plpgsql VOLATILE
  COST 100;
```

Με τη χρήση του ερωτήματος που αναφέρθηκε παραπάνω, διαπιστώθηκε ότι πλέον οι φοιτητές στον πίνακα Student έγιναν 100.210 .

Εκτελώντας το ίδιο ερώτημα χωρίς τη χρήση ευρετηρίων, λάβαμε μέσο χρόνο απόδοσης 24,99 ms και το εξής query plan:

	QUERY PLAN text
1	Seq Scan on "Student" (cost=0.00..3708.63 rows=361 width=166) (actual time=0.047..28.288 rows=371 loops…
2	Filter: (father_name = 'ΖΗΣΗΣ'::bpchar)
3	Rows Removed by Filter: 99839
4	Planning Time: 0.114 ms
5	Execution Time: 28.342 ms

Στη συνέχεια χρησιμοποιήσαμε B+tree index, ακριβώς όπως και πριν, προέκυψε μέσος χρόνος απόδοσης 0,464 ms και τα αποτελέσματα διαμορφώθηκαν ως εξής :

	QUERY PLAN text
1	Bitmap Heap Scan on "Student" (cost=15.22..993.23 rows=361 width=166) (actual time=0.147..0.448 rows=371 loops=1)
2	Recheck Cond: (father_name = 'ΖΗΣΗΣ'::bpchar)
3	Heap Blocks: exact=365
4	-> Bitmap Index Scan on student_index (cost=0.00..15.13 rows=361 width=0) (actual time=0.103..0.103 rows=371 loops=1)
5	Index Cond: (father_name = 'ΖΗΣΗΣ'::bpchar)
6	Planning Time: 0.110 ms
7	Execution Time: 0.486 ms

Σειρά έχει το hash index. Προέκυψε μέσος χρόνος απόδοσης 0,830 ms και το query plan ως εξής :

	QUERY PLAN text
1	Bitmap Heap Scan on "Student" (cost=14.80..992.81 rows=361 width=166) (actual time=0.147..0.846 rows=371 loops=1)
2	Recheck Cond: (father_name = 'ΖΗΣΗΣ'::bpchar)
3	Heap Blocks: exact=365
4	-> Bitmap Index Scan on hash_student_index (cost=0.00..14.71 rows=361 width=0) (actual time=0.075..0.075 rows=371 loops=1)
5	Index Cond: (father_name = 'ΖΗΣΗΣ'::bpchar)
6	Planning Time: 0.185 ms
7	Execution Time: 0.912 ms

Τέλος, δοκιμάσαμε ξανά τη μέθοδο της συσταδοποίησης. Το query plan μας έδωσε μέσο χρόνο απόδοσης 0,211 ms και τα εξής αποτελέσματα :

	QUERY PLAN text
1	Bitmap Heap Scan on "Student" (cost=15.22..994.14 rows=361 width=166) (actual time=0.145..0.182 rows=371 loops=1)
2	Recheck Cond: (father_name = 'ΖΗΣΗΣ'::bpchar)
3	Heap Blocks: exact=10
4	-> Bitmap Index Scan on student_index (cost=0.00..15.13 rows=361 width=0) (actual time=0.138..0.138 rows=371 loops=1)
5	Index Cond: (father_name = 'ΖΗΣΗΣ'::bpchar)
6	Planning Time: 0.951 ms
7	Execution Time: 0.220 ms

Παρατηρούμε ότι ο αρχικός χρόνος εκτέλεσης του ερωτήματος αυξάνεται κατά πολλά ms, κάτι που ωστόσο ήταν αναμενόμενο λόγω της μεγάλης αύξησης του αριθμού των φοιτητών. Η χρήση των ευρετηρίων και αυτή τη φορά είναι πολύ κρίσιμη, καθώς μειώνει σημαντικά το χρόνο εκτέλεσης. Ο B+tree index φαίνεται πιο αποδοτικός από τον hash index, καθώς δίνει περίπου το μισό χρόνο εκτέλεσης από τον δεύτερο. Η μέθοδος της συσταδοποίησης στην περίπτωση των χιλιάδων φοιρητών φαίνεται ότι δίνει τα καλύτερα δυνατά αποτελέσματα, καθώς μειώνει το χρόνο εκτέλεσης της αναζήτησης μέσω B+tree index στο μισό.

Συμπεράσματα

Συμπεραίνουμε ότι για κάποιες εκατοντάδες φοιτητών και οι δύο τύποι ευρετηρίων είναι αρκετά αποδοτικοί, καθώς μειώνουν κατά το ήμισυ το χρόνο εκτέλεσης, ενώ ο hash index φαίνεται ελάχιστα πιο αποδοτικός.

Στην περίπτωση των χιλιάδων φοιτητών, όμως, ο B+tree index φαίνεται ότι έχει πολύ καλύτερη απόδοση από τον hash index, καθώς έχει τον μισό χρόνο εκτέλεσης. Σημειώνεται μάλιστα πως η μέθοδος της συσταδοποίησης μείωσε ξανά στο μισό το χρόνο εκτέλεσης, δίνοντας έτσι έναν πολύ καλό χρόνο απόδοσης, σε αντίθεση με την περίπτωση των εκατοντάδων φοιτητών, όπου βελτίωσε τον χρόνο μόνο για λίγα ms.

Β) Το αίτημα του δεύτερου ερωτήματος ήταν: Βρες φοιτήτριες (έχουν ονόματα θηλυκού γένους) που έχουν περάσει μάθημα με συγκεκριμένο κωδικό μαθήματος και συγκεκριμένο τελικό βαθμό.

Εκτελέσαμε το query για την αναζήτηση φοιτητριών που έχουν περάσει μάθημα με συγκεκριμένο κωδικό μαθήματος και συγκεκριμένο τελικό βαθμό και χρησιμοποιώντας την εντολή explain analyze, ώστε να μελετήσουμε το πλάνο και το χρόνο εκτέλεσης.

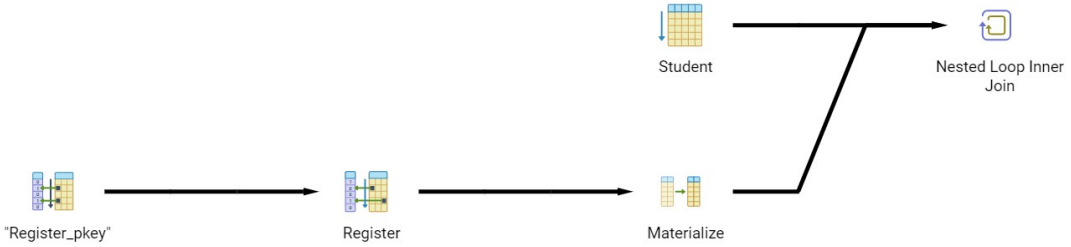
Ο μέσος χρόνος εκτέλεσης ήταν 0,436 ms και το ερώτημα υλοποιήθηκε ως εξής:

```
explain analyze
select *
from "Student" natural left join "Register"
where right(surname,1)<> 'Σ' and course_code='ΠΛΗ 101' and final_grade=7
```

	QUERY PLAN text
1	Nested Loop (cost=5.69..283.21 rows=2 width=198) (actual time=0.323..0.391 rows=2 loops=1)
2	Join Filter: ("Student".amka = "Register".amka)
3	Rows Removed by Join Filter: 256
4	-> Seq Scan on "Student" (cost=0.00..4.93 rows=109 width=165) (actual time=0.017..0.173 rows=86 loops=1)
5	Filter: ("right"((surname)::text, 1) <> 'Σ'::text)
6	Rows Removed by Filter: 24
7	-> Materialize (cost=5.69..275.02 rows=2 width=37) (actual time=0.001..0.002 rows=3 loops=86)
8	-> Bitmap Heap Scan on "Register" (cost=5.69..275.01 rows=2 width=37) (actual time=0.041..0.103 rows=3 loops=1)
9	Recheck Cond: (course_code = 'ΠΛΗ 101'::bpchar)
10	Filter: (final_grade = '7'::numeric)
11	Rows Removed by Filter: 161
12	Heap Blocks: exact=43
13	-> Bitmap Index Scan on "Register_pkey" (cost=0.00..5.69 rows=170 width=0) (actual time=0.033..0.033 rows=164 loops=1)
14	Index Cond: (course_code = 'ΠΛΗ 101'::bpchar)
15	Planning Time: 0.319 ms
16	Execution Time: 0.450 ms

Στη συνέχεια εκτελέσαμε το ίδιο query για διάφορα ευρετήρια. Προέκυψαν τα ακόλουθα αποτελέσματα:

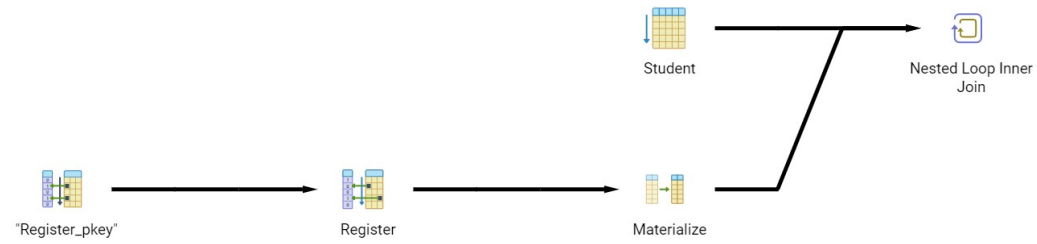
- **create index surname_index on "Student" (surname)**



	QUERY PLAN text
1	Nested Loop (cost=5.69..283.21 rows=2 width=198) (actual time=0.238..0.267 rows=2 loops=1)
2	Join Filter: ("Student".amka = "Register".amka)
3	Rows Removed by Join Filter: 256
4	-> Seq Scan on "Student" (cost=0.00..4.93 rows=109 width=165) (actual time=0.019..0.089 rows=86 loops=1)
5	Filter: ("right"((surname)::text, 1) <> 'Σ'::text)
6	Rows Removed by Filter: 24
7	-> Materialize (cost=5.69..275.02 rows=2 width=37) (actual time=0.001..0.002 rows=3 loops=86)
8	-> Bitmap Heap Scan on "Register" (cost=5.69..275.01 rows=2 width=37) (actual time=0.042..0.122 rows=3 loops=1)
9	Recheck Cond: (course_code = 'ΠΛΗ 101'::bpchar)
10	Filter: (final_grade = '7'::numeric)
11	Rows Removed by Filter: 161
12	Heap Blocks: exact=43
13	-> Bitmap Index Scan on "Register_pkey" (cost=0.00..5.69 rows=170 width=0) (actual time=0.034..0.034 rows=164 loops=1)
14	Index Cond: (course_code = 'ΠΛΗ 101'::bpchar)
15	Planning Time: 0.261 ms
16	Execution Time: 0.321 ms

Παρατηρούμε ότι ο μέσος χρόνος εκτέλεσης μειώνεται για περίπου 100 ms.

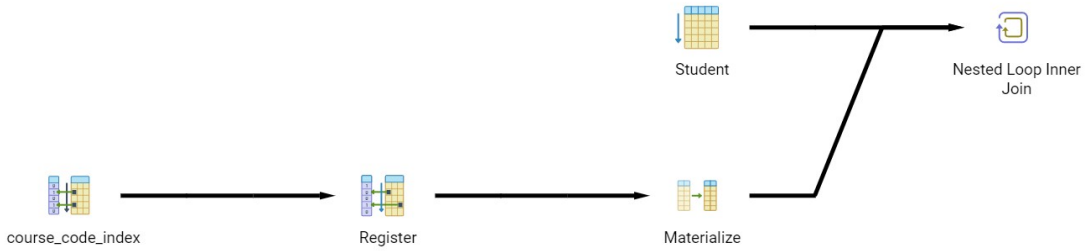
- **create index hash_surname_index on "Student" using hash(surname)**



	QUERY PLAN text
1	Nested Loop (cost=5.69..283.21 rows=2 width=198) (actual time=0.234..0.264 rows=2 loops=1)
2	Join Filter: ("Student".amka = "Register".amka)
3	Rows Removed by Join Filter: 256
4	-> Seq Scan on "Student" (cost=0.00..4.93 rows=109 width=165) (actual time=0.017..0.085 rows=86 loops=1)
5	Filter: ("right"((surname)::text, 1) <> 'Σ'::text)
6	Rows Removed by Filter: 24
7	-> Materialize (cost=5.69..275.02 rows=2 width=37) (actual time=0.001..0.002 rows=3 loops=86)
8	-> Bitmap Heap Scan on "Register" (cost=5.69..275.01 rows=2 width=37) (actual time=0.059..0.123 rows=3 loops=1)
9	Recheck Cond: (course_code = 'ΠΛΗ 101'::bpchar)
10	Filter: (final_grade = '7'::numeric)
11	Rows Removed by Filter: 161
12	Heap Blocks: exact=43
13	-> Bitmap Index Scan on "Register_pkey" (cost=0.00..5.69 rows=170 width=0) (actual time=0.051..0.051 rows=164 loops=1)
14	Index Cond: (course_code = 'ΠΛΗ 101'::bpchar)
15	Planning Time: 0.286 ms
16	Execution Time: 0.306 ms

Παρατηρούμε ότι ο μέσος χρόνος εκτέλεσης σε σχέση με το αρχικό query μειώνεται και πάλι για περίπου 100 ms και είναι λίγο μικρότερος από τον χρόνο εκτέλεσης του query με b+tree index στο surname.

- **create index course_code_index on "Register" (course_code)**



	QUERY PLAN text
1	Nested Loop (cost=5.57..283.09 rows=2 width=198) (actual time=0.252..0.281 rows=2 loops=1)
2	Join Filter: ("Student".amka = "Register".amka)
3	Rows Removed by Join Filter: 256
4	-> Seq Scan on "Student" (cost=0.00..4.93 rows=109 width=165) (actual time=0.017..0.086 rows=86 loops=1)
5	Filter: ("right"((surname)::text, 1) <> 'Σ'::text)
6	Rows Removed by Filter: 24
7	-> Materialize (cost=5.57..274.90 rows=2 width=37) (actual time=0.001..0.002 rows=3 loops=86)
8	-> Bitmap Heap Scan on "Register" (cost=5.57..274.89 rows=2 width=37) (actual time=0.063..0.138 rows=3 loops=1)
9	Recheck Cond: (course_code = 'ΠΛΗ 101'::bpchar)
10	Filter: (final_grade = '7'::numeric)
11	Rows Removed by Filter: 161
12	Heap Blocks: exact=43
13	-> Bitmap Index Scan on course_code_index (cost=0.00..5.57 rows=170 width=0) (actual time=0.052..0.052 rows=164 loops=1)
14	Index Cond: (course_code = 'ΠΛΗ 101'::bpchar)
15	Planning Time: 0.280 ms
16	Execution Time: 0.319 ms

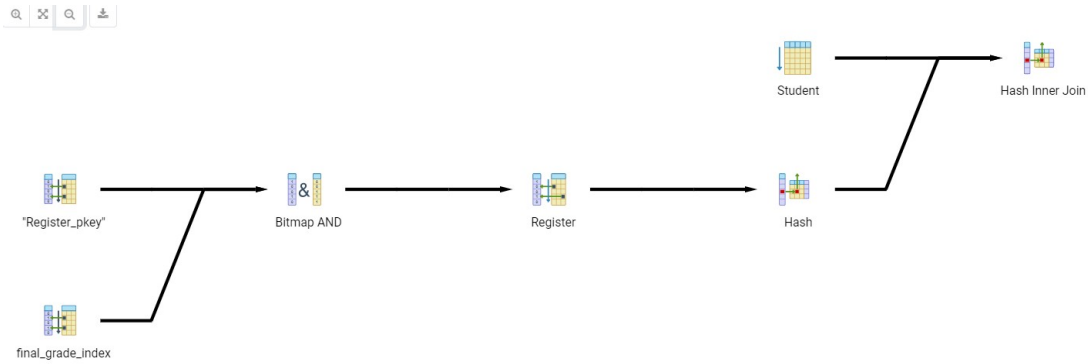
Παρατηρούμε ότι ο μέσος χρόνος εκτέλεσης σε σχέση με το αρχικό query μειώνεται ξανά για περίπου 100 ms.

- **create index hash_course_code_index on "Register" using hash(course_code)**

	QUERY PLAN text
1	Nested Loop (cost=5.69..283.21 rows=2 width=198) (actual time=0.263..0.293 rows=2 loops=1)
2	Join Filter: ("Student".amka = "Register".amka)
3	Rows Removed by Join Filter: 256
4	-> Seq Scan on "Student" (cost=0.00..4.93 rows=109 width=165) (actual time=0.022..0.092 rows=86 loops=1)
5	Filter: ("right"((surname)::text, 1) <> 'Σ'::text)
6	Rows Removed by Filter: 24
7	-> Materialize (cost=5.69..275.02 rows=2 width=37) (actual time=0.001..0.002 rows=3 loops=86)
8	-> Bitmap Heap Scan on "Register" (cost=5.69..275.01 rows=2 width=37) (actual time=0.053..0.143 rows=3 loops=1)
9	Recheck Cond: (course_code = 'ΠΛΗ 101'::bpchar)
10	Filter: (final_grade = '7'::numeric)
11	Rows Removed by Filter: 161
12	Heap Blocks: exact=43
13	-> Bitmap Index Scan on "Register_pkey" (cost=0.00..5.69 rows=170 width=0) (actual time=0.044..0.044 rows=164 loops=1)
14	Index Cond: (course_code = 'ΠΛΗ 101'::bpchar)
15	Planning Time: 0.425 ms
16	Execution Time: 0.345 ms

Παρατηρούμε ότι ο μέσος χρόνος εκτέλεσης σε σχέση με το αρχικό query μειώνεται ξανά για περίπου 100 ms, όμως το hash index στο course_code δίνει μέχρι στιγμής τα χειρότερα αποτελέσματα από τα υπόλοιπα ευρετήρια.

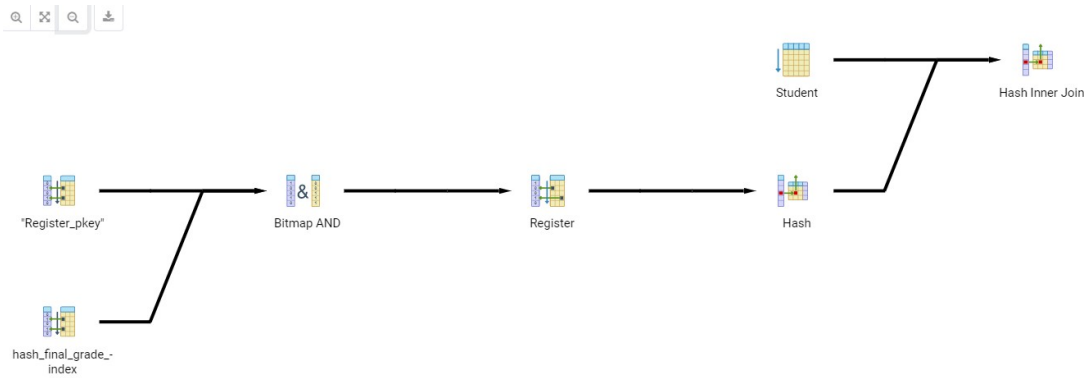
- **create index final_grade_index on "Register" (final_grade)**



	QUERY PLAN text
1	Hash Join (cost=24.89..30.38 rows=2 width=198) (actual time=0.195..0.212 rows=2 loops=1)
2	Hash Cond: ("Student".amka = "Register".amka)
3	-> Seq Scan on "Student" (cost=0.00..4.93 rows=109 width=165) (actual time=0.016..0.085 rows=86 loops=1)
4	Filter: ("right"((surname)::text, 1) <> 'Σ'::text)
5	Rows Removed by Filter: 24
6	-> Hash (cost=24.87..24.87 rows=2 width=37) (actual time=0.108..0.109 rows=3 loops=1)
7	Buckets: 1024 Batches: 1 Memory Usage: 9kB
8	-> Bitmap Heap Scan on "Register" (cost=17.33..24.87 rows=2 width=37) (actual time=0.095..0.099 rows=3 loops=1)
9	Recheck Cond: ((course_code = 'ΠΛΗ 101'::bpchar) AND (final_grade = '7'::numeric))
10	Heap Blocks: exact=3
11	-> BitmapAnd (cost=17.33..17.33 rows=2 width=0) (actual time=0.092..0.092 rows=0 loops=1)
12	-> Bitmap Index Scan on "Register_pkey" (cost=0.00..5.69 rows=170 width=0) (actual time=0.040..0.040 rows=164 loops=1)
13	Index Cond: (course_code = 'ΠΛΗ 101'::bpchar)
14	-> Bitmap Index Scan on final_grade_index (cost=0.00..11.39 rows=413 width=0) (actual time=0.048..0.048 rows=409 loops=1)
15	Index Cond: (final_grade = '7'::numeric)
16	Planning Time: 0.277 ms
17	Execution Time: 0.262 ms

Παρατηρούμε ότι ο μέσος χρόνος εκτέλεσης σε σχέση με το αρχικό query μειώνεται για περίπου 200 ms, επομένως το b+tree index στο final_grade μας δίνει τον καλύτερο χρόνο εκτέλεσης μέχρι στιγμής.

- **create index hash_final_grade_index on "Register" using hash (final_grade)**



	QUERY PLAN text
1	Nested Loop (cost=5.69..283.21 rows=2 width=198) (actual time=0.234..0.264 rows=2 loops=1)
2	Join Filter: ("Student".amka = "Register".amka)
3	Rows Removed by Join Filter: 256
4	-> Seq Scan on "Student" (cost=0.00..4.93 rows=109 width=165) (actual time=0.017..0.085 rows=86 loops=1)
5	Filter: ("right"((surname)::text, 1) <> 'Σ'::text)
6	Rows Removed by Filter: 24
7	-> Materialize (cost=5.69..275.02 rows=2 width=37) (actual time=0.001..0.002 rows=3 loops=86)
8	-> Bitmap Heap Scan on "Register" (cost=5.69..275.01 rows=2 width=37) (actual time=0.059..0.123 rows=3 loops=1)
9	Recheck Cond: (course_code = 'ΠΛΗ 101'::bpchar)
10	Filter: (final_grade = '7'::numeric)
11	Rows Removed by Filter: 161
12	Heap Blocks: exact=43
13	-> Bitmap Index Scan on "Register_pkey" (cost=0.00..5.69 rows=170 width=0) (actual time=0.051..0.051 rows=164 loops=1)
14	Index Cond: (course_code = 'ΠΛΗ 101'::bpchar)
15	Planning Time: 0.286 ms
16	Execution Time: 0.306 ms

Παρατηρούμε ότι ο μέσος χρόνος εκτέλεσης σε σχέση με το αρχικό query μειώνεται για περίπου 100 ms.

- **cluster "Student" using surname_index;**

	QUERY PLAN text
1	Nested Loop (cost=5.69..283.21 rows=2 width=198) (actual time=0.133..0.244 rows=2 loops=1)
2	Join Filter: ("Student".amka = "Register".amka)
3	Rows Removed by Join Filter: 256
4	-> Seq Scan on "Student" (cost=0.00..4.93 rows=109 width=165) (actual time=0.015..0.083 rows=86 loops=1)
5	Filter: ("right"((surname)::text, 1) <> 'Σ'::text)
6	Rows Removed by Filter: 24
7	-> Materialize (cost=5.69..275.02 rows=2 width=37) (actual time=0.001..0.001 rows=3 loops=86)
8	-> Bitmap Heap Scan on "Register" (cost=5.69..275.01 rows=2 width=37) (actual time=0.041..0.105 rows=3 loops=1)
9	Recheck Cond: (course_code = 'ΠΛΗ 101'::bpchar)
10	Filter: (final_grade = '7'::numeric)
11	Rows Removed by Filter: 161
12	Heap Blocks: exact=43
13	-> Bitmap Index Scan on "Register_pkey" (cost=0.00..5.69 rows=170 width=0) (actual time=0.033..0.033 rows=164 loops=1)
14	Index Cond: (course_code = 'ΠΛΗ 101'::bpchar)
15	Planning Time: 1.270 ms
16	Execution Time: 0.279 ms

Παρατηρούμε ότι ο μέσος χρόνος εκτέλεσης σε σχέση με το αρχικό query μειώνεται για περίπου 200 ms, ενώ σε σχέση με το b+tree index στο surname μειώνεται κατά 40 ms περίπου.

- **cluster "Register" using course_code_index;**

	QUERY PLAN text
1	Nested Loop (cost=5.57..283.09 rows=2 width=198) (actual time=0.133..0.243 rows=2 loops=1)
2	Join Filter: ("Student".amka = "Register".amka)
3	Rows Removed by Join Filter: 256
4	-> Seq Scan on "Student" (cost=0.00..4.93 rows=109 width=165) (actual time=0.015..0.085 rows=86 loops=1)
5	Filter: ("right"((surname)::text, 1) <> 'Σ'::text)
6	Rows Removed by Filter: 24
7	-> Materialize (cost=5.57..274.90 rows=2 width=37) (actual time=0.001..0.001 rows=3 loops=86)
8	-> Bitmap Heap Scan on "Register" (cost=5.57..274.89 rows=2 width=37) (actual time=0.066..0.096 rows=3 loops=1)
9	Recheck Cond: (course_code = 'ΠΛΗ 101'::bpchar)
10	Filter: (final_grade = '7'::numeric)
11	Rows Removed by Filter: 161
12	Heap Blocks: exact=3
13	-> Bitmap Index Scan on course_code_index (cost=0.00..5.57 rows=170 width=0) (actual time=0.061..0.061 rows=164 loops=1)
14	Index Cond: (course_code = 'ΠΛΗ 101'::bpchar)
15	Planning Time: 0.629 ms
16	Execution Time: 0.282 ms

Παρατηρούμε ότι ο μέσος χρόνος εκτέλεσης σε σχέση με το αρχικό query μειώνεται για περίπου 200 ms, ενώ σε σχέση με το b+tree index στο course_code μειώνεται κατά 40 ms περίπου.

- **cluster "Register" using final_grade_index;**

	QUERY PLAN text
1	Hash Join (cost=24.89..30.38 rows=2 width=198) (actual time=0.206..0.274 rows=2 loops=1)
2	Hash Cond: ("Student".amka = "Register".amka)
3	-> Seq Scan on "Student" (cost=0.00..4.93 rows=109 width=165) (actual time=0.015..0.083 rows=86 loops=1)
4	Filter: ("right"((surname)::text, 1) <> 'Σ'::text)
5	Rows Removed by Filter: 24
6	-> Hash (cost=24.87..24.87 rows=2 width=37) (actual time=0.165..0.165 rows=3 loops=1)
7	Buckets: 1024 Batches: 1 Memory Usage: 9kB
8	-> Bitmap Heap Scan on "Register" (cost=17.33..24.87 rows=2 width=37) (actual time=0.159..0.160 rows=3 loops=1)
9	Recheck Cond: ((course_code = 'ΠΛΗ 101'::bpchar) AND (final_grade = '7'::numeric))
10	Heap Blocks: exact=1
11	-> BitmapAnd (cost=17.33..17.33 rows=2 width=0) (actual time=0.156..0.156 rows=0 loops=1)
12	-> Bitmap Index Scan on "Register_pkey" (cost=0.00..5.69 rows=170 width=0) (actual time=0.084..0.084 rows=164 loops=1)
13	Index Cond: (course_code = 'ΠΛΗ 101'::bpchar)
14	-> Bitmap Index Scan on final_grade_index (cost=0.00..11.39 rows=413 width=0) (actual time=0.068..0.068 rows=409 loops=1)
15	Index Cond: (final_grade = '7'::numeric)
16	Planning Time: 0.637 ms
17	Execution Time: 0.314 ms

Παρατηρούμε ότι ο μέσος χρόνος εκτέλεσης σε σχέση με το αρχικό query μειώνεται για περίπου 100 ms, ενώ σε σχέση με το b+tree index στο final_grade αυξάνεται κατά 50 ms περίπου.

Στη συνέχεια χρειάστηκε να επαναλάβουμε όλα τα προηγούμενα βήματα για μερικές χιλιάδες φοιτητών. Προέκυψαν τα ακόλουθα αποτελέσματα:

- **Χωρίς ευρετήρια**

	QUERY PLAN text
1	Nested Loop (cost=5.35..244.74 rows=1 width=198) (actual time=0.094..0.289 rows=2 loops=1)
2	Join Filter: ("Student".amka = "Register".amka)
3	Rows Removed by Join Filter: 222
4	-> Bitmap Heap Scan on "Register" (cost=5.35..238.45 rows=1 width=37) (actual time=0.035..0.099 rows=3 loops=1)
5	Recheck Cond: (course_code = 'ΠΛΗ 101'::bpchar)
6	Filter: (final_grade = '7'::numeric)
7	Rows Removed by Filter: 161
8	Heap Blocks: exact=43
9	-> Bitmap Index Scan on "Register_pkey" (cost=0.00..5.34 rows=124 width=0) (actual time=0.029..0.029 rows=164 loops=1)
10	Index Cond: (course_code = 'ΠΛΗ 101'::bpchar)
11	-> Seq Scan on "Student" (cost=0.00..4.93 rows=109 width=165) (actual time=0.004..0.057 rows=75 loops=3)
12	Filter: ("right"((surname)::text, 1) <> 'Σ'::text)
13	Rows Removed by Filter: 22
14	Planning Time: 0.206 ms
15	Execution Time: 0.312 ms

➢ Με b+tree index στο surname

	QUERY PLAN text
1	Merge Join (cost=238.75..239.02 rows=1 width=199) (actual time=0.165..0.175 rows=2 loops=1)
2	Merge Cond: ("Student".amka = "Register".amka)
3	-> Index Scan using "Student_pkey" on "Student" (cost=0.29..3762.51 rows=29661 width=166) (actual time=0.007..0.066 rows=76 loops=1)
4	Filter: ("right"((surname)::text, 1) <> 'Σ'::text)
5	Rows Removed by Filter: 22
6	-> Sort (cost=238.46..238.46 rows=1 width=37) (actual time=0.101..0.101 rows=3 loops=1)
7	Sort Key: "Register".amka
8	Sort Method: quicksort Memory: 25kB
9	-> Bitmap Heap Scan on "Register" (cost=5.35..238.45 rows=1 width=37) (actual time=0.033..0.096 rows=3 loops=1)
10	Recheck Cond: (course_code = 'ΠΛΗ 101'::bpchar)
11	Filter: (final_grade = '7'::numeric)
12	Rows Removed by Filter: 161
13	Heap Blocks: exact=43
14	-> Bitmap Index Scan on "Register_pkey" (cost=0.00..5.34 rows=124 width=0) (actual time=0.027..0.027 rows=164 loops=1)
15	Index Cond: (course_code = 'ΠΛΗ 101'::bpchar)
16	Planning Time: 0.204 ms
17	Execution Time: 0.201 ms

➢ Με hash index στο surname

	QUERY PLAN text
1	Merge Join (cost=238.75..239.02 rows=1 width=199) (actual time=0.159..0.170 rows=2 loops=1)
2	Merge Cond: ("Student".amka = "Register".amka)
3	-> Index Scan using "Student_pkey" on "Student" (cost=0.29..3762.51 rows=29661 width=166) (actual time=0.007..0.064 rows=76 loops=1)
4	Filter: ("right"((surname)::text, 1) <> 'Σ'::text)
5	Rows Removed by Filter: 22
6	-> Sort (cost=238.46..238.46 rows=1 width=37) (actual time=0.097..0.097 rows=3 loops=1)
7	Sort Key: "Register".amka
8	Sort Method: quicksort Memory: 25kB
9	-> Bitmap Heap Scan on "Register" (cost=5.35..238.45 rows=1 width=37) (actual time=0.031..0.092 rows=3 loops=1)
10	Recheck Cond: (course_code = 'ΠΛΗ 101'::bpchar)
11	Filter: (final_grade = '7'::numeric)
12	Rows Removed by Filter: 161
13	Heap Blocks: exact=43
14	-> Bitmap Index Scan on "Register_pkey" (cost=0.00..5.34 rows=124 width=0) (actual time=0.025..0.025 rows=164 loops=1)
15	Index Cond: (course_code = 'ΠΛΗ 101'::bpchar)
16	Planning Time: 0.189 ms
17	Execution Time: 0.195 ms

➢ Με b+tree index στο course_code

	QUERY PLAN text
1	Merge Join (cost=238.62..238.90 rows=1 width=199) (actual time=0.163..0.173 rows=2 loops=1)
2	Merge Cond: ("Student".amka = "Register".amka)
3	-> Index Scan using "Student_pkey" on "Student" (cost=0.29..3762.51 rows=29661 width=166) (actual time=0.007..0.064 rows=76 loops=1)
4	Filter: ("right"((surname)::text, 1) <> 'Σ'::text)
5	Rows Removed by Filter: 22
6	-> Sort (cost=238.33..238.34 rows=1 width=37) (actual time=0.100..0.101 rows=3 loops=1)
7	Sort Key: "Register".amka
8	Sort Method: quicksort Memory: 25kB
9	-> Bitmap Heap Scan on "Register" (cost=5.22..238.32 rows=1 width=37) (actual time=0.031..0.096 rows=3 loops=1)
10	Recheck Cond: (course_code = 'ΠΛΗ 101'::bpchar)
11	Filter: (final_grade = '7'::numeric)
12	Rows Removed by Filter: 161
13	Heap Blocks: exact=43
14	-> Bitmap Index Scan on course_code_index (cost=0.00..5.22 rows=124 width=0) (actual time=0.026..0.026 rows=164 loops=1)
15	Index Cond: (course_code = 'ΠΛΗ 101'::bpchar)
16	Planning Time: 0.211 ms
17	Execution Time: 0.200 ms

➢ Με hash index στο course_code

	QUERY PLAN text
1	Merge Join (cost=238.33..238.61 rows=1 width=199) (actual time=0.170..0.181 rows=2 loops=1)
2	Merge Cond: ("Student".amka = "Register".amka)
3	-> Index Scan using "Student_pkey" on "Student" (cost=0.29..3762.51 rows=29661 width=166) (actual time=0.006..0.070 rows=76 loops=1)
4	Filter: ("right"((surname)::text, 1) <> 'Σ'::text)
5	Rows Removed by Filter: 22
6	-> Sort (cost=238.04..238.05 rows=1 width=37) (actual time=0.103..0.103 rows=3 loops=1)
7	Sort Key: "Register".amka
8	Sort Method: quicksort Memory: 25kB
9	-> Bitmap Heap Scan on "Register" (cost=4.93..238.03 rows=1 width=37) (actual time=0.021..0.098 rows=3 loops=1)
10	Recheck Cond: (course_code = 'ΠΛΗ 101'::bpchar)
11	Filter: (final_grade = '7'::numeric)
12	Rows Removed by Filter: 161
13	Heap Blocks: exact=43
14	-> Bitmap Index Scan on hash_course_code_index (cost=0.00..4.93 rows=124 width=0) (actual time=0.015..0.015 rows=164 loops=1)
15	Index Cond: (course_code = 'ΠΛΗ 101'::bpchar)
16	Planning Time: 1.290 ms
17	Execution Time: 0.210 ms

➢ Με b+tree index στο final_grade

	QUERY PLAN text
1	Merge Join (cost=21.17..21.44 rows=1 width=199) (actual time=0.164..0.175 rows=2 loops=1)
2	Merge Cond: ("Student".amka = "Register".amka)
3	-> Index Scan using "Student_pkey" on "Student" (cost=0.29..3762.51 rows=29661 width=166) (actual time=0.007..0.066 rows=76 loops=1)
4	Filter: ("right"((surname)::text, 1) <> 'Σ'::text)
5	Rows Removed by Filter: 22
6	-> Sort (cost=20.88..20.89 rows=1 width=37) (actual time=0.100..0.100 rows=3 loops=1)
7	Sort Key: "Register".amka
8	Sort Method: quicksort Memory: 25kB
9	-> Bitmap Heap Scan on "Register" (cost=16.86..20.87 rows=1 width=37) (actual time=0.091..0.093 rows=3 loops=1)
10	Recheck Cond: ((course_code = 'ΠΛΗ 101'::bpchar) AND (final_grade = '7'::numeric))
11	Heap Blocks: exact=3
12	-> BitmapAnd (cost=16.86..16.86 rows=1 width=0) (actual time=0.088..0.088 rows=0 loops=1)
13	-> Bitmap Index Scan on "Register_pkey" (cost=0.00..5.34 rows=124 width=0) (actual time=0.030..0.030 rows=164 loops=1)
14	Index Cond: (course_code = 'ΠΛΗ 101'::bpchar)
15	-> Bitmap Index Scan on final_grade_index (cost=0.00..11.26 rows=396 width=0) (actual time=0.055..0.055 rows=409 loops=1)
16	Index Cond: (final_grade = '7'::numeric)
17	Planning Time: 0.907 ms
18	Execution Time: 0.209 ms

➢ Με hash index στο final_grade

	QUERY PLAN text
1	Merge Join (cost=24.88..25.15 rows=1 width=199) (actual time=0.119..0.130 rows=2 loops=1)
2	Merge Cond: ("Student".amka = "Register".amka)
3	-> Index Scan using "Student_pkey" on "Student" (cost=0.29..3762.51 rows=29661 width=166) (actual time=0.007..0.065 rows=76 loops=1)
4	Filter: ("right"((surname)::text, 1) <> 'Σ'::text)
5	Rows Removed by Filter: 22
6	-> Sort (cost=24.59..24.60 rows=1 width=37) (actual time=0.058..0.058 rows=3 loops=1)
7	Sort Key: "Register".amka
8	Sort Method: quicksort Memory: 25kB
9	-> Bitmap Heap Scan on "Register" (cost=20.57..24.58 rows=1 width=37) (actual time=0.051..0.053 rows=3 loops=1)
10	Recheck Cond: ((course_code = 'ΠΛΗ 101'::bpchar) AND (final_grade = '7'::numeric))
11	Heap Blocks: exact=3
12	-> BitmapAnd (cost=20.57..20.57 rows=1 width=0) (actual time=0.048..0.048 rows=0 loops=1)
13	-> Bitmap Index Scan on "Register_pkey" (cost=0.00..5.34 rows=124 width=0) (actual time=0.027..0.027 rows=164 loops=1)
14	Index Cond: (course_code = 'ΠΛΗ 101'::bpchar)
15	-> Bitmap Index Scan on hash_final_grade_index (cost=0.00..14.97 rows=396 width=0) (actual time=0.017..0.017 rows=409 loops=1)
16	Index Cond: (final_grade = '7'::numeric)
17	Planning Time: 0.879 ms
18	Execution Time: 0.157 ms

- **Με cluster "Student" using surname_index;**

	QUERY PLAN text
1	Merge Join (cost=238.75..238.99 rows=1 width=199) (actual time=0.206..0.224 rows=2 loops=1)
2	Merge Cond: ("Student".amka = "Register".amka)
3	-> Index Scan using "Student_pkey" on "Student" (cost=0.29..3225.79 rows=29661 width=166) (actual time=0.009..0.120 rows=76 loops=1)
4	Filter: ("right"((surname)::text, 1) <> 'Σ'::text)
5	Rows Removed by Filter: 22
6	-> Sort (cost=238.46..238.46 rows=1 width=37) (actual time=0.097..0.097 rows=3 loops=1)
7	Sort Key: "Register".amka
8	Sort Method: quicksort Memory: 25kB
9	-> Bitmap Heap Scan on "Register" (cost=5.35..238.45 rows=1 width=37) (actual time=0.033..0.090 rows=3 loops=1)
10	Recheck Cond: (course_code = 'ΠΛΗ 101'::bpchar)
11	Filter: (final_grade = '7'::numeric)
12	Rows Removed by Filter: 161
13	Heap Blocks: exact=43
14	-> Bitmap Index Scan on "Register_pkey" (cost=0.00..5.34 rows=124 width=0) (actual time=0.027..0.027 rows=164 loops=1)
15	Index Cond: (course_code = 'ΠΛΗ 101'::bpchar)
16	Planning Time: 1.153 ms
17	Execution Time: 0.248 ms

- **Με cluster "Register" using course_code_index;**

	QUERY PLAN text
1	Merge Join (cost=238.62..238.86 rows=1 width=199) (actual time=0.323..0.348 rows=2 loops=1)
2	Merge Cond: ("Student".amka = "Register".amka)
3	-> Index Scan using "Student_pkey" on "Student" (cost=0.29..3225.79 rows=29661 width=166) (actual time=0.011..0.234 rows=76 loops=1)
4	Filter: ("right"((surname)::text, 1) <> 'Σ'::text)
5	Rows Removed by Filter: 22
6	-> Sort (cost=238.33..238.34 rows=1 width=37) (actual time=0.102..0.103 rows=3 loops=1)
7	Sort Key: "Register".amka
8	Sort Method: quicksort Memory: 25kB
9	-> Bitmap Heap Scan on "Register" (cost=5.22..238.32 rows=1 width=37) (actual time=0.057..0.097 rows=3 loops=1)
10	Recheck Cond: (course_code = 'ΠΛΗ 101'::bpchar)
11	Filter: (final_grade = '7'::numeric)
12	Rows Removed by Filter: 161
13	Heap Blocks: exact=3
14	-> Bitmap Index Scan on course_code_index (cost=0.00..5.22 rows=124 width=0) (actual time=0.052..0.052 rows=164 loops=1)
15	Index Cond: (course_code = 'ΠΛΗ 101'::bpchar)
16	Planning Time: 1.100 ms
17	Execution Time: 0.381 ms

- **Με cluster "Register" using final_grade_index;**

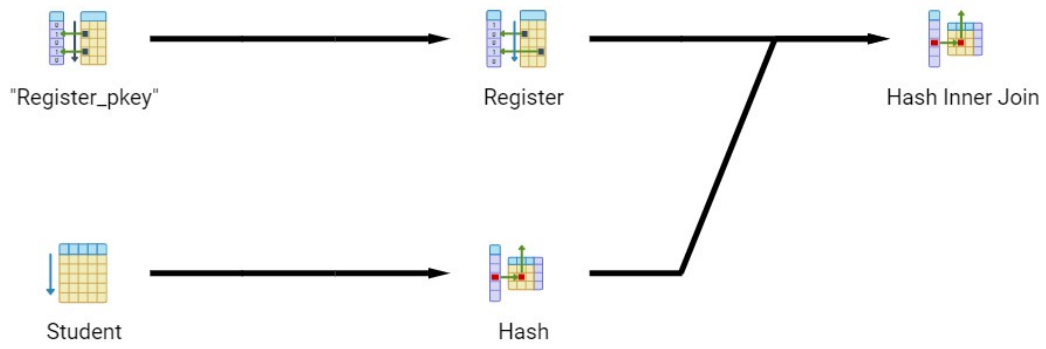
	QUERY PLAN text
1	Merge Join (cost=21.17..21.41 rows=1 width=199) (actual time=0.244..0.264 rows=2 loops=1)
2	Merge Cond: ("Student".amka = "Register".amka)
3	-> Index Scan using "Student_pkey" on "Student" (cost=0.29..3225.79 rows=29661 width=166) (actual time=0.009..0.132 rows=76 loops=1)
4	Filter: ("right"((surname)::text, 1) <> 'Σ'::text)
5	Rows Removed by Filter: 22
6	-> Sort (cost=20.88..20.89 rows=1 width=37) (actual time=0.124..0.124 rows=3 loops=1)
7	Sort Key: "Register".amka
8	Sort Method: quicksort Memory: 25kB
9	-> Bitmap Heap Scan on "Register" (cost=16.86..20.87 rows=1 width=37) (actual time=0.119..0.119 rows=3 loops=1)
10	Recheck Cond: ((course_code = 'ΠΛΗ 101'::bpchar) AND (final_grade = '7'::numeric))
11	Heap Blocks: exact=1
12	-> BitmapAnd (cost=16.86..16.86 rows=1 width=0) (actual time=0.116..0.116 rows=0 loops=1)
13	-> Bitmap Index Scan on "Register_pkey" (cost=0.00..5.34 rows=124 width=0) (actual time=0.053..0.053 rows=164 loops=1)
14	Index Cond: (course_code = 'ΠΛΗ 101'::bpchar)
15	-> Bitmap Index Scan on final_grade_index (cost=0.00..11.26 rows=396 width=0) (actual time=0.060..0.060 rows=409 loops=1)
16	Index Cond: (final_grade = '7'::numeric)
17	Planning Time: 0.931 ms
18	Execution Time: 0.290 ms

Συμπεράσματα

Συμπεραίνουμε ότι για κάποιες εκατοντάδες φοιτητών τους καλύτερους μέσους χρόνους εκτέλεσης τους παίρνουμε για b+tree index στο final_grade, cluster "Student" using surname_index και cluster "Register" using course_code_index.

Στην περίπτωση των χιλιάδων φοιτητών, οι καλύτεροι μέσοι χρόνοι εκτέλεσης προέκυψαν για hash index στο final_grade, hash index στο surname και b+tree index στο course_code, με το hash index στο final_grade να είναι το πιο αποδοτικό.

Στη συνέχεια δοκιμάσαμε να απενεργοποιήσουμε τους αλγόριθμους υπολογισμούς συνδέσεων που χρησιμοποιούνται.
Πρώτα, απενεργοποιήσαμε το Nested Loop Inner Join, με την εντολή
set enable_nestloop=off;



	QUERY PLAN text
1	Hash Join (cost=11.98..280.87 rows=2 width=198) (actual time=0.187..0.219 rows=2 loops=1)
2	Hash Cond: ("Register".amka = "Student".amka)
3	-> Bitmap Heap Scan on "Register" (cost=5.69..274.57 rows=2 width=37) (actual time=0.084..0.115 rows=3 loops=1)
4	Recheck Cond: (course_code = 'ΠΛΗ 101'::bpchar)
5	Filter: (final_grade = '7'::numeric)
6	Rows Removed by Filter: 161
7	Heap Blocks: exact=41
8	-> Bitmap Index Scan on "Register_pkey" (cost=0.00..5.69 rows=170 width=0) (actual time=0.031..0.031 rows=164 loops=1)
9	Index Cond: (course_code = 'ΠΛΗ 101'::bpchar)
10	-> Hash (cost=4.93..4.93 rows=109 width=165) (actual time=0.097..0.097 rows=86 loops=1)
11	Buckets: 1024 Batches: 1 Memory Usage: 25kB
12	-> Seq Scan on "Student" (cost=0.00..4.93 rows=109 width=165) (actual time=0.015..0.084 rows=86 loops=1)
13	Filter: ("right"((surname)::text, 1) <> 'Σ'::text)
14	Rows Removed by Filter: 24
15	Planning Time: 0.262 ms
16	Execution Time: 0.260 ms

Παρατηρούμε ότι ο μέσος χρόνος εκτέλεσης του ερωτήματος σε σχέση με το αρχικό query βελτιώθηκε για 200 ms περίπου.

- **Με b+tree index στο surname**

	QUERY PLAN text
1	Hash Join (cost=15.99..434.51 rows=1 width=198) (actual time=0.193..0.273 rows=2 loops=1)
2	Hash Cond: ("Register".amka = "Student".amka)
3	-> Bitmap Heap Scan on "Register" (cost=9.70..428.22 rows=1 width=37) (actual time=0.054..0.133 rows=3 loops=1)
4	Recheck Cond: (course_code = 'ΠΛΗ 101'::bpchar)
5	Filter: (final_grade = '7'::numeric)
6	Rows Removed by Filter: 161
7	Heap Blocks: exact=52
8	-> Bitmap Index Scan on "Register_pkey" (cost=0.00..9.70 rows=171 width=0) (actual time=0.043..0.044 rows=164 loops=1)
9	Index Cond: (course_code = 'ΠΛΗ 101'::bpchar)
10	-> Hash (cost=4.93..4.93 rows=109 width=165) (actual time=0.131..0.132 rows=86 loops=1)
11	Buckets: 1024 Batches: 1 Memory Usage: 25kB
12	-> Seq Scan on "Student" (cost=0.00..4.93 rows=109 width=165) (actual time=0.021..0.112 rows=86 loops=1)
13	Filter: ("right"((surname)::text, 1) <> 'Σ'::text)
14	Rows Removed by Filter: 24
15	Planning Time: 0.346 ms

- **Με hash index στο surname**

	QUERY PLAN text
1	Hash Join (cost=15.99..434.51 rows=1 width=198) (actual time=0.197..0.278 rows=2 loops=1)
2	Hash Cond: ("Register".amka = "Student".amka)
3	-> Bitmap Heap Scan on "Register" (cost=9.70..428.22 rows=1 width=37) (actual time=0.055..0.135 rows=3 loops=1)
4	Recheck Cond: (course_code = 'ΠΛΗ 101'::bpchar)
5	Filter: (final_grade = '7'::numeric)
6	Rows Removed by Filter: 161
7	Heap Blocks: exact=52
8	-> Bitmap Index Scan on "Register_pkey" (cost=0.00..9.70 rows=171 width=0) (actual time=0.044..0.044 rows=164 loops=1)
9	Index Cond: (course_code = 'ΠΛΗ 101'::bpchar)
10	-> Hash (cost=4.93..4.93 rows=109 width=165) (actual time=0.135..0.135 rows=86 loops=1)
11	Buckets: 1024 Batches: 1 Memory Usage: 25kB
12	-> Seq Scan on "Student" (cost=0.00..4.93 rows=109 width=165) (actual time=0.022..0.115 rows=86 loops=1)
13	Filter: ("right"((surname)::text, 1) <> 'Σ'::text)
14	Rows Removed by Filter: 24
15	Planning Time: 0.352 ms
16	Execution Time: 0.331 ms

- **Με b+tree index στο course_code**

	QUERY PLAN text
1	Hash Join (cost=11.86..430.39 rows=1 width=198) (actual time=0.191..0.280 rows=2 loops=1)
2	Hash Cond: ("Register".amka = "Student".amka)
3	-> Bitmap Heap Scan on "Register" (cost=5.57..424.10 rows=1 width=37) (actual time=0.052..0.140 rows=3 loops=1)
4	Recheck Cond: (course_code = 'ΠΛΗ 101'::bpchar)
5	Filter: (final_grade = '7'::numeric)
6	Rows Removed by Filter: 161
7	Heap Blocks: exact=52
8	-> Bitmap Index Scan on course_code_index (cost=0.00..5.57 rows=171 width=0) (actual time=0.041..0.041 rows=164 loops=1)
9	Index Cond: (course_code = 'ΠΛΗ 101'::bpchar)
10	-> Hash (cost=4.93..4.93 rows=109 width=165) (actual time=0.132..0.132 rows=86 loops=1)
11	Buckets: 1024 Batches: 1 Memory Usage: 25kB
12	-> Seq Scan on "Student" (cost=0.00..4.93 rows=109 width=165) (actual time=0.020..0.112 rows=86 loops=1)
13	Filter: ("right"((surname)::text, 1) <> 'Σ'::text)
14	Rows Removed by Filter: 24
15	Planning Time: 0.360 ms
16	Execution Time: 0.332 ms

- **Με hash index στο course_code**

	QUERY PLAN text
1	Hash Join (cost=11.99..294.22 rows=1 width=198) (actual time=0.153..0.179 rows=2 loops=1)
2	Hash Cond: ("Register".amka = "Student".amka)
3	-> Bitmap Heap Scan on "Register" (cost=5.70..287.93 rows=1 width=37) (actual time=0.038..0.063 rows=3 loops=1)
4	Recheck Cond: (course_code = 'ΠΛΗ 101'::bpchar)
5	Filter: (final_grade = '7'::numeric)
6	Rows Removed by Filter: 161
7	Heap Blocks: exact=3
8	-> Bitmap Index Scan on "Register_pkey" (cost=0.00..5.70 rows=171 width=0) (actual time=0.033..0.033 rows=164 loops=1)
9	Index Cond: (course_code = 'ΠΛΗ 101'::bpchar)
10	-> Hash (cost=4.93..4.93 rows=109 width=165) (actual time=0.102..0.102 rows=86 loops=1)
11	Buckets: 1024 Batches: 1 Memory Usage: 25kB
12	-> Seq Scan on "Student" (cost=0.00..4.93 rows=109 width=165) (actual time=0.017..0.087 rows=86 loops=1)
13	Filter: ("right"((surname)::text, 1) <> 'Σ'::text)
14	Rows Removed by Filter: 24
15	Planning Time: 0.270 ms
16	Execution Time: 0.217 ms

- **Με b+tree index στο final_grade**

	QUERY PLAN text
1	Hash Join (cost=14.87..20.22 rows=1 width=198) (actual time=0.112..0.207 rows=2 loops=1)
2	Hash Cond: ("Student".amka = "Register".amka)
3	-> Seq Scan on "Student" (cost=0.00..4.93 rows=109 width=165) (actual time=0.021..0.114 rows=86 loops=1)
4	Filter: ("right"((surname)::text, 1) <> 'Σ'::text)
5	Rows Removed by Filter: 24
6	-> Hash (cost=14.86..14.86 rows=1 width=37) (actual time=0.074..0.074 rows=3 loops=1)
7	Buckets: 1024 Batches: 1 Memory Usage: 9kB
8	-> Bitmap Heap Scan on "Register" (cost=10.85..14.86 rows=1 width=37) (actual time=0.067..0.069 rows=3 loops=1)
9	Recheck Cond: ((final_grade = '7'::numeric) AND (course_code = 'ΠΛΗ 101'::bpchar))
10	Heap Blocks: exact=1
11	-> BitmapAnd (cost=10.85..10.85 rows=1 width=0) (actual time=0.062..0.062 rows=0 loops=1)
12	-> Bitmap Index Scan on final_grade_index (cost=0.00..4.90 rows=81 width=0) (actual time=0.022..0.022 rows=72 loops=1)
13	Index Cond: (final_grade = '7'::numeric)
14	-> Bitmap Index Scan on "Register_pkey" (cost=0.00..5.70 rows=171 width=0) (actual time=0.037..0.037 rows=164 loops=1)
15	Index Cond: (course_code = 'ΠΛΗ 101'::bpchar)
16	Planning Time: 0.383 ms
17	Execution Time: 0.269 ms

➢ Με hash index στο final_grade

	QUERY PLAN text
1	Hash Join (cost=14.58..19.93 rows=1 width=198) (actual time=0.102..0.195 rows=2 loops=1)
2	Hash Cond: ("Student".amka = "Register".amka)
3	-> Seq Scan on "Student" (cost=0.00..4.93 rows=109 width=165) (actual time=0.019..0.110 rows=86 loops=1)
4	Filter: ("right"((surname)::text, 1) <> 'Σ'::text)
5	Rows Removed by Filter: 24
6	-> Hash (cost=14.57..14.57 rows=1 width=37) (actual time=0.068..0.068 rows=3 loops=1)
7	Buckets: 1024 Batches: 1 Memory Usage: 9kB
8	-> Bitmap Heap Scan on "Register" (cost=10.56..14.57 rows=1 width=37) (actual time=0.061..0.063 rows=3 loops=1)
9	Recheck Cond: ((final_grade = '7'::numeric) AND (course_code = 'ΠΛΗ 101'::bpchar))
10	Heap Blocks: exact=1
11	-> BitmapAnd (cost=10.56..10.56 rows=1 width=0) (actual time=0.055..0.055 rows=0 loops=1)
12	-> Bitmap Index Scan on hash_final_grade_index (cost=0.00..4.61 rows=81 width=0) (actual time=0.012..0.012 rows=72 loops=1)
13	Index Cond: (final_grade = '7'::numeric)
14	-> Bitmap Index Scan on "Register_pkey" (cost=0.00..5.70 rows=171 width=0) (actual time=0.041..0.041 rows=164 loops=1)
15	Index Cond: (course_code = 'ΠΛΗ 101'::bpchar)
16	Planning Time: 0.339 ms
17	Execution Time: 0.253 ms

➢ Με cluster "Student" using surname_index;

	QUERY PLAN text
1	Hash Join (cost=15.99..434.51 rows=1 width=198) (actual time=0.269..0.350 rows=2 loops=1)
2	Hash Cond: ("Register".amka = "Student".amka)
3	-> Bitmap Heap Scan on "Register" (cost=9.70..428.22 rows=1 width=37) (actual time=0.059..0.138 rows=3 loops=1)
4	Recheck Cond: (course_code = 'ΠΛΗ 101'::bpchar)
5	Filter: (final_grade = '7'::numeric)
6	Rows Removed by Filter: 161
7	Heap Blocks: exact=52
8	-> Bitmap Index Scan on "Register_pkey" (cost=0.00..9.70 rows=171 width=0) (actual time=0.045..0.045 rows=164 loops=1)
9	Index Cond: (course_code = 'ΠΛΗ 101'::bpchar)
10	-> Hash (cost=4.93..4.93 rows=109 width=165) (actual time=0.183..0.184 rows=86 loops=1)
11	Buckets: 1024 Batches: 1 Memory Usage: 25kB
12	-> Seq Scan on "Student" (cost=0.00..4.93 rows=109 width=165) (actual time=0.019..0.152 rows=86 loops=1)
13	Filter: ("right"((surname)::text, 1) <> 'Σ'::text)
14	Rows Removed by Filter: 24
15	Planning Time: 1.319 ms
16	Execution Time: 0.399 ms

➢ Mε cluster "Register" using course_code_index;

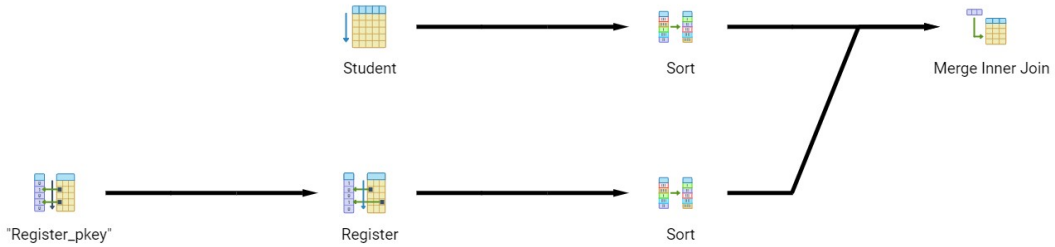
	QUERY PLAN text
1	Hash Join (cost=11.86..294.09 rows=1 width=198) (actual time=0.181..0.207 rows=2 loops=1)
2	Hash Cond: ("Register".amka = "Student".amka)
3	-> Bitmap Heap Scan on "Register" (cost=5.57..287.80 rows=1 width=37) (actual time=0.066..0.090 rows=3 loops=1)
4	Recheck Cond: (course_code = 'ΠΛΗ 101'::bpchar)
5	Filter: (final_grade = '7'::numeric)
6	Rows Removed by Filter: 161
7	Heap Blocks: exact=3
8	-> Bitmap Index Scan on course_code_index (cost=0.00..5.57 rows=171 width=0) (actual time=0.060..0.060 rows=164 loops=1)
9	Index Cond: (course_code = 'ΠΛΗ 101'::bpchar)
10	-> Hash (cost=4.93..4.93 rows=109 width=165) (actual time=0.106..0.107 rows=86 loops=1)
11	Buckets: 1024 Batches: 1 Memory Usage: 25kB
12	-> Seq Scan on "Student" (cost=0.00..4.93 rows=109 width=165) (actual time=0.015..0.086 rows=86 loops=1)
13	Filter: ("right"((surname)::text, 1) <> 'Σ'::text)
14	Rows Removed by Filter: 24
15	Planning Time: 0.792 ms
16	Execution Time: 0.251 ms

➢ Mε cluster "Register" using final_grade_index;

	QUERY PLAN text
1	Hash Join (cost=14.87..20.22 rows=1 width=198) (actual time=0.163..0.235 rows=2 loops=1)
2	Hash Cond: ("Student".amka = "Register".amka)
3	-> Seq Scan on "Student" (cost=0.00..4.93 rows=109 width=165) (actual time=0.015..0.086 rows=86 loops=1)
4	Filter: ("right"((surname)::text, 1) <> 'Σ'::text)
5	Rows Removed by Filter: 24
6	-> Hash (cost=14.86..14.86 rows=1 width=37) (actual time=0.135..0.135 rows=3 loops=1)
7	Buckets: 1024 Batches: 1 Memory Usage: 9kB
8	-> Bitmap Heap Scan on "Register" (cost=10.85..14.86 rows=1 width=37) (actual time=0.127..0.128 rows=3 loops=1)
9	Recheck Cond: ((final_grade = '7'::numeric) AND (course_code = 'ΠΛΗ 101'::bpchar))
10	Heap Blocks: exact=1
11	-> BitmapAnd (cost=10.85..10.85 rows=1 width=0) (actual time=0.122..0.122 rows=0 loops=1)
12	-> Bitmap Index Scan on final_grade_index (cost=0.00..4.90 rows=81 width=0) (actual time=0.046..0.046 rows=72 loops=1)
13	Index Cond: (final_grade = '7'::numeric)
14	-> Bitmap Index Scan on "Register_pkey" (cost=0.00..5.70 rows=171 width=0) (actual time=0.073..0.073 rows=164 loops=1)
15	Index Cond: (course_code = 'ΠΛΗ 101'::bpchar)
16	Planning Time: 0.795 ms
17	Execution Time: 0.276 ms

Τον καλύτερο μέσο χρόνο απόδοσης τον πήραμε με hash index στο course_code.

Έπειτα, απενεργοποιήσαμε το Hash Inner Join, με την εντολή
set enable_hashjoin=off;



	QUERY PLAN text
1	Merge Join (cost=283.20..283.72 rows=2 width=198) (actual time=0.249..0.252 rows=2 loops=1)
2	Merge Cond: ("Student".amka = "Register".amka)
3	-> Sort (cost=8.61..8.89 rows=109 width=165) (actual time=0.115..0.119 rows=76 loops=1)
4	Sort Key: "Student".amka
5	Sort Method: quicksort Memory: 47kB
6	-> Seq Scan on "Student" (cost=0.00..4.93 rows=109 width=165) (actual time=0.015..0.085 rows=86 loops=1)
7	Filter: ("right"((surname)::text, 1) <> 'Σ'::text)
8	Rows Removed by Filter: 24
9	-> Sort (cost=274.58..274.59 rows=2 width=37) (actual time=0.122..0.123 rows=3 loops=1)
10	Sort Key: "Register".amka
11	Sort Method: quicksort Memory: 25kB
12	-> Bitmap Heap Scan on "Register" (cost=5.69..274.57 rows=2 width=37) (actual time=0.096..0.117 rows=3 loops=1)
13	Recheck Cond: (course_code = 'ΠΛΗ 101'::bpchar)
14	Filter: (final_grade = '7'::numeric)
15	Rows Removed by Filter: 161
16	Heap Blocks: exact=41
17	-> Bitmap Index Scan on "Register_pkey" (cost=0.00..5.69 rows=170 width=0) (actual time=0.040..0.040 rows=164 loops=1)
18	Index Cond: (course_code = 'ΠΛΗ 101'::bpchar)
19	Planning Time: 0.275 ms
20	Execution Time: 0.300 ms

Παρατηρούμε ότι ο μέσος χρόνος εκτέλεσης του ερωτήματος σε σχέση με το αρχικό query βελτιώθηκε για 150 ms περίπου, ενώ σε σχέση με το προηγούμενο βήμα όπου απενεργοποιήσαμε το Nested Loop Inner Join, ο χρόνος αυξήθηκα κατά 40 ms περίπου.

- **Με b+tree index στο surname**

	QUERY PLAN text
2	Merge Cond: ("Student".amka = "Register".amka)
3	-> Sort (cost=8.61..8.89 rows=109 width=165) (actual time=0.155..0.159 rows=76 loops=1)
4	Sort Key: "Student".amka
5	Sort Method: quicksort Memory: 47kB
6	-> Seq Scan on "Student" (cost=0.00..4.93 rows=109 width=165) (actual time=0.023..0.120 rows=86 loops=1)
7	Filter: ("right"((surname)::text, 1) <> 'Σ'::text)
8	Rows Removed by Filter: 24
9	-> Sort (cost=287.94..287.94 rows=1 width=37) (actual time=0.141..0.141 rows=3 loops=1)
10	Sort Key: "Register".amka
11	Sort Method: quicksort Memory: 25kB
12	-> Bitmap Heap Scan on "Register" (cost=5.70..287.93 rows=1 width=37) (actual time=0.094..0.126 rows=3 loops=1)
13	Recheck Cond: (course_code = 'ΠΛΗ 101'::bpchar)
14	Filter: (final_grade = '7'::numeric)
15	Rows Removed by Filter: 161
16	Heap Blocks: exact=14
17	-> Bitmap Index Scan on "Register_pkey" (cost=0.00..5.70 rows=171 width=0) (actual time=0.056..0.056 rows=164 loops=1)
18	Index Cond: (course_code = 'ΠΛΗ 101'::bpchar)
19	Planning Time: 0.384 ms
20	Execution Time: 0.383 ms

- **Με hash index στο surname**

	QUERY PLAN text
2	Merge Cond: ("Student".amka = "Register".amka)
3	-> Sort (cost=8.61..8.89 rows=109 width=165) (actual time=0.151..0.155 rows=76 loops=1)
4	Sort Key: "Student".amka
5	Sort Method: quicksort Memory: 47kB
6	-> Seq Scan on "Student" (cost=0.00..4.93 rows=109 width=165) (actual time=0.023..0.118 rows=86 loops=1)
7	Filter: ("right"((surname)::text, 1) <> 'Σ'::text)
8	Rows Removed by Filter: 24
9	-> Sort (cost=287.94..287.94 rows=1 width=37) (actual time=0.118..0.119 rows=3 loops=1)
10	Sort Key: "Register".amka
11	Sort Method: quicksort Memory: 25kB
12	-> Bitmap Heap Scan on "Register" (cost=5.70..287.93 rows=1 width=37) (actual time=0.075..0.102 rows=3 loops=1)
13	Recheck Cond: (course_code = 'ΠΛΗ 101'::bpchar)
14	Filter: (final_grade = '7'::numeric)
15	Rows Removed by Filter: 161
16	Heap Blocks: exact=14
17	-> Bitmap Index Scan on "Register_pkey" (cost=0.00..5.70 rows=171 width=0) (actual time=0.047..0.047 rows=164 loops=1)
18	Index Cond: (course_code = 'ΠΛΗ 101'::bpchar)
19	Planning Time: 1.589 ms
20	Execution Time: 0.358 ms

➢ Με b+tree index στο course_code

	QUERY PLAN text
2	Merge Cond: ("Student".amka = "Register".amka)
3	-> Sort (cost=8.61..8.89 rows=109 width=165) (actual time=0.147..0.151 rows=76 loops=1)
4	Sort Key: "Student".amka
5	Sort Method: quicksort Memory: 47kB
6	-> Seq Scan on "Student" (cost=0.00..4.93 rows=109 width=165) (actual time=0.021..0.114 rows=86 loops=1)
7	Filter: ("right"((surname)::text, 1) <> 'Σ'::text)
8	Rows Removed by Filter: 24
9	-> Sort (cost=287.81..287.82 rows=1 width=37) (actual time=0.124..0.124 rows=3 loops=1)
10	Sort Key: "Register".amka
11	Sort Method: quicksort Memory: 25kB
12	-> Bitmap Heap Scan on "Register" (cost=5.57..287.80 rows=1 width=37) (actual time=0.082..0.110 rows=3 loops=1)
13	Recheck Cond: (course_code = 'ΠΛΗ 101'::bpchar)
14	Filter: (final_grade = '7'::numeric)
15	Rows Removed by Filter: 161
16	Heap Blocks: exact=14
17	-> Bitmap Index Scan on course_code_index (cost=0.00..5.57 rows=171 width=0) (actual time=0.046..0.046 rows=164 loops=1)
18	Index Cond: (course_code = 'ΠΛΗ 101'::bpchar)
19	Planning Time: 0.457 ms
20	Execution Time: 0.351 ms

➢ Με hash index στο course_code

	QUERY PLAN text
2	Merge Cond: ("Student".amka = "Register".amka)
3	-> Sort (cost=8.61..8.89 rows=109 width=165) (actual time=0.179..0.183 rows=76 loops=1)
4	Sort Key: "Student".amka
5	Sort Method: quicksort Memory: 47kB
6	-> Seq Scan on "Student" (cost=0.00..4.93 rows=109 width=165) (actual time=0.022..0.139 rows=86 loops=1)
7	Filter: ("right"((surname)::text, 1) <> 'Σ'::text)
8	Rows Removed by Filter: 24
9	-> Sort (cost=287.94..287.94 rows=1 width=37) (actual time=0.112..0.112 rows=3 loops=1)
10	Sort Key: "Register".amka
11	Sort Method: quicksort Memory: 25kB
12	-> Bitmap Heap Scan on "Register" (cost=5.70..287.93 rows=1 width=37) (actual time=0.077..0.096 rows=3 loops=1)
13	Recheck Cond: (course_code = 'ΠΛΗ 101'::bpchar)
14	Filter: (final_grade = '7'::numeric)
15	Rows Removed by Filter: 161
16	Heap Blocks: exact=3
17	-> Bitmap Index Scan on "Register_pkey" (cost=0.00..5.70 rows=171 width=0) (actual time=0.048..0.048 rows=164 loops=1)
18	Index Cond: (course_code = 'ΠΛΗ 101'::bpchar)
19	Planning Time: 0.405 ms
20	Execution Time: 0.399 ms

- **Με b+tree index στο final_grade**

	QUERY PLAN text
3	-> Sort (cost=8.61..8.89 rows=109 width=165) (actual time=0.145..0.150 rows=76 loops=1)
4	Sort Key: "Student".amka
5	Sort Method: quicksort Memory: 47kB
6	-> Seq Scan on "Student" (cost=0.00..4.93 rows=109 width=165) (actual time=0.020..0.112 rows=86 loops=1)
7	Filter: ("right"((surname)::text, 1) <> 'Σ'::text)
8	Rows Removed by Filter: 24
9	-> Sort (cost=14.87..14.88 rows=1 width=37) (actual time=0.094..0.094 rows=3 loops=1)
10	Sort Key: "Register".amka
11	Sort Method: quicksort Memory: 25kB
12	-> Bitmap Heap Scan on "Register" (cost=10.85..14.86 rows=1 width=37) (actual time=0.078..0.080 rows=3 loops=1)
13	Recheck Cond: ((final_grade = '7'::numeric) AND (course_code = 'ΠΛΗ 101'::bpchar))
14	Heap Blocks: exact=1
15	-> BitmapAnd (cost=10.85..10.85 rows=1 width=0) (actual time=0.073..0.074 rows=0 loops=1)
16	-> Bitmap Index Scan on final_grade_index (cost=0.00..4.90 rows=81 width=0) (actual time=0.027..0.027 rows=72 loops=1)
17	Index Cond: (final_grade = '7'::numeric)
18	-> Bitmap Index Scan on "Register_pkey" (cost=0.00..5.70 rows=171 width=0) (actual time=0.043..0.043 rows=164 loops=1)
19	Index Cond: (course_code = 'ΠΛΗ 101'::bpchar)
20	Planning Time: 0.372 ms
21	Execution Time: 0.322 ms

- **Με hash index στο final_grade**

	QUERY PLAN text
3	-> Sort (cost=8.61..8.89 rows=109 width=165) (actual time=0.156..0.161 rows=76 loops=1)
4	Sort Key: "Student".amka
5	Sort Method: quicksort Memory: 47kB
6	-> Seq Scan on "Student" (cost=0.00..4.93 rows=109 width=165) (actual time=0.022..0.121 rows=86 loops=1)
7	Filter: ("right"((surname)::text, 1) <> 'Σ'::text)
8	Rows Removed by Filter: 24
9	-> Sort (cost=14.58..14.59 rows=1 width=37) (actual time=0.096..0.096 rows=3 loops=1)
10	Sort Key: "Register".amka
11	Sort Method: quicksort Memory: 25kB
12	-> Bitmap Heap Scan on "Register" (cost=10.56..14.57 rows=1 width=37) (actual time=0.080..0.081 rows=3 loops=1)
13	Recheck Cond: ((final_grade = '7'::numeric) AND (course_code = 'ΠΛΗ 101'::bpchar))
14	Heap Blocks: exact=1
15	-> BitmapAnd (cost=10.56..10.56 rows=1 width=0) (actual time=0.072..0.072 rows=0 loops=1)
16	-> Bitmap Index Scan on hash_final_grade_index (cost=0.00..4.61 rows=81 width=0) (actual time=0.018..0.018 rows=72 loops=1)
17	Index Cond: (final_grade = '7'::numeric)
18	-> Bitmap Index Scan on "Register_pkey" (cost=0.00..5.70 rows=171 width=0) (actual time=0.050..0.050 rows=164 loops=1)
19	Index Cond: (course_code = 'ΠΛΗ 101'::bpchar)
20	Planning Time: 0.371 ms
21	Execution Time: 0.345 ms

- **Με cluster "Student" using surname_index;**

	QUERY PLAN text
2	Merge Cond: ("Student".amka = "Register".amka)
3	-> Sort (cost=8.61..8.89 rows=109 width=165) (actual time=0.151..0.155 rows=76 loops=1)
4	Sort Key: "Student".amka
5	Sort Method: quicksort Memory: 47kB
6	-> Seq Scan on "Student" (cost=0.00..4.93 rows=109 width=165) (actual time=0.021..0.113 rows=86 loops=1)
7	Filter: ("right"((surname)::text, 1) <> 'Σ'::text)
8	Rows Removed by Filter: 24
9	-> Sort (cost=287.94..287.94 rows=1 width=37) (actual time=0.113..0.113 rows=3 loops=1)
10	Sort Key: "Register".amka
11	Sort Method: quicksort Memory: 25kB
12	-> Bitmap Heap Scan on "Register" (cost=5.70..287.93 rows=1 width=37) (actual time=0.078..0.106 rows=3 loops=1)
13	Recheck Cond: (course_code = 'ΠΛΗ 101'::bpchar)
14	Filter: (final_grade = '7'::numeric)
15	Rows Removed by Filter: 161
16	Heap Blocks: exact=14
17	-> Bitmap Index Scan on "Register_pkey" (cost=0.00..5.70 rows=171 width=0) (actual time=0.048..0.048 rows=164 loops=1)
18	Index Cond: (course_code = 'ΠΛΗ 101'::bpchar)
19	Planning Time: 1.474 ms
20	Execution Time: 0.352 ms

- **Με cluster "Register" using course_code_index;**

	QUERY PLAN text
2	Merge Cond: ("Student".amka = "Register".amka)
3	-> Sort (cost=8.61..8.89 rows=109 width=165) (actual time=0.116..0.119 rows=76 loops=1)
4	Sort Key: "Student".amka
5	Sort Method: quicksort Memory: 47kB
6	-> Seq Scan on "Student" (cost=0.00..4.93 rows=109 width=165) (actual time=0.016..0.086 rows=86 loops=1)
7	Filter: ("right"((surname)::text, 1) <> 'Σ'::text)
8	Rows Removed by Filter: 24
9	-> Sort (cost=287.81..287.82 rows=1 width=37) (actual time=0.108..0.108 rows=3 loops=1)
10	Sort Key: "Register".amka
11	Sort Method: quicksort Memory: 25kB
12	-> Bitmap Heap Scan on "Register" (cost=5.57..287.80 rows=1 width=37) (actual time=0.090..0.104 rows=3 loops=1)
13	Recheck Cond: (course_code = 'ΠΛΗ 101'::bpchar)
14	Filter: (final_grade = '7'::numeric)
15	Rows Removed by Filter: 161
16	Heap Blocks: exact=3
17	-> Bitmap Index Scan on course_code_index (cost=0.00..5.57 rows=171 width=0) (actual time=0.071..0.071 rows=164 loops=1)
18	Index Cond: (course_code = 'ΠΛΗ 101'::bpchar)
19	Planning Time: 0.835 ms
20	Execution Time: 0.285 ms

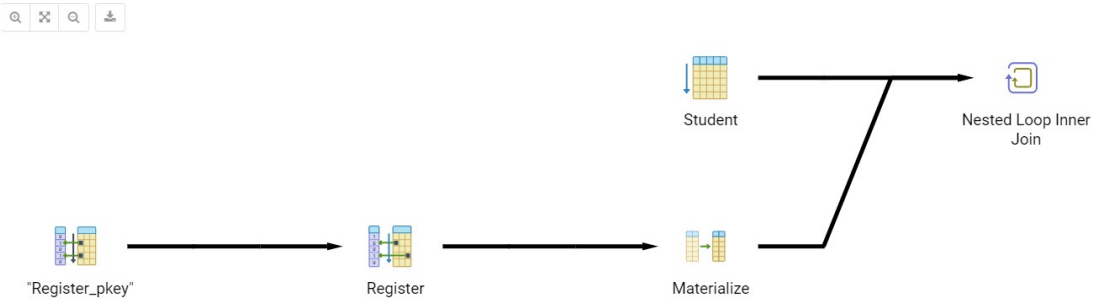
- **Με cluster "Register" using final_grade_index;**

	QUERY PLAN text
3	-> Sort (cost=8.61..8.89 rows=109 width=165) (actual time=0.116..0.118 rows=76 loops=1)
4	Sort Key: "Student".amka
5	Sort Method: quicksort Memory: 47kB
6	-> Seq Scan on "Student" (cost=0.00..4.93 rows=109 width=165) (actual time=0.015..0.085 rows=86 loops=1)
7	Filter: ("right"((surname)::text, 1) <> 'Σ'::text)
8	Rows Removed by Filter: 24
9	-> Sort (cost=14.87..14.88 rows=1 width=37) (actual time=0.140..0.140 rows=3 loops=1)
10	Sort Key: "Register".amka
11	Sort Method: quicksort Memory: 25kB
12	-> Bitmap Heap Scan on "Register" (cost=10.85..14.86 rows=1 width=37) (actual time=0.133..0.134 rows=3 loops=...
13	Recheck Cond: ((final_grade = '7'::numeric) AND (course_code = 'ΠΛΗ 101'::bpchar))
14	Heap Blocks: exact=1
15	-> BitmapAnd (cost=10.85..10.85 rows=1 width=0) (actual time=0.128..0.128 rows=0 loops=1)
16	-> Bitmap Index Scan on final_grade_index (cost=0.00..4.90 rows=81 width=0) (actual time=0.045..0.045 ro...
17	Index Cond: (final_grade = '7'::numeric)
18	-> Bitmap Index Scan on "Register_pkey" (cost=0.00..5.70 rows=171 width=0) (actual time=0.081..0.081 row...
19	Index Cond: (course_code = 'ΠΛΗ 101'::bpchar)
20	Planning Time: 0.793 ms
21	Execution Time: 0.315 ms

Τον καλύτερο μέσο χρόνο απόδοσης τον πήραμε με cluster "Register" using course_code_index.

Τέλος, απενεργοποιήσαμε το Merge Inner Join με την εντολή

set enable_mergejoin=off;



	QUERY PLAN text
1	Nested Loop (cost=10000000005.69..10000000282.77 rows=2 width=198) (actual time=0.172..0.285 rows=2 loops=1)
2	Join Filter: ("Student".amka = "Register".amka)
3	Rows Removed by Join Filter: 256
4	-> Seq Scan on "Student" (cost=0.00..4.93 rows=109 width=165) (actual time=0.018..0.091 rows=86 loops=1)
5	Filter: ("right"((surname)::text, 1) <> 'Σ'::text)
6	Rows Removed by Filter: 24
7	-> Materialize (cost=5.69..274.58 rows=2 width=37) (actual time=0.001..0.002 rows=3 loops=86)
8	-> Bitmap Heap Scan on "Register" (cost=5.69..274.57 rows=2 width=37) (actual time=0.113..0.138 rows=3 loops=1)
9	Recheck Cond: (course_code = 'ΠΛΗ 101'::bpchar)
10	Filter: (final_grade = '7'::numeric)
11	Rows Removed by Filter: 161
12	Heap Blocks: exact=41
13	-> Bitmap Index Scan on "Register_pkey" (cost=0.00..5.69 rows=170 width=0) (actual time=0.035..0.035 rows=164 loops=1)
14	Index Cond: (course_code = 'ΠΛΗ 101'::bpchar)
15	Planning Time: 0.225 ms
16	Execution Time: 0.327 ms

Παρατηρούμε ότι ο μέσος χρόνος εκτέλεσης του ερωτήματος σε σχέση με το αρχικό query βελτιώθηκε για 100 ms περίπου, ενώ σε σχέση με τα προηγούμενα βήματα μας δίνει το χειρότερο χρόνο απόδοσης.

- **Με b+tree index στο surname**

	QUERY PLAN text
1	Nested Loop (cost=10000000005.70..10000000294.49 rows=1 width=198) (actual time=0.143..0.297 rows=2 loops=1)
2	Join Filter: ("Student".amka = "Register".amka)
3	Rows Removed by Join Filter: 256
4	-> Seq Scan on "Student" (cost=0.00..4.93 rows=109 width=165) (actual time=0.022..0.117 rows=86 loops=1)
5	Filter: ("right"((surname)::text, 1) <> 'Σ'::text)
6	Rows Removed by Filter: 24
7	-> Materialize (cost=5.70..287.93 rows=1 width=37) (actual time=0.001..0.001 rows=3 loops=86)
8	-> Bitmap Heap Scan on "Register" (cost=5.70..287.93 rows=1 width=37) (actual time=0.073..0.101 rows=3 loops=1)
9	Recheck Cond: (course_code = 'ΠΛΗ 101'::bpchar)
10	Filter: (final_grade = '7'::numeric)
11	Rows Removed by Filter: 161
12	Heap Blocks: exact=14
13	-> Bitmap Index Scan on "Register_pkey" (cost=0.00..5.70 rows=171 width=0) (actual time=0.047..0.047 rows=164 loops=1)
14	Index Cond: (course_code = 'ΠΛΗ 101'::bpchar)
15	Planning Time: 0.302 ms
16	Execution Time: 0.353 ms

➢ Με hash index στο surname

	QUERY PLAN text
1	Nested Loop (cost=10000000005.70..10000000294.49 rows=1 width=198) (actual time=0.141..0.261 rows=2 loops=1)
2	Join Filter: ("Student".amka = "Register".amka)
3	Rows Removed by Join Filter: 256
4	-> Seq Scan on "Student" (cost=0.00..4.93 rows=109 width=165) (actual time=0.022..0.099 rows=86 loops=1)
5	Filter: ("right"((surname)::text, 1) <> 'Σ'::text)
6	Rows Removed by Filter: 24
7	-> Materialize (cost=5.70..287.93 rows=1 width=37) (actual time=0.001..0.001 rows=3 loops=86)
8	-> Bitmap Heap Scan on "Register" (cost=5.70..287.93 rows=1 width=37) (actual time=0.077..0.099 rows=3 loops=1)
9	Recheck Cond: (course_code = 'ΠΛΗ 101'::bpchar)
10	Filter: (final_grade = '7'::numeric)
11	Rows Removed by Filter: 161
12	Heap Blocks: exact=14
13	-> Bitmap Index Scan on "Register_pkey" (cost=0.00..5.70 rows=171 width=0) (actual time=0.048..0.048 rows=164 loops=1)
14	Index Cond: (course_code = 'ΠΛΗ 101'::bpchar)
15	Planning Time: 0.303 ms
16	Execution Time: 0.321 ms

➢ Με b+tree index στο course_code

	QUERY PLAN text
1	Nested Loop (cost=10000000005.57..10000000294.36 rows=1 width=198) (actual time=0.139..0.291 rows=2 loops=1)
2	Join Filter: ("Student".amka = "Register".amka)
3	Rows Removed by Join Filter: 256
4	-> Seq Scan on "Student" (cost=0.00..4.93 rows=109 width=165) (actual time=0.022..0.115 rows=86 loops=1)
5	Filter: ("right"((surname)::text, 1) <> 'Σ'::text)
6	Rows Removed by Filter: 24
7	-> Materialize (cost=5.57..287.81 rows=1 width=37) (actual time=0.001..0.001 rows=3 loops=86)
8	-> Bitmap Heap Scan on "Register" (cost=5.57..287.80 rows=1 width=37) (actual time=0.069..0.098 rows=3 loops=1)
9	Recheck Cond: (course_code = 'ΠΛΗ 101'::bpchar)
10	Filter: (final_grade = '7'::numeric)
11	Rows Removed by Filter: 161
12	Heap Blocks: exact=14
13	-> Bitmap Index Scan on course_code_index (cost=0.00..5.57 rows=171 width=0) (actual time=0.042..0.042 rows=164 loops=1)
14	Index Cond: (course_code = 'ΠΛΗ 101'::bpchar)
15	Planning Time: 0.289 ms
16	Execution Time: 0.341 ms

➢ Με hash index στο course_code

	QUERY PLAN text
1	Nested Loop (cost=10000000005.70..10000000294.49 rows=1 width=198) (actual time=0.125..0.278 rows=2 loops=1)
2	Join Filter: ("Student".amka = "Register".amka)
3	Rows Removed by Join Filter: 256
4	-> Seq Scan on "Student" (cost=0.00..4.93 rows=109 width=165) (actual time=0.021..0.116 rows=86 loops=1)
5	Filter: ("right"((surname)::text, 1) <> 'Σ'::text)
6	Rows Removed by Filter: 24
7	-> Materialize (cost=5.70..287.93 rows=1 width=37) (actual time=0.001..0.001 rows=3 loops=86)
8	-> Bitmap Heap Scan on "Register" (cost=5.70..287.93 rows=1 width=37) (actual time=0.065..0.084 rows=3 loops=1)
9	Recheck Cond: (course_code = 'ΠΛΗ 101'::bpchar)
10	Filter: (final_grade = '7'::numeric)
11	Rows Removed by Filter: 161
12	Heap Blocks: exact=3
13	-> Bitmap Index Scan on "Register_pkey" (cost=0.00..5.70 rows=171 width=0) (actual time=0.044..0.044 rows=164 loops=1)
14	Index Cond: (course_code = 'ΠΛΗ 101'::bpchar)
15	Planning Time: 0.298 ms
16	Execution Time: 0.329 ms

➢ Με b+tree index στο final_grade

	QUERY PLAN text
1	Nested Loop (cost=10000000010.85..10000000021.42 rows=1 width=198) (actual time=0.116..0.267 rows=2 loops=1)
2	Join Filter: ("Student".amka = "Register".amka)
3	Rows Removed by Join Filter: 256
4	-> Seq Scan on "Student" (cost=0.00..4.93 rows=109 width=165) (actual time=0.021..0.115 rows=86 loops=1)
5	Filter: ("right"((surname)::text, 1) <> 'Σ'::text)
6	Rows Removed by Filter: 24
7	-> Materialize (cost=10.85..14.87 rows=1 width=37) (actual time=0.001..0.001 rows=3 loops=86)
8	-> Bitmap Heap Scan on "Register" (cost=10.85..14.86 rows=1 width=37) (actual time=0.073..0.075 rows=3 loops=1)
9	Recheck Cond: ((final_grade = '7'::numeric) AND (course_code = 'ΠΛΗ 101'::bpchar))
10	Heap Blocks: exact=1
11	-> BitmapAnd (cost=10.85..10.85 rows=1 width=0) (actual time=0.068..0.068 rows=0 loops=1)
12	-> Bitmap Index Scan on final_grade_index (cost=0.00..4.90 rows=81 width=0) (actual time=0.026..0.026 rows=72 loops=1)
13	Index Cond: (final_grade = '7'::numeric)
14	-> Bitmap Index Scan on "Register_pkey" (cost=0.00..5.70 rows=171 width=0) (actual time=0.037..0.037 rows=164 loops=1)
15	Index Cond: (course_code = 'ΠΛΗ 101'::bpchar)
16	Planning Time: 0.292 ms
17	Execution Time: 0.317 ms

➢ Με hash index στο final_grade

	QUERY PLAN text
1	Nested Loop (cost=10000000010.56..10000000021.13 rows=1 width=198) (actual time=0.100..0.226 rows=2 loops=1)
2	Join Filter: ("Student".amka = "Register".amka)
3	Rows Removed by Join Filter: 256
4	-> Seq Scan on "Student" (cost=0.00..4.93 rows=109 width=165) (actual time=0.021..0.095 rows=86 loops=1)
5	Filter: ("right"((surname)::text, 1) <> 'Σ'::text)
6	Rows Removed by Filter: 24
7	-> Materialize (cost=10.56..14.58 rows=1 width=37) (actual time=0.001..0.001 rows=3 loops=86)
8	-> Bitmap Heap Scan on "Register" (cost=10.56..14.57 rows=1 width=37) (actual time=0.059..0.061 rows=3 loops=1)
9	Recheck Cond: ((final_grade = '7'::numeric) AND (course_code = 'ΠΛΗ 101'::bpchar))
10	Heap Blocks: exact=1
11	-> BitmapAnd (cost=10.56..10.56 rows=1 width=0) (actual time=0.054..0.054 rows=0 loops=1)
12	-> Bitmap Index Scan on hash_final_grade_index (cost=0.00..4.61 rows=81 width=0) (actual time=0.012..0.012 rows=72 loops=1)
13	Index Cond: (final_grade = '7'::numeric)
14	-> Bitmap Index Scan on "Register_pkey" (cost=0.00..5.70 rows=171 width=0) (actual time=0.038..0.038 rows=164 loops=1)
15	Index Cond: (course_code = 'ΠΛΗ 101'::bpchar)
16	Planning Time: 0.255 ms
17	Execution Time: 0.270 ms

- **Με cluster "Student" using surname_index;**

	QUERY PLAN text
1	Nested Loop (cost=10000000005.70..10000000294.49 rows=1 width=198) (actual time=0.111..0.225 rows=2 loops=1)
2	Join Filter: ("Student".amka = "Register".amka)
3	Rows Removed by Join Filter: 256
4	-> Seq Scan on "Student" (cost=0.00..4.93 rows=109 width=165) (actual time=0.015..0.086 rows=86 loops=1)
5	Filter: ("right"((surname)::text, 1) <> 'Σ'::text)
6	Rows Removed by Filter: 24
7	-> Materialize (cost=5.70..287.93 rows=1 width=37) (actual time=0.001..0.001 rows=3 loops=86)
8	-> Bitmap Heap Scan on "Register" (cost=5.70..287.93 rows=1 width=37) (actual time=0.056..0.078 rows=3 loops=1)
9	Recheck Cond: (course_code = 'ΠΛΗ 101'::bpchar)
10	Filter: (final_grade = '7'::numeric)
11	Rows Removed by Filter: 161
12	Heap Blocks: exact=14
13	-> Bitmap Index Scan on "Register_pkey" (cost=0.00..5.70 rows=171 width=0) (actual time=0.036..0.036 rows=164 loops=1)
14	Index Cond: (course_code = 'ΠΛΗ 101'::bpchar)
15	Planning Time: 0.918 ms
16	Execution Time: 0.268 ms

- **Με cluster "Register" using course_code_index;**

	QUERY PLAN text
1	Nested Loop (cost=10000000005.57..10000000294.36 rows=1 width=198) (actual time=0.139..0.255 rows=2 loops=1)
2	Join Filter: ("Student".amka = "Register".amka)
3	Rows Removed by Join Filter: 256
4	-> Seq Scan on "Student" (cost=0.00..4.93 rows=109 width=165) (actual time=0.016..0.089 rows=86 loops=1)
5	Filter: ("right"((surname)::text, 1) <> 'Σ'::text)
6	Rows Removed by Filter: 24
7	-> Materialize (cost=5.57..287.81 rows=1 width=37) (actual time=0.001..0.001 rows=3 loops=86)
8	-> Bitmap Heap Scan on "Register" (cost=5.57..287.80 rows=1 width=37) (actual time=0.083..0.099 rows=3 loops=1)
9	Recheck Cond: (course_code = 'ΠΛΗ 101'::bpchar)
10	Filter: (final_grade = '7'::numeric)
11	Rows Removed by Filter: 161
12	Heap Blocks: exact=3
13	-> Bitmap Index Scan on course_code_index (cost=0.00..5.57 rows=171 width=0) (actual time=0.065..0.065 rows=164 loops=1)
14	Index Cond: (course_code = 'ΠΛΗ 101'::bpchar)
15	Planning Time: 0.767 ms
16	Execution Time: 0.297 ms

- **Με cluster "Register" using final_grade_index;**

	QUERY PLAN text
1	Nested Loop (cost=10000000010.85..10000000021.42 rows=1 width=198) (actual time=0.165..0.279 rows=2 loops=1)
2	Join Filter: ("Student".amka = "Register".amka)
3	Rows Removed by Join Filter: 256
4	-> Seq Scan on "Student" (cost=0.00..4.93 rows=109 width=165) (actual time=0.015..0.087 rows=86 loops=1)
5	Filter: ("right"((surname)::text, 1) <> 'Σ'::text)
6	Rows Removed by Filter: 24
7	-> Materialize (cost=10.85..14.87 rows=1 width=37) (actual time=0.002..0.002 rows=3 loops=86)
8	-> Bitmap Heap Scan on "Register" (cost=10.85..14.86 rows=1 width=37) (actual time=0.127..0.128 rows=3 loops=...
9	Recheck Cond: ((final_grade = '7'::numeric) AND (course_code = 'ΠΛΗ 101'::bpchar))
10	Heap Blocks: exact=1
11	-> BitmapAnd (cost=10.85..10.85 rows=1 width=0) (actual time=0.122..0.122 rows=0 loops=1)
12	-> Bitmap Index Scan on final_grade_index (cost=0.00..4.90 rows=81 width=0) (actual time=0.050..0.050 ro...
13	Index Cond: (final_grade = '7'::numeric)
14	-> Bitmap Index Scan on "Register_pkey" (cost=0.00..5.70 rows=171 width=0) (actual time=0.070..0.070 row...
15	Index Cond: (course_code = 'ΠΛΗ 101'::bpchar)
16	Planning Time: 0.764 ms
17	Execution Time: 0.323 ms

Τον καλύτερο μέσο χρόνο απόδοσης τον πήραμε με cluster "Student" using surname_index.

Τέλος, μας ζητήθηκε να αλλάξουμε τη σειρά των συνδέσεων, επομένως εκτελέσαμε το ακόλουθο query:

explain analyze

select *
from "Register" natural left join "Student"
where right(surname,1)<> 'Σ' and course_code='ΠΛΗ 101' and final_grade=7

	QUERY PLAN text
1	Nested Loop (cost=5.69..282.77 rows=2 width=198) (actual time=0.158..0.329 rows=2 loops=1)
2	Join Filter: ("Register".amka = "Student".amka)
3	Rows Removed by Join Filter: 256
4	-> Seq Scan on "Student" (cost=0.00..4.93 rows=109 width=165) (actual time=0.016..0.133 rows=86 loops=1)
5	Filter: ("right"((surname)::text, 1) <> 'Σ'::text)
6	Rows Removed by Filter: 24
7	-> Materialize (cost=5.69..274.58 rows=2 width=37) (actual time=0.001..0.002 rows=3 loops=86)
8	-> Bitmap Heap Scan on "Register" (cost=5.69..274.57 rows=2 width=37) (actual time=0.083..0.116 rows=3 loops=1)
9	Recheck Cond: (course_code = 'ΠΛΗ 101'::bpchar)
10	Filter: (final_grade = '7'::numeric)
11	Rows Removed by Filter: 161
12	Heap Blocks: exact=41
13	-> Bitmap Index Scan on "Register_pkey" (cost=0.00..5.69 rows=170 width=0) (actual time=0.033..0.033 rows=16...
14	Index Cond: (course_code = 'ΠΛΗ 101'::bpchar)
15	Planning Time: 0.292 ms
16	Execution Time: 0.368 ms

Παρατηρούμε ότι ο μέσος χρόνος εκτέλεσης του ερωτήματος είναι 0,368 ms.

- ***Με b+tree index στο surname***

	QUERY PLAN text
1	Nested Loop (cost=5.69..282.77 rows=2 width=198) (actual time=0.155..0.378 rows=2 loops=1)
2	Join Filter: ("Register".amka = "Student".amka)
3	Rows Removed by Join Filter: 256
4	-> Seq Scan on "Student" (cost=0.00..4.93 rows=109 width=165) (actual time=0.020..0.166 rows=86 loops=1)
5	Filter: ("right"((surname)::text, 1) <> 'Σ'::text)
6	Rows Removed by Filter: 24
7	-> Materialize (cost=5.69..274.58 rows=2 width=37) (actual time=0.001..0.002 rows=3 loops=86)
8	-> Bitmap Heap Scan on "Register" (cost=5.69..274.57 rows=2 width=37) (actual time=0.092..0.119 rows=3 loops=1)
9	Recheck Cond: (course_code = 'ΠΛΗ 101'::bpchar)
10	Filter: (final_grade = '7'::numeric)
11	Rows Removed by Filter: 161
12	Heap Blocks: exact=41
13	-> Bitmap Index Scan on "Register_pkey" (cost=0.00..5.69 rows=170 width=0) (actual time=0.035..0.035 rows=164 loops=1)
14	Index Cond: (course_code = 'ΠΛΗ 101'::bpchar)
15	Planning Time: 0.308 ms
16	Execution Time: 0.429 ms

Παρατηρούμε ότι ο μέσος χρόνος εκτέλεσης σε σχέση με το αρχικό query αυξάνεται για περίπου 100 ms.

- ***Με hash index στο surname***

	QUERY PLAN text
1	Nested Loop (cost=5.69..282.77 rows=2 width=198) (actual time=0.143..0.248 rows=2 loops=1)
2	Join Filter: ("Register".amka = "Student".amka)
3	Rows Removed by Join Filter: 256
4	-> Seq Scan on "Student" (cost=0.00..4.93 rows=109 width=165) (actual time=0.024..0.097 rows=86 loops=1)
5	Filter: ("right"((surname)::text, 1) <> 'Σ'::text)
6	Rows Removed by Filter: 24
7	-> Materialize (cost=5.69..274.58 rows=2 width=37) (actual time=0.001..0.001 rows=3 loops=86)
8	-> Bitmap Heap Scan on "Register" (cost=5.69..274.57 rows=2 width=37) (actual time=0.085..0.107 rows=3 loops=1)
9	Recheck Cond: (course_code = 'ΠΛΗ 101'::bpchar)
10	Filter: (final_grade = '7'::numeric)
11	Rows Removed by Filter: 161
12	Heap Blocks: exact=41
13	-> Bitmap Index Scan on "Register_pkey" (cost=0.00..5.69 rows=170 width=0) (actual time=0.036..0.036 rows=164 loops=1)
14	Index Cond: (course_code = 'ΠΛΗ 101'::bpchar)
15	Planning Time: 0.732 ms
16	Execution Time: 0.311 ms

Παρατηρούμε ότι ο μέσος χρόνος εκτέλεσης σε σχέση με το αρχικό query μειώνεται για περίπου 50 ms.

- ***Με b+tree index στο course_code***

	QUERY PLAN text
1	Nested Loop (cost=5.57..282.65 rows=2 width=198) (actual time=0.242..0.346 rows=2 loops=1)
2	Join Filter: ("Register".amka = "Student".amka)
3	Rows Removed by Join Filter: 256
4	-> Seq Scan on "Student" (cost=0.00..4.93 rows=109 width=165) (actual time=0.016..0.087 rows=86 loops=1)
5	Filter: ("right"((surname)::text, 1) <> 'Σ'::text)
6	Rows Removed by Filter: 24
7	-> Materialize (cost=5.57..274.46 rows=2 width=37) (actual time=0.002..0.003 rows=3 loops=86)
8	-> Bitmap Heap Scan on "Register" (cost=5.57..274.45 rows=2 width=37) (actual time=0.160..0.212 rows=3 loops=1)
9	Recheck Cond: (course_code = 'ΠΛΗ 101'::bpchar)
10	Filter: (final_grade = '7'::numeric)
11	Rows Removed by Filter: 161
12	Heap Blocks: exact=41
13	-> Bitmap Index Scan on course_code_index (cost=0.00..5.57 rows=170 width=0) (actual time=0.051..0.051 rows=164 loops=1)
14	Index Cond: (course_code = 'ΠΛΗ 101'::bpchar)
15	Planning Time: 0.285 ms
16	Execution Time: 0.387 ms

Παρατηρούμε ότι ο μέσος χρόνος εκτέλεσης σε σχέση με το αρχικό query είναι περίπου ο ίδιος.

- ***Με hash index στο course_code***

	QUERY PLAN text
1	Nested Loop (cost=5.69..282.77 rows=2 width=198) (actual time=0.154..0.327 rows=2 loops=1)
2	Join Filter: ("Register".amka = "Student".amka)
3	Rows Removed by Join Filter: 256
4	-> Seq Scan on "Student" (cost=0.00..4.93 rows=109 width=165) (actual time=0.017..0.128 rows=86 loops=1)
5	Filter: ("right"((surname)::text, 1) <> 'Σ'::text)
6	Rows Removed by Filter: 24
7	-> Materialize (cost=5.69..274.58 rows=2 width=37) (actual time=0.001..0.002 rows=3 loops=86)
8	-> Bitmap Heap Scan on "Register" (cost=5.69..274.57 rows=2 width=37) (actual time=0.094..0.120 rows=3 loops=1)
9	Recheck Cond: (course_code = 'ΠΛΗ 101'::bpchar)
10	Filter: (final_grade = '7'::numeric)
11	Rows Removed by Filter: 161
12	Heap Blocks: exact=41
13	-> Bitmap Index Scan on "Register_pkey" (cost=0.00..5.69 rows=170 width=0) (actual time=0.034..0.034 rows=164 loops=1)
14	Index Cond: (course_code = 'ΠΛΗ 101'::bpchar)
15	Planning Time: 0.413 ms
16	Execution Time: 0.369 ms

Παρατηρούμε ξανά ότι ο μέσος χρόνος εκτέλεσης σε σχέση με το αρχικό query μειώνεται για περίπου 50 ms.

- ***Με b+tree index στο final_grade***

	QUERY PLAN text
1	Hash Join (cost=24.89..30.38 rows=2 width=198) (actual time=0.109..0.180 rows=2 loops=1)
2	Hash Cond: ("Student".amka = "Register".amka)
3	-> Seq Scan on "Student" (cost=0.00..4.93 rows=109 width=165) (actual time=0.016..0.085 rows=86 loops=1)
4	Filter: ("right"((surname)::text, 1) <> 'Σ'::text)
5	Rows Removed by Filter: 24
6	-> Hash (cost=24.87..24.87 rows=2 width=37) (actual time=0.080..0.080 rows=3 loops=1)
7	Buckets: 1024 Batches: 1 Memory Usage: 9kB
8	-> Bitmap Heap Scan on "Register" (cost=17.33..24.87 rows=2 width=37) (actual time=0.077..0.078 rows=3 loops=1)
9	Recheck Cond: ((course_code = 'ΠΛΗ 101'::bpchar) AND (final_grade = '7'::numeric))
10	Heap Blocks: exact=1
11	-> BitmapAnd (cost=17.33..17.33 rows=2 width=0) (actual time=0.074..0.074 rows=0 loops=1)
12	-> Bitmap Index Scan on "Register_pkey" (cost=0.00..5.69 rows=170 width=0) (actual time=0.033..0.033 rows=164 loops=1)
13	Index Cond: (course_code = 'ΠΛΗ 101'::bpchar)
14	-> Bitmap Index Scan on final_grade_index (cost=0.00..11.39 rows=413 width=0) (actual time=0.038..0.038 rows=409 loops=1)
15	Index Cond: (final_grade = '7'::numeric)
16	Planning Time: 0.292 ms
17	Execution Time: 0.226 ms

Παρατηρούμε ότι ο μέσος χρόνος εκτέλεσης σε σχέση με το αρχικό query μειώνεται για περίπου 150 ms.

- ***Με hash index στο final_grade***

	QUERY PLAN text
1	Hash Join (cost=28.60..34.09 rows=2 width=198) (actual time=0.088..0.157 rows=2 loops=1)
2	Hash Cond: ("Student".amka = "Register".amka)
3	-> Seq Scan on "Student" (cost=0.00..4.93 rows=109 width=165) (actual time=0.015..0.084 rows=86 loops=1)
4	Filter: ("right"((surname)::text, 1) <> 'Σ'::text)
5	Rows Removed by Filter: 24
6	-> Hash (cost=28.58..28.58 rows=2 width=37) (actual time=0.060..0.060 rows=3 loops=1)
7	Buckets: 1024 Batches: 1 Memory Usage: 9kB
8	-> Bitmap Heap Scan on "Register" (cost=21.04..28.58 rows=2 width=37) (actual time=0.057..0.058 rows=3 loops=1)
9	Recheck Cond: ((course_code = 'ΠΛΗ 101'::bpchar) AND (final_grade = '7'::numeric))
10	Heap Blocks: exact=1
11	-> BitmapAnd (cost=21.04..21.04 rows=2 width=0) (actual time=0.053..0.053 rows=0 loops=1)
12	-> Bitmap Index Scan on "Register_pkey" (cost=0.00..5.69 rows=170 width=0) (actual time=0.032..0.032 rows=164 loops=1)
13	Index Cond: (course_code = 'ΠΛΗ 101'::bpchar)
14	-> Bitmap Index Scan on hash_final_grade_index (cost=0.00..15.10 rows=413 width=0) (actual time=0.018..0.018 rows=409 loop…
15	Index Cond: (final_grade = '7'::numeric)
16	Planning Time: 0.309 ms
17	Execution Time: 0.215 ms

Παρατηρούμε ότι ο μέσος χρόνος εκτέλεσης σε σχέση με το αρχικό query μειώνεται για περίπου 150 ms, επομένως το hash index στο final_grade είναι το αποδοτικότερο ευρετήριο.

- ***Με cluster "Student" using surname_index;***

	QUERY PLAN text
1	Nested Loop (cost=5.69..282.77 rows=2 width=198) (actual time=0.122..0.224 rows=2 loops=1)
2	Join Filter: ("Register".amka = "Student".amka)
3	Rows Removed by Join Filter: 256
4	-> Seq Scan on "Student" (cost=0.00..4.93 rows=109 width=165) (actual time=0.015..0.084 rows=86 loops=1)
5	Filter: ("right"((surname)::text, 1) <> 'Σ'::text)
6	Rows Removed by Filter: 24
7	-> Materialize (cost=5.69..274.58 rows=2 width=37) (actual time=0.001..0.001 rows=3 loops=86)
8	-> Bitmap Heap Scan on "Register" (cost=5.69..274.57 rows=2 width=37) (actual time=0.075..0.096 rows=3 loops=1)
9	Recheck Cond: (course_code = 'ΠΛΗ 101'::bpchar)
10	Filter: (final_grade = '7'::numeric)
11	Rows Removed by Filter: 161
12	Heap Blocks: exact=41
13	-> Bitmap Index Scan on "Register_pkey" (cost=0.00..5.69 rows=170 width=0) (actual time=0.029..0.029 rows=164 loops=1)
14	Index Cond: (course_code = 'ΠΛΗ 101'::bpchar)
15	Planning Time: 0.803 ms
16	Execution Time: 0.259 ms

Παρατηρούμε ότι ο μέσος χρόνος εκτέλεσης σε σχέση με το αρχικό query μειώνεται για περίπου 100 ms.

- ***Με cluster "Register" using course_code_index;***

	QUERY PLAN text
1	Nested Loop (cost=5.57..283.09 rows=2 width=198) (actual time=0.127..0.231 rows=2 loops=1)
2	Join Filter: ("Register".amka = "Student".amka)
3	Rows Removed by Join Filter: 256
4	-> Seq Scan on "Student" (cost=0.00..4.93 rows=109 width=165) (actual time=0.015..0.086 rows=86 loops=1)
5	Filter: ("right"((surname)::text, 1) <> 'Σ'::text)
6	Rows Removed by Filter: 24
7	-> Materialize (cost=5.57..274.90 rows=2 width=37) (actual time=0.001..0.001 rows=3 loops=86)
8	-> Bitmap Heap Scan on "Register" (cost=5.57..274.89 rows=2 width=37) (actual time=0.085..0.094 rows=3 loops=1)
9	Recheck Cond: (course_code = 'ΠΛΗ 101'::bpchar)
10	Filter: (final_grade = '7'::numeric)
11	Rows Removed by Filter: 161
12	Heap Blocks: exact=3
13	-> Bitmap Index Scan on course_code_index (cost=0.00..5.57 rows=170 width=0) (actual time=0.060..0.060 rows=164 loops=1)
14	Index Cond: (course_code = 'ΠΛΗ 101'::bpchar)
15	Planning Time: 0.797 ms
16	Execution Time: 0.266 ms

Παρατηρούμε και πάλι ότι ο μέσος χρόνος εκτέλεσης σε σχέση με το αρχικό query μειώνεται για περίπου 100 ms.

- ***Με cluster "Register" using final_grade_index;***

	QUERY PLAN text
1	Hash Join (cost=24.89..30.38 rows=2 width=198) (actual time=0.221..0.291 rows=2 loops=1)
2	Hash Cond: ("Student".amka = "Register".amka)
3	-> Seq Scan on "Student" (cost=0.00..4.93 rows=109 width=165) (actual time=0.016..0.084 rows=86 loops=1)
4	Filter: ("right"((surname)::text, 1) <> 'Σ'::text)
5	Rows Removed by Filter: 24
6	-> Hash (cost=24.87..24.87 rows=2 width=37) (actual time=0.186..0.186 rows=3 loops=1)
7	Buckets: 1024 Batches: 1 Memory Usage: 9kB
8	-> Bitmap Heap Scan on "Register" (cost=17.33..24.87 rows=2 width=37) (actual time=0.173..0.174 rows=3 loops=1)
9	Recheck Cond: ((course_code = 'ΠΛΗ 101'::bpchar) AND (final_grade = '7'::numeric))
10	Heap Blocks: exact=1
11	-> BitmapAnd (cost=17.33..17.33 rows=2 width=0) (actual time=0.171..0.171 rows=0 loops=1)
12	-> Bitmap Index Scan on "Register_pkey" (cost=0.00..5.69 rows=170 width=0) (actual time=0.083..0.083 rows=164 loops=1)
13	Index Cond: (course_code = 'ΠΛΗ 101'::bpchar)
14	-> Bitmap Index Scan on final_grade_index (cost=0.00..11.39 rows=413 width=0) (actual time=0.084..0.084 rows=409 loops=1)
15	Index Cond: (final_grade = '7'::numeric)
16	Planning Time: 0.693 ms
17	Execution Time: 0.331 ms

Παρατηρούμε ότι ο μέσος χρόνος εκτέλεσης σε σχέση με το αρχικό query μειώνεται για περίπου 50 ms.

Τον καλύτερο μέσο χρόνο απόδοσης τον πήραμε με hash index στο final_grade.