

## ΒΑΣΕΙΣ ΔΕΔΟΜΕΝΩΝ - ΠΛΗ302

### Β' ΦΑΣΗ ΕΡΓΑΣΤΗΡΙΑΚΗΣ ΕΡΓΑΣΙΑΣ

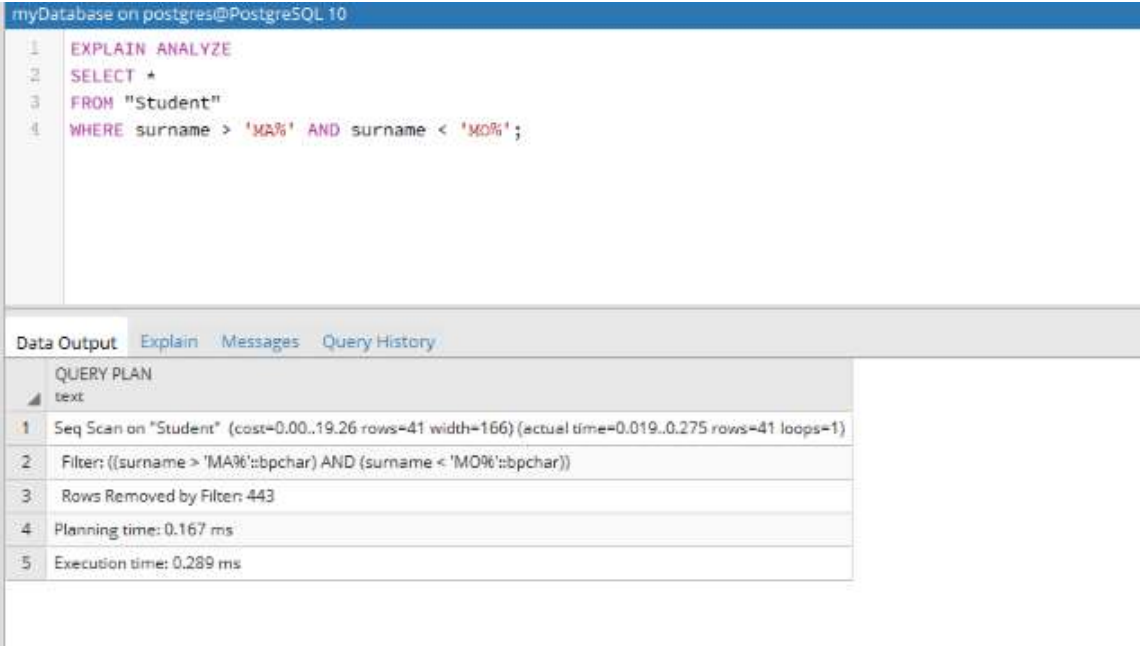
Ομάδα :

Κωνσταντίνα Μουδήρη - 2012030128

Αγγελική Χρήστενα - 2013030106

#### Μελέτη απόδοσης ερωτήσεων – φυσικός σχεδιασμός:

**A)** Αρχικά είχαμε να μελετήσουμε το εξής αίτημα: "Βρες τους φοιτητές που έχουν επώνυμο στο διάστημα αλφαριθμητικών από 'ΜΑ' έως 'ΜΟ' ". Ξεκινήσαμε λοιπόν με τη βάση που ήδη είχαμε από την α' φάση της εργαστηριακής άσκησης, στην οποία υπήρχαν λίγες εκατοντάδες φοιτητών. Κάνοντας χρήση της εντολής EXPLAIN ANALYZE μελετήσαμε το παραπάνω ερώτημα παίρνοντας τον χρόνο εκτέλεσής του, αρχικά **χωρίς** την χρήση κάποιου ευρετηρίου, και σημειώσαμε το παρακάτω αποτέλεσμα.



The screenshot shows a PostgreSQL query window with the following SQL code:

```
1 EXPLAIN ANALYZE
2 SELECT *
3 FROM "Student"
4 WHERE surname > 'MA%' AND surname < 'MO%';
```

Below the query, the 'Data Output' tab is selected, showing the 'QUERY PLAN' for the query. The plan details are as follows:

Step	Operation	Cost	Rows	Width	Actual Time	Loops
1	Seq Scan on "Student"	0.00..19.26	41	166	0.019..0.275	1
2	Filter: ((surname > 'MA%::bpchar) AND (surname < 'MO%::bpchar))					
3	Rows Removed by Filter: 443					
4	Planning time: 0.167 ms					
5	Execution time: 0.289 ms					

Εικόνα 1 Χωρίς Index

Στην συνέχεια για το ίδιο ερώτημα δοκιμάσαμε διαδοχικά να δημιουργήσουμε κατάλληλα ευρετήρια που θεωρήσαμε ότι θα μπορούσαν να επιταχύνουν την εκτέλεση του αιτήματος και μελετήσαμε εκ νέου το πλάνο εκτέλεσης.

Τα ευρετήρια που χρησιμοποιήθηκαν ήταν τύπου Btree και Hash.

Το Hash σαν είδος ευρετηρίου χρησιμοποιείται μόνο σε σχέσεις ισότητας σε αντίθεση με το Btree, που αν και όχι τόσο αποδοτικό όσο το Hash σε σχέσεις ισότητας μπορεί να χρησιμοποιηθεί σε μεγαλύτερο φάσμα περιπτώσεων, καθώς πιάνει και τις περιπτώσεις συγκρίσεων μεταξύ τιμών. Γενικότερα όμως η χρήση του Hash δεν συνίσταται.

Όπως φαίνεται και στο παρακάτω στιγμιότυπο αρχικά δημιουργήσαμε ένα ευρετήριο τύπου **Btree**, κατασκευασμένο ως προς το γνώρισμα surname, για το οποίο υπάρχει και η συνθήκη στο WHERE. Παρατηρούμε ότι ο χρόνος εκτέλεσης μετά τη χρήση του index έχει επιταχυνθεί, παρ' όλα αυτά όμως το αποτέλεσμά μας δεν μπορεί να είναι αντιπροσωπευτικό καθώς ο αριθμός των φοιτητών είναι αρκετά μικρός.



The screenshot shows a PostgreSQL query window with the following SQL commands:

```
1. EXPLAIN ANALYZE
2. SELECT *
3. FROM "Student"
4. WHERE surname > 'MA%' AND surname < 'MO%';
5.
6. CREATE INDEX student_surname_idx ON "Student" USING btree(surname);
```

Below the SQL editor, the 'Data Output' tab is selected, displaying the 'QUERY PLAN' for the executed query. The plan shows a 'Bitmap Heap Scan' on the 'Student' table, which involves a 'Recheck Cond' and a 'Heap Blocks' count. It then shows a 'Bitmap Index Scan' on the 'student\_surname\_idx' index, followed by an 'Index Cond'. The plan also includes 'Planning time' and 'Execution time'.

Step	Operation	Cost	Rows	Width	Actual Time	Loops
1	Bitmap Heap Scan on "Student"	(cost=4.69..17.31)	rows=41	width=166	(actual time=0.080..0.099)	rows=41 loops=1
2	Recheck Cond: ((surname > 'MA%'::bpchar) AND (surname < 'MO%'::bpchar))					
3	Heap Blocks: exact=12					
4	-> Bitmap Index Scan on student_surname_idx	(cost=0.00..4.68)	rows=41	width=0	(actual time=0.073..0.073)	rows=41 loops=1
5	Index Cond: ((surname > 'MA%'::bpchar) AND (surname < 'MO%'::bpchar))					
6	Planning time	0.209 ms				
7	Execution time	0.141 ms				

Εικόνα 2 Index B-tree

Έπειτα, δοκιμάσαμε να επιταχύνουμε το ερώτημα αξιοποιώντας τη δυνατότητα της ομαδοποίησης (clustering). Σε αυτό το σημείο, βλέπουμε πως οι χρόνοι στην περίπτωση του index τύπου Btree και του clustering είναι παραπλήσιοι, κι αυτό γιατί όταν οι εισαγωγές, όπως στη δική μας περίπτωση, είναι λίγες δεν μπορούμε να διακρίνουμε κάποια διαφορά με την χρήση του clustering.

myDatabase on postgres@PostgreSQL 10	
1	EXPLAIN ANALYZE
2	SELECT *
3	FROM "Student"
4	WHERE surname > 'MA%' AND surname < 'MO%';
5	
6	CREATE INDEX student_surname_idx ON "Student" USING btree(surname);
7	
8	CLUSTER "Student" USING student_surname_idx;

Data Output	Explain	Messages	Query History
<div> <div>QUERY PLAN</div> <div>text</div> <div> <div>1</div> <div>Bitmap Heap Scan on "Student" (cost=4.69..17.31 rows=41 width=166) (actual time=0.078..0.084 rows=41 loops=1)</div> <div>2</div> <div>Recheck Cond: ((surname &gt; 'MA% '::bpchar) AND (surname &lt; 'MO% '::bpchar))</div> <div>3</div> <div>Heap Blocks: exact=2</div> <div>4</div> <div>-&gt; Bitmap Index Scan on student_surname_idx (cost=0.00..4.68 rows=41 width=0) (actual time=0.071..0.071 rows=41 loops=1)</div> <div>5</div> <div>Index Cond: ((surname &gt; 'MA% '::bpchar) AND (surname &lt; 'MO% '::bpchar))</div> <div>6</div> <div>Planning time: 0.206 ms</div> <div>7</div> <div>Execution time: 0.121 ms</div> </div> </div>			

Εικόνα 3 Cluster Index B-tree

Επιπλέον, δοκιμάσαμε να δημιουργήσουμε ένα **Hash Index** κατασκευασμένο ως προς το attribute surname, έχοντας κάνει drop τα παραπάνω ευρετήρια που είχαμε δημιουργήσει και στην συνέχεια με την EXPLAIN ANALYZE είδαμε τον χρόνο εκτέλεσης του query. Παρατηρούμε πως σε αυτή την περίπτωση ο Query Optimizer δεν χρησιμοποιεί το υλοποιημένο ευρετήριο και έτσι ο χρόνος εκτέλεσης είναι παρόμοιος με τον αρχικό. Ο λόγος είναι ότι τα Hash Tables προτιμώνται για την εύρεση μίας εγγραφής κι όχι εύρους τιμών, όπως στην περίπτωση μας.

myDatabase on postgres@PostgreSQL 10	
1	EXPLAIN ANALYZE
2	SELECT *
3	FROM "Student"
4	WHERE surname > 'MA%' AND surname < 'MO%';
5	
6	CREATE INDEX student_name_idx ON "Student" USING hash(surname);

Data Output	Explain	Messages	Query History
<div> <div>QUERY PLAN</div> <div>text</div> <div> <div>1</div> <div>Seq Scan on "Student" (cost=0.00..19.26 rows=41 width=166) (actual time=0.100..0.263 rows=41 loops=1)</div> <div>2</div> <div>Filter: ((surname &gt; 'MA% '::bpchar) AND (surname &lt; 'MO% '::bpchar))</div> <div>3</div> <div>Rows Removed by Filter: 443</div> <div>4</div> <div>Planning time: 0.129 ms</div> <div>5</div> <div>Execution time: 0.284 ms</div> </div> </div>			

Εικόνα 4 Hash Index

Τέλος, αυξήσαμε το πλήθος των φοιτητών στον πίνακα της βάσης σας 100.000 φοιτητές, χρησιμοποιώντας συναρτήσεις εισαγωγής όπως και προηγουμένως. Σβήσαμε τα ευρετήρια που δημιουργήσαμε μέχρι τώρα και μελετήσαμε εκ νέου τα πλάνα εκτέλεσης πριν και μετά τη δημιουργία ευρετηρίων.

Αρχικά εκτελέσαμε το ερώτημα **χωρίς** την χρήση κάποιου ευρετηρίου, και σημειώσαμε το παρακάτω αποτέλεσμα. Παρατηρούμε πως ο query optimizer κάνει sequential scan για να βρει στον πίνακα "Student".

myDatabase on postgres@PostgreSQL 10	
1	EXPLAIN ANALYZE
2	SELECT *
3	FROM "Student"
4	WHERE surname > 'MA%' AND surname < 'MO%';

Data Output	Explain	Messages	Query History
QUERY PLAN			
text			
1	Seq Scan on "Student" (cost=0.00..3951.94 rows=6454 width=166) (actual time=0.183..58.839 rows=6222 loops=1)		
2	Filter: ((surname > 'MA% '::bpchar) AND (surname < 'MO% '::bpchar))		
3	Rows Removed by Filter: 93778		
4	Planning time: 0.342 ms		
5	Execution time: 58.999 ms		

Εικόνα 5 Χωρίς Index

Έπειτα, δημιουργήσαμε ένα ευρετήριο τύπου Btree, κατασκευασμένο ως προς το γνώρισμα surname όπως και προηγουμένως, για το οποίο υπάρχει και η συνθήκη στο WHERE. Περιμένουμε να μειωθεί ο χρόνος εκτέλεσης του query, καθώς γνωρίζουμε πως τα Btrees αποτελούν μια καλή επιλογή ευρετηρίων για αναζήτηση εύρους τιμών. Παρατηρούμε ότι όντως ο χρόνος εκτέλεσης μετά τη χρήση του index έχει επιταχυνθεί σημαντικά.

myDatabase on postgres@PostgreSQL 10

1

EXPLAIN ANALYZE

2

SELECT \*

3

FROM "Student"

4

WHERE surname > 'MA%' AND surname < 'MO%';

5

6

CREATE INDEX student\_surname\_idx ON "Student" USING btree(surname);

Data Output

Explain

Messages

Query History

QUERY PLAN

text

1

Bitmap Heap Scan on "Student" (cost=266.57..2815.38 rows=6454 width=166) (actual time=4.998..7.144 rows=62...

2

Recheck Cond: ((surname > 'MA% '::bpchar) AND (surname < 'MO% '::bpchar))

3

Heap Blocks: exact=2281

4

-> Bitmap Index Scan on student\_surname\_idx (cost=0.00..264.96 rows=6454 width=0) (actual time=4.681..4.681 ...

5

Index Cond: ((surname > 'MA% '::bpchar) AND (surname < 'MO% '::bpchar))

6

Planning time: 0.349 ms

7

Execution time: 7.332 ms

Εικόνα 6 Index B-tree

Έπειτα, δοκιμάσαμε να επιταχύνουμε περαιτέρω το ερώτημα αξιοποιώντας τη δυνατότητα της ομαδοποίησης (clustering). Σε αυτό το σημείο, βλέπουμε πως οι χρόνοι στην περίπτωση του index τύπου Btree και του clustering είναι παραπλήσιοι με τον χρόνο εκτέλεσης λόγω clustering να έχει επιταχυνθεί ακόμη περισσότερο.

myDatabase on postgres@PostgreSQL 10	
1	EXPLAIN ANALYZE
2	SELECT *
3	FROM "Student"
4	WHERE surname > 'MA%' AND surname < 'MO%';
5	
6	CREATE INDEX student_surname_idx ON "Student" USING btree(surname);
7	
8	CLUSTER "Student" USING student_surname_idx;
Data Output Explain Messages Query History	
	QUERY PLAN text
1	Bitmap Heap Scan on "Student" (cost=266.57..2821.38 rows=6454 width=166) (actual time=4.508..5.121 rows=62...
2	Recheck Cond: ((surname > 'MA% '::bpchar) AND (surname < 'MO% '::bpchar))
3	Heap Blocks: exact=154
4	-> Bitmap Index Scan on student_surname_idx (cost=0.00..264.96 rows=6454 width=0) (actual time=4.486..4.486 ...
5	Index Cond: ((surname > 'MA% '::bpchar) AND (surname < 'MO% '::bpchar))
6	Planning time: 0.252 ms
7	Execution time: 5.279 ms

Εικόνα 7 Cluster Index B-tree

Σε αυτό το βήμα, χρησιμοποιήσαμε την τεχνική της ομαδοποίησης (clustering) χρησιμοποιώντας το κλειδί του πίνακα "Student". Βλέπουμε πως ο χρόνος στην περίπτωση αυτή είναι παραπλήσιος του χρόνου με την υλοποίηση του Btree.

myDatabase on postgres@PostgreSQL 10	
1	EXPLAIN ANALYZE
2	SELECT *
3	FROM "Student"
4	WHERE surname > 'MA%' AND surname < 'MO%';
5	
6	CREATE INDEX student_surname_idx ON "Student" USING btree(surname);
7	
8	CLUSTER "Student" USING "Student_pkey";
Data Output Explain Messages Query History	
	QUERY PLAN text
1	Bitmap Heap Scan on "Student" (cost=266.57..2816.38 rows=6454 width=166) (actual time=4.804..6.854 rows=62...
2	Recheck Cond: ((surname > 'MA% '::bpchar) AND (surname < 'MO% '::bpchar))
3	Heap Blocks: exact=2295
4	-> Bitmap Index Scan on student_surname_idx (cost=0.00..264.96 rows=6454 width=0) (actual time=4.499..4.499 ...
5	Index Cond: ((surname > 'MA% '::bpchar) AND (surname < 'MO% '::bpchar))
6	Planning time: 0.245 ms
7	Execution time: 7.103 ms

Εικόνα 8 Cluster using PK, B-tree

Τέλος, δοκιμάσαμε να δημιουργήσουμε ένα Hash Index κατασκευασμένο ως προς το attribute surname, έχοντας κάνει drop τα παραπάνω ευρετήρια που είχαμε δημιουργήσει και στην συνέχεια με την EXPLAIN ANALYSE είδαμε τον χρόνο εκτέλεσης του query. Είχαμε όπως και πριν κατά νου ότι το Hash Index δεν αποτελεί μια καλή επιλογή για το ερώτημά μας. Παρατηρούμε πως σε αυτή την περίπτωση ο χρόνος εκτέλεσης είναι κοντά στον αρχικό χρόνο που είχαμε χωρίς index, και αυτό επειδή το σύστημα δεν χρησιμοποιεί τον index που έχουμε φτιάξει, επειδή δεν θα ήταν αποδοτικός.

myDatabase on postgres@PostgreSQL 10	
1	EXPLAIN ANALYZE
2	SELECT *
3	FROM "Student"
4	WHERE surname > 'MA%' AND surname < 'MO%';
5	
6	CREATE INDEX student_surname_idx ON "Student" USING hash(surname);
7	
8	DROP INDEX student_surname_idx;

Data Output	Explain	Messages	Query History
QUERY PLAN			
text			
1	Seq Scan on "Student" (cost=0.00..3953.00 rows=6454 width=166) (actual time=0.040..60.023 rows=6222 loops=1)		
2	Filter: ((surname > 'MA%::bpchar) AND (surname < 'MO%::bpchar))		
3	Rows Removed by Filter: 93778		
4	Planning time: 0.412 ms		
5	Execution time: 60.191 ms		

Εικόνα 9 Hash Index

**Β)** Στο Β ερώτημα μελετήσαμε το εξής αίτημα: "Βρες ζεύγη κωδικών φοιτητών τέτοια ώστε σε κάθε τέτοιο ζεύγος οι δύο φοιτητές να έχουν περάσει με τον ίδιο βαθμό κάποιο μάθημα". Όπως και στο προηγούμενο ζήτημα, έτσι και εδώ, φροντίσαμε να έχουμε ήδη στη βάση μας αρκετές δεκάδες χιλιάδες εγγραφές φοιτητών στον πίνακα "Register" (η αρχική βάση που μας είχε δοθεί με την εκφώνηση της α' φάσης της εργαστηριακής εργασίας είχε ήδη πάνω από 40 χιλιάδες εγγραφές στον συγκεκριμένο πίνακα). Έτσι αρχικά δεν χρειάστηκε να εισάγουμε νέες εγγραφές. Κάνοντας χρήση της εντολής EXPLAIN ANALYSE εξετάσαμε την απόδοση του ερωτήματος, αρχικά **χωρίς** την χρήση κάποιου ευρετηρίου, και σημειώσαμε το παρακάτω αποτέλεσμα.

myDatabase on postgres@PostgreSQL 10	
1	EXPLAIN ANALYZE
2	SELECT r1.amka, r2.amka
3	FROM "Register" r1, "Register" r2
4	WHERE r1.final_grade=r2.final_grade AND r1.course_code = r2.course_code
Data Output Explain Messages Query History	
QUERY PLAN	
text	
1	Merge Join (cost=9430.19..30974.54 rows=1212260 width=8) (actual time=216.995..893.842 rows=2108948 loops=1)
2	Merge Cond: ((r1.final_grade = r2.final_grade) AND (r1.course_code = r2.course_code))
3	-> Sort (cost=4715.09..4825.03 rows=43974 width=19) (actual time=110.347..112.680 rows=20185 loops=1)
4	Sort Key: r1.final_grade, r1.course_code
5	Sort Method: quicksort Memory: 3905kB
6	-> Seq Scan on "Register" r1 (cost=0.00..1323.74 rows=43974 width=19) (actual time=0.020..6.696 rows=43974 loops=1)
7	-> Sort (cost=4715.09..4825.03 rows=43974 width=19) (actual time=106.637..193.235 rows=2108949 loops=1)
8	Sort Key: r2.final_grade, r2.course_code
9	Sort Method: quicksort Memory: 3905kB
10	-> Seq Scan on "Register" r2 (cost=0.00..1323.74 rows=43974 width=19) (actual time=0.030..5.601 rows=43974 loops=1)
11	Planning time: 0.749 ms
12	Execution time: 945.530 ms

Εικόνα 10 Χωρίς Index

Παρατηρούμε ότι ο χρόνος εκτέλεσης είναι αρκετά μεγάλος. Γι' αυτό το λόγο αποφασίσαμε να δημιουργήσουμε ένα Btree index στον πίνακα "Register" πάνω στο attribute final\_grade. Ωστόσο, βλέπουμε στο παρακάτω στιγμιότυπο, ότι δεν υπήρξε καμία βελτίωση ως προς τον χρόνο εκτέλεσης του αιτήματος, πράγμα που ήταν αναμενόμενο. Αλλά επίσης παρατηρούμε ότι δεν χρησιμοποιείται ο index που κατασκευάσαμε κατά την αναζήτηση.

myDatabase on postgres@PostgreSQL 10	
1	EXPLAIN ANALYZE
2	SELECT r1.amka, r2.amka
3	FROM "Register" r1, "Register" r2
4	WHERE r1.final_grade=r2.final_grade AND r1.course_code = r2.course_code
5	
6	CREATE INDEX final_grade_idx ON "Register" USING btree(final_grade);
Data Output Explain Messages Query History	
QUERY PLAN	
text	
1	Merge Join (cost=9430.19..30974.54 rows=1212260 width=8) (actual time=219.770..895.345 rows=2108948 loops=1)
2	Merge Cond: ((r1.final_grade = r2.final_grade) AND (r1.course_code = r2.course_code))
3	-> Sort (cost=4715.09..4825.03 rows=43974 width=19) (actual time=118.256..120.675 rows=20185 loops=1)
4	Sort Key: r1.final_grade, r1.course_code
5	Sort Method: quicksort Memory: 3905kB
6	-> Seq Scan on "Register" r1 (cost=0.00..1323.74 rows=43974 width=19) (actual time=0.016..6.122 rows=43974 loops=1)
7	-> Sort (cost=4715.09..4825.03 rows=43974 width=19) (actual time=101.501..186.333 rows=2108949 loops=1)
8	Sort Key: r2.final_grade, r2.course_code
9	Sort Method: quicksort Memory: 3905kB
10	-> Seq Scan on "Register" r2 (cost=0.00..1323.74 rows=43974 width=19) (actual time=0.027..6.624 rows=43974 loops=1)
11	Planning time: 0.573 ms
12	Execution time: 947.773 ms

Εικόνα 11 Index B-tree (final grade)



Έπειτα δημιουργήσαμε ένα hash index στον πίνακα Register πάνω στο final\_grade, συνεχίζοντας ωστόσο να παίρνουμε μεγάλους χρόνους εκτέλεσης καθώς επίσης και πάλι δεν χρησιμοποιείται από το σύστημα ο hash index, όπως βλέπουμε παρακάτω:

myDatabase on postgres@PostgreSQL 10	
1	EXPLAIN ANALYZE
2	SELECT r1.amka, r2.amka
3	FROM "Register" r1, "Register" r2
4	WHERE r1.final_grade=r2.final_grade AND r1.course_code = r2.course_code
5	
6	CREATE INDEX final_grade_idx ON "Register" USING hash(final_grade);
7	
Data Output Explain Messages Query History	
QUERY PLAN	
text	
1	Merge Join (cost=9430.19..30974.54 rows=1212260 width=8) (actual time=232.489..898.747 rows=2108948 loops=1)
2	Merge Cond: ((r1.final_grade = r2.final_grade) AND (r1.course_code = r2.course_code))
3	-> Sort (cost=4715.09..4825.03 rows=43974 width=19) (actual time=118.068..120.424 rows=20185 loops=1)
4	Sort Key: r1.final_grade, r1.course_code
5	Sort Method: quicksort Memory: 3905kB
6	-> Seq Scan on "Register" r1 (cost=0.00..1323.74 rows=43974 width=19) (actual time=0.028..6.531 rows=43974 loops=1)
7	-> Sort (cost=4715.09..4825.03 rows=43974 width=19) (actual time=114.407..199.248 rows=2108949 loops=1)
8	Sort Key: r2.final_grade, r2.course_code
9	Sort Method: quicksort Memory: 3905kB
10	-> Seq Scan on "Register" r2 (cost=0.00..1323.74 rows=43974 width=19) (actual time=0.036..8.150 rows=43974 loops=1)
11	Planning time: 0.631 ms
12	Execution time: 949.864 ms

Εικόνα 12 Index Hash (final grade)

Μη βλέποντας κάποια βελτίωση στους χρόνους εκτέλεσης, δημιουργήσαμε ένα ευρετήριο τύπου Hash πάνω στον πίνακα "Register" ως προς το κλειδί του πίνακα, course\_code, καθώς γνωρίζουμε πως πολύ συχνά το πιο χρήσιμο ευρετήριο για μια σχέση είναι ένα ευρετήριο ως προς το κλειδί της σχέσης(εικόνα 13). Δυστυχώς και πάλι βλέπουμε ότι ο index δεν χρησιμοποιείται. Στην συνέχεια με τον ίδιο τρόπο φτιάξαμε index B-tree αλλά και πάλι το αποτέλεσμα ήταν το ίδιο (εικόνα 14).

Τα ίδια αποτελέσματα έχει και παρακάτω η ομαδοποίηση (clustering) χρησιμοποιώντας το κλειδί της σχέσης "Register". Σε αυτή την περίπτωση γνωρίζουμε πως μόνο σε μαζικές εισαγωγές θα βλέπαμε διαφορά με την χρήση ενός cluster ευρετηρίου (εικόνα 15).

myDatabase on postgres@PostgreSQL 10	
1	EXPLAIN ANALYZE
2	SELECT r1.amka, r2.amka
3	FROM "Register" r1, "Register" r2
4	WHERE r1.final_grade=r2.final_grade AND r1.course_code = r2.course_code
5	
6	CREATE INDEX final_grade_idx ON "Register" USING hash(course_code);
7	
Data Output Explain Messages Query History	
QUERY PLAN	
text	
1	Merge Join (cost=9430.19..30974.54 rows=1212260 width=8) (actual time=206.794..857.572 rows=2108948 loops=1)
2	Merge Cond: ((r1.final_grade = r2.final_grade) AND (r1.course_code = r2.course_code))
3	-> Sort (cost=4715.09..4825.03 rows=43974 width=19) (actual time=105.991..108.507 rows=20185 loops=1)
4	Sort Key: r1.final_grade, r1.course_code
5	Sort Method: quicksort Memory: 3905kB
6	-> Seq Scan on "Register" r1 (cost=0.00..1323.74 rows=43974 width=19) (actual time=0.018..6.941 rows=43974 loops=1)
7	-> Sort (cost=4715.09..4825.03 rows=43974 width=19) (actual time=100.792..183.210 rows=2108949 loops=1)
8	Sort Key: r2.final_grade, r2.course_code
9	Sort Method: quicksort Memory: 3905kB
10	-> Seq Scan on "Register" r2 (cost=0.00..1323.74 rows=43974 width=19) (actual time=0.026..5.894 rows=43974 loops=1)
11	Planning time: 0.612 ms
12	Execution time: 909.049 ms

Εικόνα 13 Index Hash (course code)

myDatabase on postgres@PostgreSQL 10	
1	EXPLAIN ANALYZE
2	SELECT r1.amka, r2.amka
3	FROM "Register" r1, "Register" r2
4	WHERE r1.final_grade=r2.final_grade AND r1.course_code = r2.course_code
5	
6	CREATE INDEX final_grade_idx ON "Register" USING btree(course_code);
7	
Data Output Explain Messages Query History	
QUERY PLAN	
text	
1	Merge Join (cost=9430.19..30974.54 rows=1212260 width=8) (actual time=280.741..935.775 rows=2108948 loops=1)
2	Merge Cond: ((r1.final_grade = r2.final_grade) AND (r1.course_code = r2.course_code))
3	-> Sort (cost=4715.09..4825.03 rows=43974 width=19) (actual time=124.092..126.465 rows=20185 loops=1)
4	Sort Key: r1.final_grade, r1.course_code
5	Sort Method: quicksort Memory: 3905kB
6	-> Seq Scan on "Register" r1 (cost=0.00..1323.74 rows=43974 width=19) (actual time=0.026..9.906 rows=43974 loops=1)
7	-> Sort (cost=4715.09..4825.03 rows=43974 width=19) (actual time=156.635..238.541 rows=2108949 loops=1)
8	Sort Key: r2.final_grade, r2.course_code
9	Sort Method: quicksort Memory: 3905kB
10	-> Seq Scan on "Register" r2 (cost=0.00..1323.74 rows=43974 width=19) (actual time=0.039..7.339 rows=43974 loops=1)
11	Planning time: 0.637 ms
12	Execution time: 984.834 ms

Εικόνα 14 Index B-tree (course code)

myDatabase on postgres@PostgreSQL 10	
1	EXPLAIN ANALYZE
2	SELECT r1.amka, r2.amka
3	FROM "Register" r1, "Register" r2
4	WHERE r1.final_grade=r2.final_grade AND r1.course_code = r2.course_code
5	
6	CLUSTER "Register" USING "Register_pkey"
7	
Data Output Explain Messages Query History	
QUERY PLAN	
text	
1	Merge Join (cost=8304.19..29848.54 rows=1212260 width=8) (actual time=201.208..883.968 rows=2108948 loops=1)
2	Merge Cond: ((r1.final_grade = r2.final_grade) AND (r1.course_code = r2.course_code))
3	-> Sort (cost=4152.09..4262.03 rows=43974 width=19) (actual time=105.989..108.526 rows=20185 loops=1)
4	Sort Key: r1.final_grade, r1.course_code
5	Sort Method: quicksort Memory: 3828kB
6	-> Seq Scan on "Register" r1 (cost=0.00..760.74 rows=43974 width=19) (actual time=0.022..8.673 rows=43974 loops=1)
7	-> Sort (cost=4152.09..4262.03 rows=43974 width=19) (actual time=95.207..179.166 rows=2108949 loops=1)
8	Sort Key: r2.final_grade, r2.course_code
9	Sort Method: quicksort Memory: 3828kB
10	-> Seq Scan on "Register" r2 (cost=0.00..760.74 rows=43974 width=19) (actual time=0.027..6.511 rows=43974 loops=1)
11	Planning time: 0.666 ms
12	Execution time: 936.565 ms

Εικόνα 15 Cluster PK

Επιπλέον παρακάτω δοκιμάσαμε να απενεργοποιήσουμε επιλεκτικά αλγορίθμους υπολογισμού συνδέσεων και να καταγράψουμε τις παρατηρήσεις μας, όσον αφορά την αλλαγή των πλάνων εκτέλεσης και της απόδοσής τους.

Αρχικά, θέσαμε σε κατάσταση off το merge join, και χωρίς την χρήση κάποιου ευρετηρίου παρατηρήσαμε σημαντική βελτίωση στον χρόνο εκτέλεσης του αιτήματος.

myDatabase on postgres@PostgreSQL 10	
1	EXPLAIN ANALYZE
2	SELECT r1.amka, r2.amka
3	FROM "Register" r1, "Register" r2
4	WHERE r1.final_grade=r2.final_grade AND r1.course_code = r2.course_code
5	
6	set enable_mergejoin=off;
7	
Data Output Explain Messages Query History	
QUERY PLAN	
text	
1	Hash Join (cost=1420.35..75867.29 rows=1212260 width=8) (actual time=12.389..538.899 rows=2108948 loops=1)
2	Hash Cond: ((r1.final_grade = r2.final_grade) AND (r1.course_code = r2.course_code))
3	-> Seq Scan on "Register" r1 (cost=0.00..760.74 rows=43974 width=19) (actual time=0.013..4.125 rows=43974 loops=1)
4	-> Hash (cost=760.74..760.74 rows=43974 width=19) (actual time=12.158..12.158 rows=20184 loops=1)
5	Buckets: 65536 Batches: 1 Memory Usage: 1523kB
6	-> Seq Scan on "Register" r2 (cost=0.00..760.74 rows=43974 width=19) (actual time=0.007..6.036 rows=43974 loops=1)
7	Planning time: 0.153 ms
8	Execution time: 588.426 ms

Εικόνα 16 SET merge join: off

Συνεχίσαμε, με απενεργοποιημένο το join, κατασκευάζοντας ένα B-tree Index ως προς το final\_grade. Ωστόσο είδαμε πως δεν χρησιμοποιείται το ευρετήριο από τον Query Optimizer, με αποτέλεσμα να μην παίρνουμε κάποιο καλύτερο χρόνο εκτέλεσης.

myDatabase on postgres@PostgreSQL 10	
1	EXPLAIN ANALYZE
2	SELECT r1.amka, r2.amka
3	FROM "Register" r1, "Register" r2
4	WHERE r1.final_grade=r2.final_grade AND r1.course_code = r2.course_code
5	
6	set enable_mergejoin=off;
7	
8	CREATE INDEX pairs_idx ON "Register" USING btree(final_grade);
9	
10	
11	
12	

Data Output	Explain	Messages	Query History
<div> <div>QUERY PLAN</div> <div>text</div> <div> 1 Hash Join (cost=1420.35..75867.29 rows=1212260 width=8) (actual time=18.126..577.882 rows=2108948 loops=1)  2 Hash Cond: ((r1.final_grade = r2.final_grade) AND (r1.course_code = r2.course_code))  3 -&gt; Seq Scan on "Register" r1 (cost=0.00..760.74 rows=43974 width=19) (actual time=0.014..4.605 rows=43974 loops=1)  4 -&gt; Hash (cost=760.74..760.74 rows=43974 width=19) (actual time=17.873..17.873 rows=20184 loops=1)  5 Buckets: 65536 Batches: 1 Memory Usage: 1523kB  6 -&gt; Seq Scan on "Register" r2 (cost=0.00..760.74 rows=43974 width=19) (actual time=0.010..8.676 rows=43974 loops=1)  7 Planning time: 0.622 ms  8 Execution time: 630.058 ms </div> </div>			

Εικόνα 17 merge join: off, B-tree index

Συνεχίσαμε, με απενεργοποιημένο το join, κατασκευάζοντας ένα Hash Index ως προς το final\_grade, χωρίς ωστόσο να τον χρησιμοποιεί το σύστημα κατά την εκτέλεση του query.

myDatabase on postgres@PostgreSQL 10	
1	EXPLAIN ANALYZE
2	SELECT r1.amka, r2.amka
3	FROM "Register" r1, "Register" r2
4	WHERE r1.final_grade=r2.final_grade AND r1.course_code = r2.course_code
5	
6	set enable_mergejoin=off;
7	
8	CREATE INDEX pairs_idx ON "Register" USING hash(final_grade);
9	
10	
11	
12	

Data Output	Explain	Messages	Query History
<div> <div>QUERY PLAN</div> <div>text</div> <div> 1 Hash Join (cost=1420.35..75867.29 rows=1212260 width=8) (actual time=14.767..549.402 rows=2108948 loops=1)  2 Hash Cond: ((r1.final_grade = r2.final_grade) AND (r1.course_code = r2.course_code))  3 -&gt; Seq Scan on "Register" r1 (cost=0.00..760.74 rows=43974 width=19) (actual time=0.012..4.721 rows=43974 loops=1)  4 -&gt; Hash (cost=760.74..760.74 rows=43974 width=19) (actual time=14.537..14.537 rows=20184 loops=1)  5 Buckets: 65536 Batches: 1 Memory Usage: 1523kB  6 -&gt; Seq Scan on "Register" r2 (cost=0.00..760.74 rows=43974 width=19) (actual time=0.008..7.277 rows=43974 loops=1)  7 Planning time: 0.175 ms  8 Execution time: 599.765 ms </div> </div>			

Εικόνα 18 merge join: off, hash index

Τέλος, έχουμε απενεργοποιημένο μόνο το hash join, έχοντας κάνει drop σε όλα τα προηγούμενα ευρετήρια που είχαμε υλοποιήσει. Παρατηρούμε ότι ο χρόνος εκτέλεσης είναι παραπλήσιος με αυτόν των αρχικών δοκιμών που κάναμε για το αίτημα αυτό.

myDatabase on postgres@PostgreSQL 10	
1	EXPLAIN ANALYZE
2	SELECT r1.amka, r2.amka
3	FROM "Register" r1, "Register" r2
4	WHERE r1.final_grade=r2.final_grade AND r1.course_code = r2.course_code
5	
6	
7	set enable_hashjoin=off;
8	
9	
Data Output Explain Messages Query History	
QUERY PLAN	
text	
1	Merge Join (cost=8304.19..29848.54 rows=1212260 width=8) (actual time=206.865..872.140 rows=2108948 loops=1)
2	Merge Cond: ((r1.final_grade = r2.final_grade) AND (r1.course_code = r2.course_code))
3	-> Sort (cost=4152.09..4262.03 rows=43974 width=19) (actual time=95.810..97.950 rows=20185 loops=1)
4	Sort Key: r1.final_grade, r1.course_code
5	Sort Method: quicksort Memory: 3828kB
6	-> Seq Scan on "Register" r1 (cost=0.00..760.74 rows=43974 width=19) (actual time=0.012..6.018 rows=43974 loops=1)
7	-> Sort (cost=4152.09..4262.03 rows=43974 width=19) (actual time=111.045..193.671 rows=2108949 loops=1)
8	Sort Key: r2.final_grade, r2.course_code
9	Sort Method: quicksort Memory: 3828kB
10	-> Seq Scan on "Register" r2 (cost=0.00..760.74 rows=43974 width=19) (actual time=0.026..5.968 rows=43974 loops=1)
11	Planning time: 0.541 ms
12	Execution time: 923.705 ms

Εικόνα 20 hash join: off

Γενικά τα συμπεράσματα που βγάλαμε από αυτό το κομμάτι της άσκησης είναι ότι για την αναζήτηση ενός εύρους τιμών είναι πολύ χρήσιμος ένας B-tree index. Για το δεύτερο κομμάτι που αφορούσε την αναζήτηση ίδιων τιμών ανάμεσα σε φοιτητές περιμέναμε να δούμε βελτίωση με την χρήση του hash index, παρ' όλα αυτά είδαμε ότι το σύστημα επιλέγει να μην χρησιμοποιήσει τον συγκεκριμένο index και έτσι δεν βλέπουμε καμία διαφορά στα αποτελέσματά μας. Η μόνη διαφορά που είδαμε σε αυτό το κομμάτι, η οποία ελάττωσε σημαντικά τον χρόνο εκτέλεσης, ήταν όταν απενεργοποιήσαμε το merge join.