ΒΑΣΕΙΣ ΔΕΔΟΜΕΝΩΝ - ΠΛΗ302
ΔΙΔΑΣΚΩΝ: Αντώνιος Δεληγιαννάκης
ΥΠΟΣΤΗΡΙΞΗ ΕΡΓΑΣΤΗΡΙΟΥ: Μουμουτζής Νεκτάριος, Παππάς Νίκος

Β' ΦΑΣΗ ΕΡΓΑΣΤΗΡΙΑΚΗΣ ΕΡΓΑΣΙΑΣ – ΕΑΡΙΝΟ ΕΞΑΜΗΝΟ 2017-2018

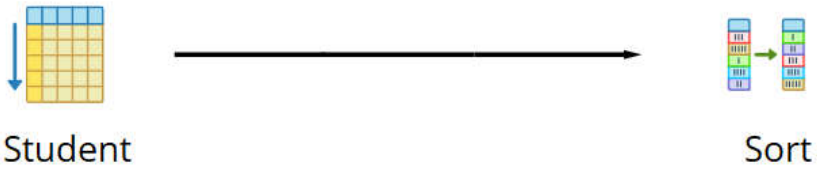
Ομάδα Εργασίας:

- ΓΑΛΛΟΥ ΟΛΥΜΠΙΑ ,ΑΜ: 2015030101
- ΚΑΛΟΓΕΡΑΚΗΣ ΣΤΕΦΑΝΟΣ ,ΑΜ: 2015030064

2. Μελέτη απόδοσης ερωτήσεων – φυσικός σχεδιασμός:

Α) *Μελετήστε το εξής αίτημα: Βρες τους φοιτητές που έχουν επώνυμο στο διάστημα αλφαριθμητικών από 'ΜΑ έως 'ΜΟ'.*

```
EXPLAIN  ANALYSE

SELECT  s.surname

FROM  "Student"  as  s

WHERE s.surname BETWEEN 'MA' AND 'ΜΠ'

ORDER BY  s.surname;
```



Data Output | Explain | Messages | Query History

	QUERY PLAN text
1	Sort (cost=4.79..4.82 rows=9 width=39) (actual time=0.274..0.275 rows=10 loops=1)
2	Sort Key: surname
3	Sort Method: quicksort Memory: 25kB
4	-> Seq Scan on "Student" s (cost=0.00..4.65 rows=9 width=39) (actual time=0.154..0.243 rows=10 loops=1)
5	Filter: ((surname >= 'MA'::bpchar) AND (surname <= 'ΜΠ'::bpchar))
6	Rows Removed by Filter: 100
7	Planning time: 2.933 ms
8	Execution time: 0.312 ms

Παρατηρήσεις και Συμπεράσματα

Από τα παραπάνω αποτελέσματα στο Data Output και την ανάλυση του query plan,συμπεραίνουμε ότι χρησιμοποιήθηκε ο αλγόριθμος quicksort με κλειδί το surname λόγω της ταξινόμησης του πίνακα με βάση το επίθετο των φοιτητών(**ORDER BY** s.surname).

Επιπλέον, παρατηρήσαμε ότι χρησιμοποιείται η μέθοδος Seq Scan στον πίνακα "Student" ,αφού μας ενδιαφέρει μόνο η στήλη του attribute 'surname',επομένως διατηρούνται μόνο tuples της στήλης των επιθέτων στον πίνακα των φοιτητών ,σύμφωνα με τις συνθήκες στο φίλτρο WHERE(Rows removed by filter=100).Τέλος, στην πρώτη εκτέλεση του query,συγκρατήσαμε τον εκτιμώμενο/αναμενόμενο χρόνο σχεδιασμού(planning time) καθώς και τον πραγματικό χρόνο που χρειάστηκε για να εκτελεστεί το query(execution time). Αυτό που έχει νόημα είναι η σύγκριση του αναμενόμενου μεγέθους του αποτελέσματος σε σχέση με το πραγματικό. Με βάση αυτές τις τιμές θα συγκρίνουμε τις παρακάτω απόπειρες για βελτίωση της απόδοσης των συνδέσεων.

Planning Time: 2.933 ms

Execution Time: 0.312 ms

Μια ακόμη παρατήρηση κατά την διάρκεια της βελτιστοποίησης είναι ότι οι χρόνοι εκτέλεσης είναι μεταβλητοί και όσες περισσότερες φορές εκτελούσαμε με explain analyse τα στατιστικά στον χρόνο για ένα ερώτημα μειώνονταν (κατά μέσο όρο).Κάποιες φορές η διαφορά στον αναμενόμενο χρόνο και στον χρόνο εκτέλεσης μπορεί να ήταν πολύ μικρή ενώ κάποιες άλλες ο αναμενόμενος χρόνος παρατηρήθηκε μέχρι και τριπλάσιος απ'ότι ο πραγματικός χρόνος.

Στόχος είναι η βελτιστοποίηση των αιτημάτων με την μείωση των χρόνων εκτέλεσης. Στην βελτίωση απόδοσης απλών ερωτημάτων έχουμε 3 βασικές εναλλακτικές:

- Δενδρικό Ευρετήριο(*b+tree index*)
- Ευρετήριο Κατακερματισμού(*hash index*)
- Ομαδοποίηση πλειάδων πίνακα με βάση κάποιο ευρετήριο(*clustering*)

Με την χρήση δεικτών (indexes) επιτυγχάνουμε πιο γρήγορη ανάκτηση δεδομένων από την βάση και επομένως βελτιστοποιείται η απόδοση των ερωτημάτων.

Η πρώτη δημιουργία ευρετηρίου ήταν με b+tree και το αναμενόμενο ήταν εξαρχής να βοηθάει έστω και λίγο στον χρόνο εκτέλεσης. Πράγματι ,όπως παρατίθεται παρακάτω ο χρόνος βελτιώθηκε σημαντικά. Συγκεκριμένα, Planning Time από 2.933ms σε 0.698ms και Execution Time από 0.312 ms σε 0.127ms.

Με b+tree index

```
1  --CREATE INDEX student_surname_idx ON "Student" USING btree(surname);
2  EXPLAIN ANALYSE
3  SELECT s.surname
4  FROM "Student" as s
5  WHERE s.surname BETWEEN 'ΜΑ' AND 'ΜΠ'
6  ORDER BY s.surname;
```

Data Output | Explain | Messages | Query History

	QUERY PLAN text
1	Sort (cost=4.79..4.82 rows=9 width=39) (actual time=0.108..0.108 rows=10 loops=1)
2	Sort Key: surname
3	Sort Method: quicksort Memory: 25kB
4	-> Seq Scan on "Student" s (cost=0.00..4.65 rows=9 width=39) (actual time=0.043..0.091 rows=10 loops=1)
5	Filter: ((surname >= 'MA'::bpchar) AND (surname <= 'ΜΠ'::bpchar))
6	Rows Removed by Filter: 100
7	Planning time: 0.698 ms
8	Execution time: 0.127 ms

Με hash index

Στην συγκεκριμένη περίπτωση το αναμενόμενο ήταν να μην βοηθήσει το ευρετήριο κατακερματισμού σε σχέση με το δενδρικό, καθώς ενδείκνυται κυρίως σε περιπτώσεις «σημείου»(point queries),ενώ στο συγκεκριμένο αίτημα έχουμε range query.Πράγματι μετά την ανάλυση των στατιστικών οδηγηθήκαμε στο συμπέρασμα ότι όχι απλά δεν είναι τόσο αποδοτικό όσο το b+tree,αλλά δεν προσέφερε ουσιαστικά και καμία βοήθεια σε σχέση με τους αρχικούς χρόνους όπου δεν υπήρχε κανένα ευρετήριο/δείκτης. Συνεπώς, στην συγκεκριμένη περίπτωση το hash index δεν συμφέρει.

```
1  --CREATE INDEX student_surname_idx ON "Student" using hash(surname);
2  EXPLAIN ANALYSE
3  SELECT s.surname
4  FROM "Student" as s
5  WHERE s.surname BETWEEN 'ΜΑ' AND 'ΜΠ'
6  ORDER BY s.surname;
```

Data Output | Explain | Messages | Query History

	QUERY PLAN text
1	Sort (cost=4.79..4.82 rows=9 width=39) (actual time=0.428..0.429 rows=10 loops=1)
2	Sort Key: surname
3	Sort Method: quicksort Memory: 25kB
4	-> Seq Scan on "Student" s (cost=0.00..4.65 rows=9 width=39) (actual time=0.187..0.375 rows=10 loops=1)
5	Filter: ((surname >= 'MA'::bpchar) AND (surname <= 'ΜΠ'::bpchar))
6	Rows Removed by Filter: 100
7	Planning time: 0.381 ms
8	Execution time: 0.512 ms

Clustering με βάση b+tree index

Στην συνέχεια δοκιμάσαμε την ομαδοποίηση πλειάδων του πίνακα Student με το αποδοτικό δενδρικό ευρετήριο που δημιουργήσαμε πριν. Ουσιαστικά, χρησιμοποιήσαμε τον δείκτη που δημιουργήσαμε πριν στο δενδρικό ευρετήριο (student_surname_idx),ο οποίος δείχνει στην στήλη surname του πίνακα “Student” και γίνεται ομαδοποίηση των εγγραφών του πίνακα με βάση αυτόν. Η διαδικασία αυτή λέγεται clustering.Όπως φαίνεται στο query plan με κυκλωμένους τους χρόνους που μας ενδιαφέρουν παρατηρούμε μεγάλη μείωση σε σχέση με τους αρχικούς χρόνους. Επίσης ,μετά από σύγκριση των χρόνων, έγινε κατανοητό ότι το clustering σε σχέση με το b+tree index αποφέρουν περίπου τα ίδια αποτελέσματα όσες φορές τα εκτελέσαμε κατά μέσο όρο. Είναι επομένως και τα 2 σχεδόν εξίσου αποδοτικά.

```
1  --DROP INDEX student_surname_idx;
2  --CREATE INDEX student_surname_idx ON "Student" using btree(surname);
3  CLUSTER "Student" USING student_surname_idx;
4  EXPLAIN ANALYSE
5  SELECT s.surname
6  FROM "Student" as s
7  WHERE s.surname BETWEEN 'ΜΑ' AND 'ΜΠ'
```

Data Output | Explain | Messages | Query History

	QUERY PLAN text
1	Sort (cost=4.79..4.82 rows=9 width=39) (actual time=0.104..0.105 rows=10 loops=1)
2	Sort Key: surname
3	Sort Method: quicksort Memory: 25kB
4	-> Seq Scan on "Student" s (cost=0.00..4.65 rows=9 width=39) (actual time=0.041..0.092 rows=10 loops=1)
5	Filter: ((surname >= 'MA'::bpchar) AND (surname <= 'ΜΠ'::bpchar))
6	Rows Removed by Filter: 100
7	Planning time: 0.781 ms
8	Execution time: 0.120 ms

Έπειτα, καταργήσαμε τα προηγούμενα ευρετήρια που είχαμε δημιουργήσει με DROP INDEX και αυξήσαμε κατά πολύ το πλήθος των φοιτητών στον πίνακα της βάσης κατά εκατοντάδες χιλιάδες, σύμφωνα και με τον κώδικα που μελετήσαμε στην 7η εργαστηριακή άσκηση.

Πιο αναλυτικά, χρησιμοποιώντας την συνάρτηση : `insert_1000_students_per_year(yearstart integer ,yearfinish integer)` έγινε η εισαγωγή 100.000 φοιτητών μέσα σε 100 έτη(1000 ανά έτος). Ακολουθώντας τα ίδια βήματα όπως και προηγουμένως, οδηγηθήκαμε στα παρακάτω αποτελέσματα ,τα οποία επαληθεύουν την καλύτερη απόδοση του clustering σε μεγάλο όγκο δεδομένων.

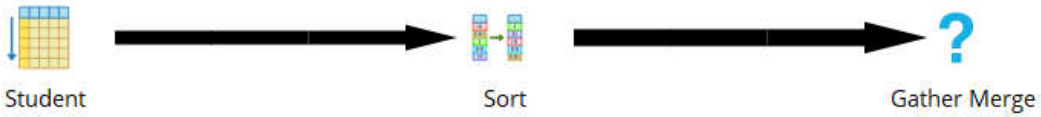
Σε αυτό το σημείο από 110 αρχικές εγγραφές φοιτητών στον πίνακα “Student”, στην βάση έχουν προστεθεί πλέον 300.000 φοιτητές ακόμη .Συνολικά δηλαδή 300110 φοιτητές.

ΧΩΡΙΣ ΚΑΝΕΝΑ ΕΥΡΕΤΗΡΙΟ/ΔΕΙΚΤΗ

```
--select insert_1000_students_per_year(1800,1899);
EXPLAIN  ANALYSE
SELECT  s.surname
FROM  "Student"  as  s
WHERE s.surname BETWEEN 'ΜΑ' AND 'ΜΠ'
ORDER BY  s.surname;
```

Data Output | Explain | Messages | Query History

	QUERY PLAN text
1	Gather Merge (cost=11090.99..13277.94 rows=18744 width=40) (actual time=308.781..340.594 rows=22260 loops=1)
2	Workers Planned: 2
3	Workers Launched: 2
4	-> Sort (cost=10090.96..10114.39 rows=9372 width=40) (actual time=242.464..243.714 rows=7420 loops=3)
5	Sort Key: surname
6	Sort Method: quicksort Memory: 1146kB
7	-> Parallel Seq Scan on "Student" s (cost=0.00..9472.69 rows=9372 width=40) (actual time=0.054..163.729 rows=7420 loops=3)
8	Filter: ((surname >= 'MA'::bpchar) AND (surname <= 'ΜΠ'::bpchar))
9	Rows Removed by Filter: 92617
10	Planning time: 1.522 ms
11	Execution time: 354.629 ms

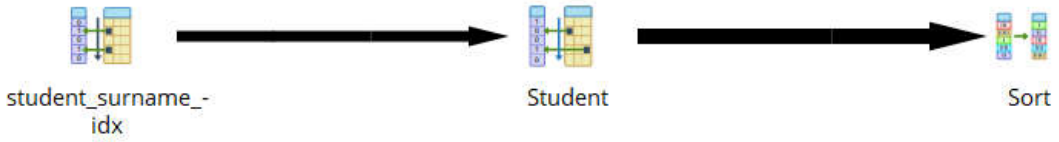


ΜΕ ΧΡΗΣΗ ΔΕΙΚΤΗ

```
CREATE INDEX student_surname_idx ON "Student"(surname);
EXPLAIN  ANALYSE
SELECT  s.surname
FROM  "Student"  as  s
WHERE s.surname BETWEEN 'ΜΑ' AND 'ΜΠ'
ORDER BY  s.surname;
```

Data Output | Explain | Messages | Query History

	QUERY PLAN text
1	Sort (cost=10479.40..10535.64 rows=22494 width=40) (actual time=178.664..181.640 rows=22260 loops=1)
2	Sort Key: surname
3	Sort Method: quicksort Memory: 2508kB
4	-> Bitmap Heap Scan on "Student" s (cost=918.99..8853.40 rows=22494 width=40) (actual time=23.048..33.511 rows=22260 loops=1)
5	Recheck Cond: ((surname >= 'MA'::bpchar) AND (surname <= 'ΜΠ'::bpchar))
6	Heap Blocks: exact=7043
7	-> Bitmap Index Scan on student_surname_idx (cost=0.00..913.36 rows=22494 width=0) (actual time=21.958..21.958 rows=22260 loops=1)
8	Index Cond: ((surname >= 'MA'::bpchar) AND (surname <= 'ΜΠ'::bpchar))
9	Planning time: 0.470 ms
10	Execution time: 182.983 ms

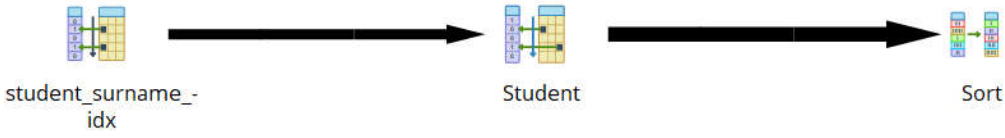


ΜΕ ΕΥΡΕΤΗΡΙΟ B+TREE

```
1  --select insert_1000_students_per_year(1800,1899);
2  --CREATE INDEX student_surname_idx ON "Student" USING btree(surname);
3
4  EXPLAIN  ANALYSE
5  SELECT  s.surname
6  FROM  "Student"  as  s
7  WHERE s.surname BETWEEN 'MA' AND 'ΜΠ'
8  ORDER BY  s.surname;
```

Data Output | Explain | Messages | Query History

	QUERY PLAN text
1	Sort (cost=10479.40..10535.64 rows=22494 width=40) (actual time=163.355..165.899 rows=22260 loops=1)
2	Sort Key: surname
3	Sort Method: quicksort Memory: 2508kB
4	-> Bitmap Heap Scan on "Student" s (cost=918.99..8853.40 rows=22494 width=40) (actual time=20.123..30.567 rows=22260 loops=1)
5	Recheck Cond: ((surname >= 'MA'::bpchar) AND (surname <= 'ΜΠ'::bpchar))
6	Heap Blocks: exact=7043
7	-> Bitmap Index Scan on student_surname_idx (cost=0.00..913.36 rows=22494 width=0) (actual time=18.963..18.963 rows=22260 loops=1)
8	Index Cond: ((surname >= 'MA'::bpchar) AND (surname <= 'ΜΠ'::bpchar))
9	Planning time: 0.414 ms
10	Execution time: 166.912 ms

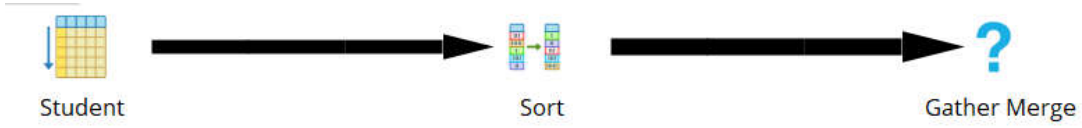


ΜΕ ΕΥΡΕΤΗΡΙΟ HASH

```
4  --CREATE INDEX student_surname_idx ON "Student" USING hash(surname);
5  EXPLAIN  ANALYSE
6  SELECT  s.surname
7  FROM  "Student"  as  s
8  WHERE s.surname BETWEEN 'ΜΑ' AND 'ΜΠ'
9  ORDER BY  s.surname;
10
```

Data Output | Explain | Messages | Query History

	QUERY PLAN text
1	Gather Merge (cost=11090.99..13277.94 rows=18744 width=40) (actual time=349.559..402.220 rows=22260 loops=1)
2	Workers Planned: 2
3	Workers Launched: 2
4	-> Sort (cost=10090.96..10114.39 rows=9372 width=40) (actual time=273.950..275.642 rows=7420 loops=3)
5	Sort Key: surname
6	Sort Method: quicksort Memory: 1161kB
7	-> Parallel Seq Scan on "Student" s (cost=0.00..9472.69 rows=9372 width=40) (actual time=0.044..183.537 rows=7420 loops=3)
8	Filter: ((surname >= 'MA'::bpchar) AND (surname <= 'ΜΠ'::bpchar))
9	Rows Removed by Filter: 92617
10	Planning time: 0.677 ms
11	Execution time: 419.211 ms



ΟΜΑΔΟΠΟΙΗΣΗ CLUSTERING ΜΕ INDEX DEFAULT

```
5  CREATE INDEX student_surname_idx ON "Student"(surname);
6  CLUSTER "Student" USING student_surname_idx;
7  EXPLAIN  ANALYSE
8  SELECT  s.surname
9  FROM  "Student"  as  s
10 WHERE s.surname BETWEEN 'ΜΑ' AND 'ΜΠ'
11 ORDER BY  s.surname;
```

Data Output | Explain | Messages | Query History

	QUERY PLAN text
1	Sort (cost=10261.40..10317.64 rows=22494 width=40) (actual time=36.201..37.098 rows=22260 loops=1)
2	Sort Key: surname
3	Sort Method: quicksort Memory: 2508kB
4	-> Bitmap Heap Scan on "Student" s (cost=918.99..8635.40 rows=22494 width=40) (actual time=19.194..27.455 rows=22260 loops=1)
5	Recheck Cond: ((surname >= 'MA'::bpchar) AND (surname <= 'ΜΠ'::bpchar))
6	Heap Blocks: exact=548
7	-> Bitmap Index Scan on student_surname_idx (cost=0.00..913.36 rows=22494 width=0) (actual time=19.089..19.089 rows=22260 loops=1)
8	Index Cond: ((surname >= 'MA'::bpchar) AND (surname <= 'ΜΠ'::bpchar))
9	Planning time: 1.000 ms
10	Execution time: 38.028 ms

ΟΜΑΔΟΠΟΙΗΣΗ CLUSTERING ΜΕ BTREE INDEX

```
5  --CREATE INDEX student_surname_idx ON "Student" USING btree(surname);
6  CLUSTER "Student" USING student_surname_idx;
7  EXPLAIN  ANALYSE
8  SELECT  s.surname
9  FROM  "Student"  as  s
10 WHERE s.surname BETWEEN 'ΜΑ' AND 'ΜΠ'
11 ORDER BY  s.surname;
```

Data Output | Explain | Messages | Query History

	QUERY PLAN text
1	Sort (cost=10261.40..10317.64 rows=22494 width=40) (actual time=32.179..33.019 rows=22260 loops=1)
2	Sort Key: surname
3	Sort Method: quicksort Memory: 2508kB
4	-> Bitmap Heap Scan on "Student" s (cost=918.99..8635.40 rows=22494 width=40) (actual time=19.358..23.854 rows=22260 loops=1)
5	Recheck Cond: ((surname >= 'MA'::bpchar) AND (surname <= 'ΜΠ'::bpchar))
6	Heap Blocks: exact=548
7	-> Bitmap Index Scan on student_surname_idx (cost=0.00..913.36 rows=22494 width=0) (actual time=19.281..19.281 rows=22260 loops=1)
8	Index Cond: ((surname >= 'MA'::bpchar) AND (surname <= 'ΜΠ'::bpchar))
9	Planning time: 0.441 ms
10	Execution time: 34.023 ms

Επεξηγηματικά, είναι προφανής η βελτίωση χρόνου με την χρήση ευρετηρίου b+tree σε σχέση με τους αρχικούς χρόνους χωρίς την χρήση κανενός ευρετηρίου. Η διαφορά όμως στην παρούσα κατάσταση όπου ο όγκος των φοιτητών είναι μερικές εκατοντάδες χιλιάδες(300110),η βελτίωση χρόνου είναι πολύ μεγαλύτερη στην διαδικασία clustering,όπου ο πραγματικός χρόνος εκτέλεσης του αιτήματος κατέρχεται από 354.629ms σε 34.023ms.Επομένως, η ομαδοποίηση είναι η πιο αποδοτική μέθοδος βελτιστοποίησης σε εκτενή όγκο δεδομένων.

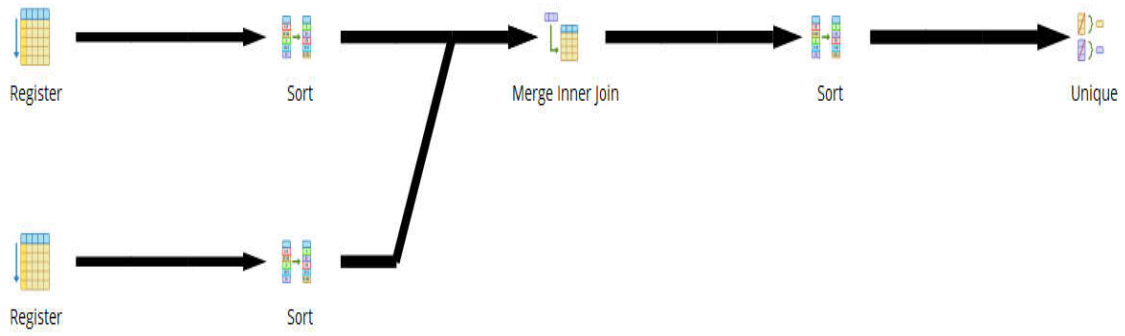
Β) *Μελετήστε το εξής αίτημα: Βρες ζεύγη κωδικών φοιτητών τέτοια ώστε σε κάθε τέτοιο ζεύγος οι δύο φοιτητές έχουν περάσει με τον ίδιο βαθμό κάποιο μάθημα.*

ΑΙΤΗΜΑ (Β):

```
SELECT DISTINCT
r1.course_code,r1.register_status,r1.final_grade,r1.amka,r2.amka

FROM "Register" as r1, "Register" as r2

WHERE r2.amka>r1.amka and r1.course_code=r2.course_code and
r1.register_status='pass' and r2.register_status='pass'

 and r1.final_grade=r2.final_grade;
```

Με την εντολή Explain Analyse, λαμβάνουμε αρχικά το συγκεκριμένο query plan και στην συνέχεια παρατηρούμε πόσο κοντά είναι οι εκτιμήσεις του σχεδιαστή με την πραγματικότητα.

	QUERY PLAN text
1	Unique (cost=2494.89..2497.59 rows=216 width=28) (actual time=126.111..130.919 rows=13492 loops=1)
2	-> Sort (cost=2494.89..2495.43 rows=216 width=28) (actual time=126.110..127.138 rows=13492 loops=1)
3	Sort Key: r1.course_code, r1.final_grade, r1.amka, r2.amka
4	Sort Method: quicksort Memory: 1439kB
5	-> Merge Join (cost=2403.80..2486.52 rows=216 width=28) (actual time=46.974..63.219 rows=13492 loops=1)
6	Merge Cond: ((r1.course_code = r2.course_code) AND (r1.final_grade = r2.final_grade))
7	Join Filter: (r2.amka > r1.amka)
8	Rows Removed by Join Filter: 18438
9	-> Sort (cost=1201.90..1214.34 rows=4975 width=24) (actual time=25.785..26.444 rows=4946 loops=1)
10	Sort Key: r1.course_code, r1.final_grade
11	Sort Method: quicksort Memory: 579kB
12	-> Seq Scan on "Register" r1 (cost=0.00..896.43 rows=4975 width=24) (actual time=1.579..10.384 rows=4946 loops=1)
13	Filter: (register_status = 'pass'::register_status_type)
14	Rows Removed by Filter: 41488
15	-> Sort (cost=1201.90..1214.34 rows=4975 width=20) (actual time=21.154..22.922 rows=31920 loops=1)
16	Sort Key: r2.course_code, r2.final_grade
17	Sort Method: quicksort Memory: 579kB
18	-> Seq Scan on "Register" r2 (cost=0.00..896.43 rows=4975 width=20) (actual time=0.719..5.683 rows=4946 loops=1)
19	Filter: (register_status = 'pass'::register_status_type)
20	Rows Removed by Filter: 41488
21	Planning time: 1.387 ms
22	Execution time: 131.980 ms



Στην συνέχεια, δοκιμάσαμε την δημιουργία ευρετηρίου με την μορφή δείκτη στην στήλη amka του πίνακα “Register” :

CREATE INDEX amk_index ON "Register"(amka);

Η εκτέλεση του query με explain analyse έδωσε την ανάλυση των στατιστικών όπου παρατηρούμε όπως επισυνάπτεται παρακάτω ότι ο πραγματικός χρόνος όπως και ο αναμενόμενος μειώθηκαν λίγο.

Data Output | Explain | Messages | Query History

	QUERY PLAN text
1	Unique (cost=2494.89..2497.59 rows=216 width=28) (actual time=120.274..124.796 rows=13492 loops=1)
2	-> Sort (cost=2494.89..2495.43 rows=216 width=28) (actual time=120.273..121.278 rows=13492 loops=1)
3	Sort Key: r1.course_code, r1.final_grade, r1.amka, r2.amka
4	Sort Method: quicksort Memory: 1439kB
5	-> Merge Join (cost=2403.80..2486.52 rows=216 width=28) (actual time=45.235..61.187 rows=13492 loops=1)
6	Merge Cond: ((r1.course_code = r2.course_code) AND (r1.final_grade = r2.final_grade))
7	Join Filter: (r2.amka > r1.amka)
8	Rows Removed by Join Filter: 18438
9	-> Sort (cost=1201.90..1214.34 rows=4975 width=24) (actual time=23.465..24.198 rows=4946 loops=1)
10	Sort Key: r1.course_code, r1.final_grade
11	Sort Method: quicksort Memory: 579kB
12	-> Seq Scan on "Register" r1 (cost=0.00..896.43 rows=4975 width=24) (actual time=1.113..8.308 rows=4946 loops=1)
13	Filter: (register_status = 'pass'::register_status_type)
14	Rows Removed by Filter: 41488
15	-> Sort (cost=1201.90..1214.34 rows=4975 width=20) (actual time=21.744..23.482 rows=31920 loops=1)
16	Sort Key: r2.course_code, r2.final_grade
17	Sort Method: quicksort Memory: 579kB
18	-> Seq Scan on "Register" r2 (cost=0.00..896.43 rows=4975 width=20) (actual time=0.574..5.889 rows=4946 loops=1)
19	Filter: (register_status = 'pass'::register_status_type)
20	Rows Removed by Filter: 41488
21	Planning time: 0.928 ms
22	Execution time: 125.709 ms

ΜΕ B+TREE INDEX(register_status)

```
2  CREATE INDEX reg_index ON "Register" USING btree(register_status);
3  EXPLAIN ANALYSE
4  SELECT DISTINCT r1.course_code,r1.register_status,r1.final_grade,r1.amka,r2.amka
5  FROM "Register" as r1, "Register" as r2
6  WHERE r2.amka>r1.amka and r1.course_code=r2.course_code and r1.register_status='pass' and r2.register_status='pass'
7   and r1.final_grade=r2.final_grade;
```

Data Output | Explain | Messages | Query History

	QUERY PLAN text
15	-> Bitmap Index Scan on reg_index (cost=0.00..93.60 rows=4975 width=0) (actual time=0.493..0.493 rows...
16	Index Cond: (register_status = 'pass'::register_status_type)
17	-> Sort (cost=778.51..790.95 rows=4975 width=20) (actual time=16.534..18.136 rows=31920 loops=1)
18	Sort Key: r2.course_code, r2.final_grade
19	Sort Method: quicksort Memory: 579kB
20	-> Bitmap Heap Scan on "Register" r2 (cost=94.85..473.03 rows=4975 width=20) (actual time=0.262..1.122 r...
21	Recheck Cond: (register_status = 'pass'::register_status_type)
22	Heap Blocks: exact=82
23	-> Bitmap Index Scan on reg_index (cost=0.00..93.60 rows=4975 width=0) (actual time=0.244..0.244 rows...
24	Index Cond: (register_status = 'pass'::register_status_type)
25	Planning time: 0.916 ms
26	Execution time: 115.337 ms

ΔΙΔΙΑΣΤΑΤΟ ΕΥΡΕΤΗΡΙΟ B+TREE (register_status,amka)

```
2  CREATE INDEX t_index ON "Register" USING btree(register_status,amka);
3  EXPLAIN ANALYSE
4  SELECT DISTINCT r1.course_code,r1.register_status,r1.final_grade,r1.amka,r2.amka
5  FROM "Register" as r1, "Register" as r2
6  WHERE r2.amka>r1.amka and r1.course_code=r2.course_code and r1.register_status='pass' and r2.register_status='pass'
7   and r1.final_grade=r2.final_grade;
```

Data Output | Explain | Messages | Query History

	QUERY PLAN text
15	-> Bitmap Index Scan on t_index (cost=0.00..93.60 rows=4975 width=0) (actual time=0.447..0.447 rows=4...
16	Index Cond: (register_status = 'pass'::register_status_type)
17	-> Sort (cost=778.51..790.95 rows=4975 width=20) (actual time=16.293..17.798 rows=31920 loops=1)
18	Sort Key: r2.course_code, r2.final_grade
19	Sort Method: quicksort Memory: 579kB
20	-> Bitmap Heap Scan on "Register" r2 (cost=94.85..473.03 rows=4975 width=20) (actual time=0.266..1.150 r...
21	Recheck Cond: (register_status = 'pass'::register_status_type)
22	Heap Blocks: exact=82
23	-> Bitmap Index Scan on t_index (cost=0.00..93.60 rows=4975 width=0) (actual time=0.247..0.247 rows=4...
24	Index Cond: (register_status = 'pass'::register_status_type)
25	Planning time: 0.952 ms
26	Execution time: 108.319 ms

ΑΛΛΑΓΗ ΣΤΗΝ ΣΕΙΡΑ ΤΩΝ ATTRIBUTES amka,register_status

```
1  DROP INDEX t_index;
2  CREATE INDEX t_index ON "Register" USING btree(amka,register_status);
3  EXPLAIN ANALYSE
4  SELECT DISTINCT r1.course_code,r1.register_status,r1.final_grade,r1.amka,r2.amka
5  FROM "Register" as r1, "Register" as r2
6  WHERE r2.amka>r1.amka and r1.course_code=r2.course_code and r1.register_status='pass' and r2.register_status='pass'
7   and r1.final_grade=r2.final_grade;
```

Data Output | Explain | Messages | Query History

	QUERY PLAN text
11	Sort Method: quicksort Memory: 579kB
12	-> Seq Scan on "Register" r1 (cost=0.00..896.43 rows=4975 width=24) (actual time=1.452..11.503 rows=494...
13	Filter: (register_status = 'pass'::register_status_type)
14	Rows Removed by Filter: 41488
15	-> Sort (cost=1201.90..1214.34 rows=4975 width=20) (actual time=21.562..23.351 rows=31920 loops=1)
16	Sort Key: r2.course_code, r2.final_grade
17	Sort Method: quicksort Memory: 579kB
18	-> Seq Scan on "Register" r2 (cost=0.00..896.43 rows=4975 width=20) (actual time=0.548..5.821 rows=4946 l...
19	Filter: (register_status = 'pass'::register_status_type)
20	Rows Removed by Filter: 41488
21	Planning time: 1.468 ms
22	Execution time: 133.189 ms

ΜΕ HASH INDEX (register_status)

CREATE INDEX reg_index ON "Register" USING hash(register_status);

	QUERY PLAN text
1	Unique (cost=1855.53..1858.23 rows=216 width=28) (actual time=107.921..111.679 rows=13492 loops=1)
2	-> Sort (cost=1855.53..1856.07 rows=216 width=28) (actual time=107.920..108.716 rows=13492 loops=1)
3	Sort Key: r1.course_code, r1.final_grade, r1.amka, r2.amka
4	Sort Method: quicksort Memory: 1439kB
5	-> Merge Join (cost=1764.44..1847.15 rows=216 width=28) (actual time=36.204..50.731 rows=13492 loops=1)
6	Merge Cond: ((r1.course_code = r2.course_code) AND (r1.final_grade = r2.final_grade))
7	Join Filter: (r2.amka > r1.amka)
8	Rows Removed by Join Filter: 18438
9	-> Sort (cost=882.22..894.66 rows=4975 width=24) (actual time=19.575..20.082 rows=4946 loops=1)
10	Sort Key: r1.course_code, r1.final_grade
11	Sort Method: quicksort Memory: 579kB
12	-> Bitmap Heap Scan on "Register" r1 (cost=198.56..576.74 rows=4975 width=24) (actual time=0.644..2.905 ...
13	Recheck Cond: (register_status = 'pass'::register_status_type)
14	Heap Blocks: exact=82
15	-> Bitmap Index Scan on amk_index (cost=0.00..197.31 rows=4975 width=0) (actual time=0.624..0.624 ro...
16	Index Cond: (register_status = 'pass'::register_status_type)
17	-> Sort (cost=882.22..894.66 rows=4975 width=20) (actual time=16.612..18.087 rows=31920 loops=1)
18	Sort Key: r2.course_code, r2.final_grade
19	Sort Method: quicksort Memory: 579kB
20	-> Bitmap Heap Scan on "Register" r2 (cost=198.56..576.74 rows=4975 width=20) (actual time=0.337..1.461 ...
21	Recheck Cond: (register_status = 'pass'::register_status_type)
22	Heap Blocks: exact=82
23	-> Bitmap Index Scan on amk_index (cost=0.00..197.31 rows=4975 width=0) (actual time=0.319..0.319 ro...
24	Index Cond: (register_status = 'pass'::register_status_type)
25	Planning time: 0.911 ms
26	Execution time: 115.985 ms

ΜΕ HASH INDEX (amka)

CREATE INDEX amk_index ON "Register" USING hash(amka);

```
2  CREATE INDEX amk_index ON "Register" USING hash(amka);
3  EXPLAIN ANALYSE
4  SELECT DISTINCT r1.course_code,r1.register_status,r1.final_grade,r1.amka,r2.amka
5  FROM "Register" as r1, "Register" as r2
6  WHERE r2.amka>r1.amka and r1.course_code=r2.course_code and r1.register_status='pass' and r2.register_status='pass'
7   and r1.final_grade=r2.final_grade;
```

Data Output | Explain | Messages | Query History

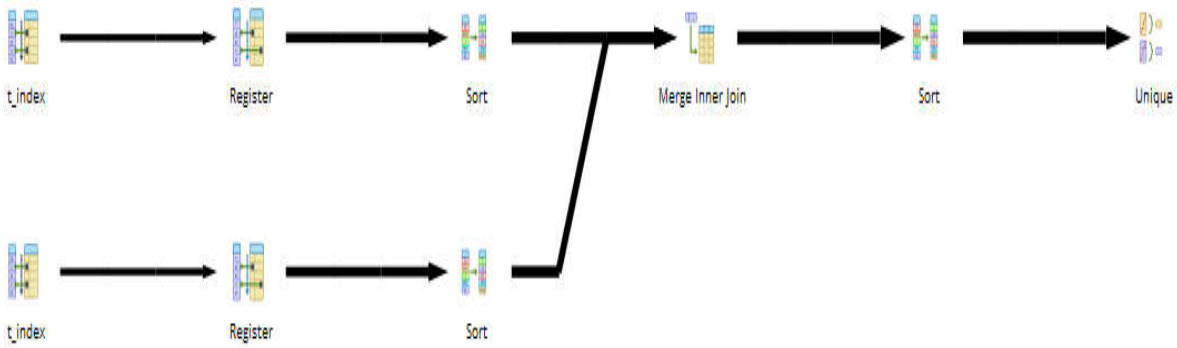
	QUERY PLAN text
10	Sort Key: r1.course_code, r1.final_grade
11	Sort Method: quicksort Memory: 579kB
12	-> Seq Scan on "Register" r1 (cost=0.00..896.43 rows=4975 width=24) (actual time=1.110..7.725 rows=4946 l...
13	Filter: (register_status = 'pass'::register_status_type)
14	Rows Removed by Filter: 41488
15	-> Sort (cost=1201.90..1214.34 rows=4975 width=20) (actual time=23.574..25.691 rows=31920 loops=1)
16	Sort Key: r2.course_code, r2.final_grade
17	Sort Method: quicksort Memory: 579kB
18	-> Seq Scan on "Register" r2 (cost=0.00..896.43 rows=4975 width=20) (actual time=0.590..6.132 rows=4946 l...
19	Filter: (register_status = 'pass'::register_status_type)
20	Rows Removed by Filter: 41488
21	Planning time: 0.840 ms
22	Execution time: 134.132 ms

ΣΥΝΔΥΑΣΜΟΣ HASH ΚΑΙ B+TREE ΕΥΡΕΤΗΡΙΩΝ

```
2  CREATE INDEX t_index ON "Register" USING btree(amka);
3  CREATE INDEX s_index ON "Register" USING hash(register_status);
4  EXPLAIN ANALYSE
5  SELECT DISTINCT r1.course_code,r1.register_status,r1.final_grade,r1.amka,r2.amka
6  FROM "Register" as r1, "Register" as r2
7  WHERE r2.amka>r1.amka and r1.course_code=r2.course_code and r1.register_status='pass' and r2.register_status='pass'
8   and r1.final_grade=r2.final_grade;
```

Data Output | Explain | Messages | Query History

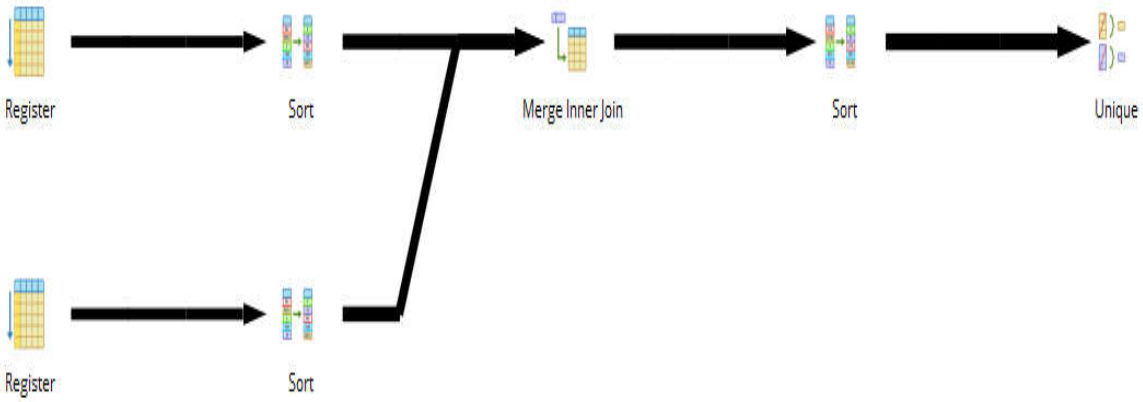
	QUERY PLAN text
15	-> Bitmap Index Scan on s_index (cost=0.00..197.31 rows=4975 width=0) (actual time=0.408..0.408 rows=...
16	Index Cond: (register_status = 'pass'::register_status_type)
17	-> Sort (cost=882.22..894.66 rows=4975 width=20) (actual time=17.470..19.164 rows=31920 loops=1)
18	Sort Key: r2.course_code, r2.final_grade
19	Sort Method: quicksort Memory: 579kB
20	-> Bitmap Heap Scan on "Register" r2 (cost=198.56..576.74 rows=4975 width=20) (actual time=0.331..1.512 ...
21	Recheck Cond: (register_status = 'pass'::register_status_type)
22	Heap Blocks: exact=82
23	-> Bitmap Index Scan on s_index (cost=0.00..197.31 rows=4975 width=0) (actual time=0.314..0.314 rows=...
24	Index Cond: (register_status = 'pass'::register_status_type)
25	Planning time: 0.949 ms
26	Execution time: 115.403 ms



Data Output | Explain | Messages | Query History

	QUERY PLAN text
1	Unique (cost=1648.11..1650.81 rows=216 width=28) (actual time=106.864..110.305 rows=13492 loops=1)
2	-> Sort (cost=1648.11..1648.65 rows=216 width=28) (actual time=106.863..107.635 rows=13492 loops=1)
3	Sort Key: r1.course_code, r1.final_grade, r1.amka, r2.amka
4	Sort Method: quicksort Memory: 1439kB
5	-> Merge Join (cost=1557.02..1639.73 rows=216 width=28) (actual time=35.772..50.467 rows=13492 loops=1)
6	Merge Cond: ((r1.course_code = r2.course_code) AND (r1.final_grade = r2.final_grade))
7	Join Filter: (r2.amka > r1.amka)
8	Rows Removed by Join Filter: 18438
9	-> Sort (cost=778.51..790.95 rows=4975 width=24) (actual time=19.739..20.326 rows=4946 loops=1)
10	Sort Key: r1.course_code, r1.final_grade
11	Sort Method: quicksort Memory: 579kB
12	-> Bitmap Heap Scan on "Register" r1 (cost=94.85..473.03 rows=4975 width=24) (actual time=0.544..2.649 r...
13	Recheck Cond: (register_status = 'pass'::register_status_type)
14	Heap Blocks: exact=82
15	-> Bitmap Index Scan on t_index (cost=0.00..93.60 rows=4975 width=0) (actual time=0.511..0.511 rows=4...
16	Index Cond: (register_status = 'pass'::register_status_type)
17	-> Sort (cost=778.51..790.95 rows=4975 width=20) (actual time=16.017..17.567 rows=31920 loops=1)
18	Sort Key: r2.course_code, r2.final_grade
19	Sort Method: quicksort Memory: 579kB
20	-> Bitmap Heap Scan on "Register" r2 (cost=94.85..473.03 rows=4975 width=20) (actual time=0.266..1.127 r...
21	Recheck Cond: (register_status = 'pass'::register_status_type)
22	Heap Blocks: exact=82
23	-> Bitmap Index Scan on t_index (cost=0.00..93.60 rows=4975 width=0) (actual time=0.251..0.251 rows=4...
24	Index Cond: (register_status = 'pass'::register_status_type)
25	Planning time: 1.019 ms
26	Execution time: 111.045 ms

CLUSTERING



```
5  CLUSTER "Register" USING reg_index;
6  EXPLAIN ANALYSE
7  SELECT DISTINCT r1.course_code,r1.register_status,r1.final_grade,r1.amka,r2.amka
8  FROM "Register" as r1, "Register" as r2
9  WHERE r2.amka>r1.amka and r1.course_code=r2.course_code and r1.register_status='pass' and r2.register_status='pass'
10  and r1.final_grade=r2.final_grade;
```

Data Output | Explain | Messages | Query History

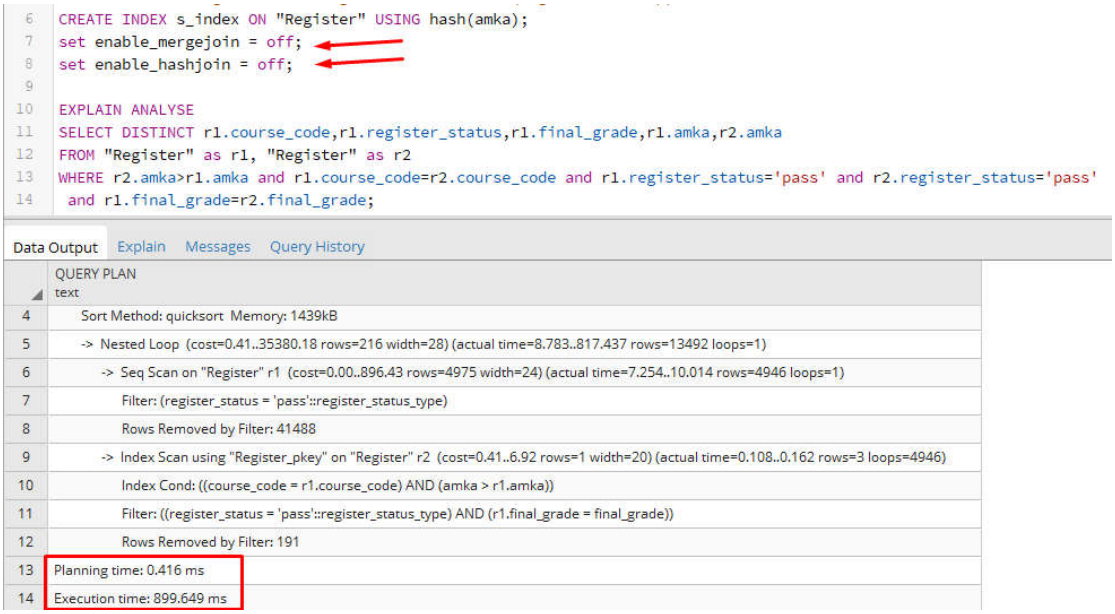
	QUERY PLAN text
16	Index Cond: (register_status = 'pass'::register_status_type)
17	-> Sort (cost=778.51..790.95 rows=4975 width=20) (actual time=20.957..23.492 rows=31920 loops=1)
18	Sort Key: r2.course_code, r2.final_grade
19	Sort Method: quicksort Memory: 579kB
20	-> Bitmap Heap Scan on "Register" r2 (cost=94.85..473.03 rows=4975 width=20) (actual time=0.476..1.798 rows=4946...
21	Recheck Cond: (register_status = 'pass'::register_status_type)
22	Heap Blocks: exact=42
23	-> Bitmap Index Scan on reg_index (cost=0.00..93.60 rows=4975 width=0) (actual time=0.456..0.456 rows=4946 loo...
24	Index Cond: (register_status = 'pass'::register_status_type)
25	Planning time: 0.962 ms
26	Execution time: 125.746 ms

Στους τελευταίους πειραματισμούς, δοκιμάσαμε την επιλεκτική απενεργοποίηση υπολογισμού συνδέσμων(joins) για την περαιτέρω κατανόηση της λειτουργίας του optimizer σχετικά με το πώς επιδρούν στα query plans .

```
7  set enable_mergejoin = off;
8  set enable_hashjoin = on;
9
10 --CLUSTER "Register" USING reg_index;
11 EXPLAIN ANALYSE
12 SELECT DISTINCT r1.course_code,r1.register_status,r1.final_grade,r1.amka,r2.amka
13 FROM "Register" as r1, "Register" as r2
14 WHERE r2.amka>r1.amka and r1.course_code=r2.course_code and r1.register_status='pass' and r2.register_status='pass'
15  and r1.final_grade=r2.final_grade;
```

Data Output | Explain | Messages | Query History

	QUERY PLAN text
8	Rows Removed by Join Filter: 18438
9	-> Seq Scan on "Register" r1 (cost=0.00..896.43 rows=4975 width=24) (actual time=7.134..8.761 rows=4946 loops=1)
10	Filter: (register_status = 'pass'::register_status_type)
11	Rows Removed by Filter: 41488
12	-> Hash (cost=896.43..896.43 rows=4975 width=20) (actual time=7.261..7.261 rows=4946 loops=1)
13	Buckets: 8192 Batches: 1 Memory Usage: 321kB
14	-> Seq Scan on "Register" r2 (cost=0.00..896.43 rows=4975 width=20) (actual time=4.199..5.827 rows=4946 loops=1)
15	Filter: (register_status = 'pass'::register_status_type)
16	Rows Removed by Filter: 41488
17	Planning time: 0.435 ms
18	Execution time: 90.089 ms



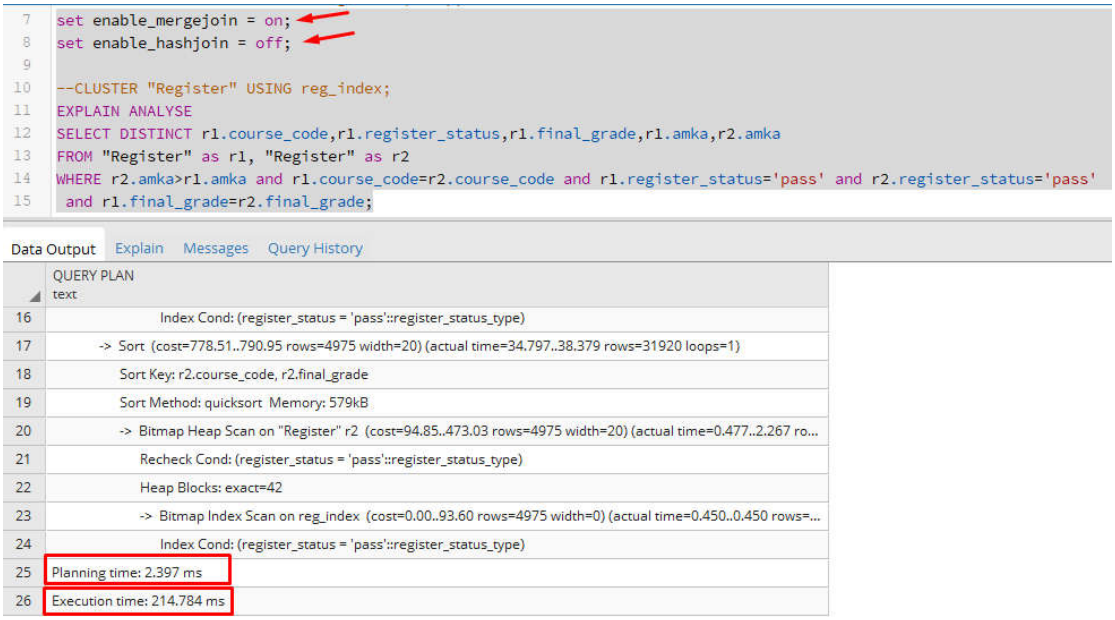
```
6  CREATE INDEX s_index ON "Register" USING hash(amka);
7  set enable_mergejoin = off;
8  set enable_hashjoin = off;
9
10 EXPLAIN ANALYSE
11 SELECT DISTINCT r1.course_code,r1.register_status,r1.final_grade,r1.amka,r2.amka
12 FROM "Register" as r1, "Register" as r2
13 WHERE r2.amka>r1.amka and r1.course_code=r2.course_code and r1.register_status='pass' and r2.register_status='pass'
14  and r1.final_grade=r2.final_grade;
```

Data Output | Explain | Messages | Query History

	QUERY PLAN text
4	Sort Method: quicksort Memory: 1439kB
5	-> Nested Loop (cost=0.41..35380.18 rows=216 width=28) (actual time=8.783..817.437 rows=13492 loops=1)
6	-> Seq Scan on "Register" r1 (cost=0.00..896.43 rows=4975 width=24) (actual time=7.254..10.014 rows=4946 loops=1)
7	Filter: (register_status = 'pass'::register_status_type)
8	Rows Removed by Filter: 41488
9	-> Index Scan using "Register_pkey" on "Register" r2 (cost=0.41..6.92 rows=1 width=20) (actual time=0.108..0.162 rows=3 loops=4946)
10	Index Cond: ((course_code = r1.course_code) AND (amka > r1.amka))
11	Filter: ((register_status = 'pass'::register_status_type) AND (r1.final_grade = final_grade))
12	Rows Removed by Filter: 191
13	Planning time: 0.416 ms
14	Execution time: 899.649 ms

Ακριβώς στην παραπάνω εικόνα, παρατηρούμε ότι ξαφνικά απότομα ο πραγματικός χρόνος εκτέλεσης του αιτήματος αυξήθηκε σε 899.6 ms!Επομένως, με την απενεργοποίηση των hash και merge join ταυτόχρονα στο συγκεκριμένο query δεν έχουμε κανένα συμφέρον ,αντιθέτως η απόδοση πέφτει δραματικά. Στην περίπτωση που απενεργοποιούμε μόνο το merge join και διατηρούμε το hash join από την άλλη επιτυγχάνουμε καλύτερη απόδοση καθώς ο πραγματικός χρόνος πέφτει από 134 ms σε 90ms και ο αναμενόμενος χρόνος από 0.8 ms σε 0.4ms.

CREATE INDEX reg_index ON "Register" USING btree(register_status);



```
7  set enable_mergejoin = on;
8  set enable_hashjoin = off;
9
10 --CLUSTER "Register" USING reg_index;
11 EXPLAIN ANALYSE
12 SELECT DISTINCT r1.course_code,r1.register_status,r1.final_grade,r1.amka,r2.amka
13 FROM "Register" as r1, "Register" as r2
14 WHERE r2.amka>r1.amka and r1.course_code=r2.course_code and r1.register_status='pass' and r2.register_status='pass'
15  and r1.final_grade=r2.final_grade;
```

Data Output | Explain | Messages | Query History

	QUERY PLAN text
16	Index Cond: (register_status = 'pass'::register_status_type)
17	-> Sort (cost=778.51..790.95 rows=4975 width=20) (actual time=34.797..38.379 rows=31920 loops=1)
18	Sort Key: r2.course_code, r2.final_grade
19	Sort Method: quicksort Memory: 579kB
20	-> Bitmap Heap Scan on "Register" r2 (cost=94.85..473.03 rows=4975 width=20) (actual time=0.477..2.267 ro...
21	Recheck Cond: (register_status = 'pass'::register_status_type)
22	Heap Blocks: exact=42
23	-> Bitmap Index Scan on reg_index (cost=0.00..93.60 rows=4975 width=0) (actual time=0.450..0.450 rows=...
24	Index Cond: (register_status = 'pass'::register_status_type)
25	Planning time: 2.397 ms
26	Execution time: 214.784 ms

Ένα ακόμη παράδειγμα που επισημαίνει ότι ο optimizer χρειαζόταν τα merge joins που απενεργοποιήσαμε, καθώς πάλι η απόδοση πέφτει.

(Planning Time: 0.9ms → 2.39ms

Execution Time: 117.3ms → 214.7ms)

Ολοκληρώνοντας και έπειτα από πολλές δοκιμές, πειραματιστήκαμε συνολικά σε πολλές μεθόδους όπως κατακερματισμός, δενδρικές δομές, ομαδοποίηση ,συνδυασμός μεθόδων(hash με b+tree), καθώς και αλλαγή των χαρακτηριστικών στα ευρετήρια για βέλτιστη απόδοση. Γενικά, τα ευρετήρια κατακερματισμού/hashing είναι αποδοτικά σε point queries(ισότητες),ενώ τα b+trees συμφέρουν κυρίως σε ερωτήσεις με εύρος τιμών,range queries(ανισότητες).Σύμφωνα με όλα τα παραπάνω, οδηγηθήκαμε στο συμπέρασμα ότι τις περισσότερες, κατά μέσο όρο φορές, η ομαδοποίηση είναι συμφέρουσα και η πιο αποδοτική μέθοδος για βελτιστοποίηση των αιτημάτων, ειδικά όταν ο όγκος των δεδομένων μεγαλώνει. Πολλές φορές, παρατηρήσαμε ότι ο βελτιστοποιητής αυτόματα παρέχει/καταλήγει σε βέλτιστα plans,επιλέγοντας πιο «έξυπνους» δρόμους(εντοπίζοντας αρχικά όλα τα alternative paths).