

Σχολή Ηλεκτρολόγων Υπολονιστών

Βάσεις Δεδομένων

Φάση Α και φυσικός σχεδιασμός

Ομάδα LAB30244760

Τρίμας Χρήστος	2016030054
Παντελής Κωσταντίνος	2015030070

Σκοπός εργαστηριακής άσκησης:

Σκοπός της εργασίας είναι η εξοικείωση των φοιτητών με τις σχεσιακές βάσεις δεδομένων και με την γλώσσα PostgresSql, όπως και το περιβάλλον pgadmin. Η παρούσα αναφορά, περιλαμβάνει την αντιστοιχία των συναρτήσεων που υλοποιήθηκαν με τα ερωτήματα της Α φάσης, καθώς επίσης και την μελέτη απόδοσης των indexes που δημιουργήθηκαν κατά την φάση του φυσικού σχεδιασμού της βάσης.

Φάση Α:

Διαχείριση δεδομένων μέρος 3:

3.1 → χωρίστηκε σε 3 συναρτήσεις. insert_professors_3_1(), insert_students_3_1(), insert_labstaff_3_1().

 $3.2 \rightarrow \text{calc_final_grade_}3_2().$

Ανάκτηση δεδομένων και υπολογισμοί μέρος 4:

 $4.1 \rightarrow$ χωρίστηκε σε 2 συναρτήσεις. find_4_1_lstaff(), find_4_1_prof().

 $4.2 \rightarrow \text{prof}_4_2\text{-office_hours()}.$

 $4.3 \rightarrow \text{find}_4_3_\text{max_grades}()$.

 $4.4 \rightarrow \text{find}_4_4_\text{curr}_\text{c_r}$ ().

 $4.5 \rightarrow \text{find}_4_5_\text{after}()$.

 $4.6 \rightarrow \text{find}_4_6_\text{lab}()$.

 $4.7 \rightarrow \text{find}_4_7\text{_workload}()$.

4.8 → find_4_8_room_dif_sub().

 $4.9 \rightarrow \text{find}_4_9 \text{-operation}()$.

Λειτουργικότητα με κατασκευή εναυσμάτων μέρος 5:

- $\underline{5.1} \rightarrow \Sigma \nu \nu \alpha \rho \tau \dot{\eta} \sigma \epsilon \iota \varsigma$: check_participant_participation_5_1(), get_person_participation_5_1(), get_lab_activities_5_1(), get_course_lab_hours_5_1(), check_responsible_participation_5_1() και **triggers**: inspect_participant_participation_5_1, inspect_responsible_participation_5_1.
- $\underline{5.2} \rightarrow \Sigma \nu \nu \alpha \rho \tau \dot{\eta} \sigma \epsilon \iota \varsigma$: check_activity_time_constrains_5_2(), check_activity_schedule_constrains_5_2(), triggers: notify_on_illegal_activity_input_5_2, notify_on_illegal_activity_sched_5_2.
- $5.3 \rightarrow \Sigma \nu \nu \alpha \rho \tau \dot{\eta} \sigma \epsilon \iota \varsigma$: copy_existent_courserun_5_3(), insert_courserun_5_3(), triggers: create_future_courses_5_3.

Λειτουργικότητα με χρήση όψεων μέρος 6:

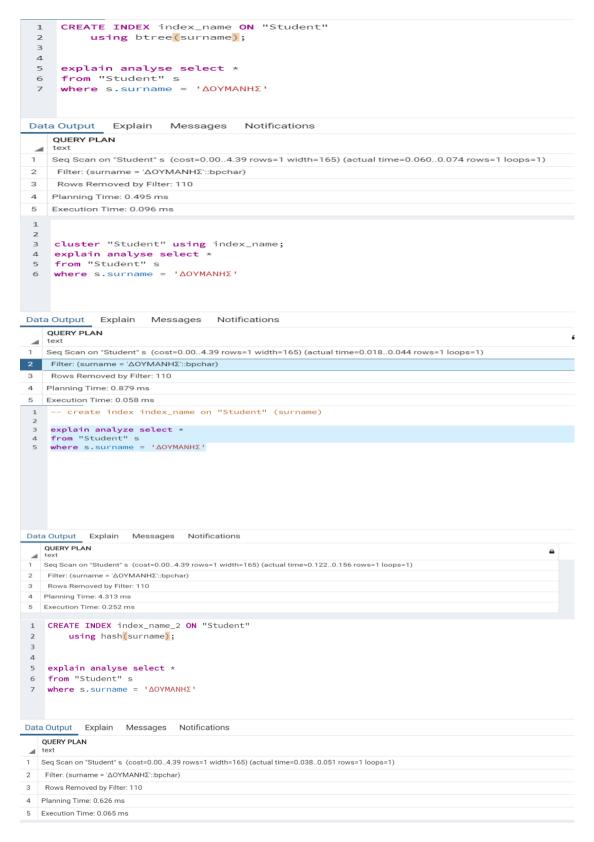
- $6.1 \rightarrow \text{passed } 6.1 \text{ students}$
- $\underline{6.2}$ → weekly_6_2_program

Φυσικός σχεδιασμός:

3.Α.) Στο συγκεκριμένο ερώτημα, ζητήθηκε να μελετηθεί το εξής ερώτημα: Να βρεθούν οι φοιτητές που έχουν ένα ορισμένο πατρώνυμο. Τυχαία επιλέχτηκε το επίθετο ΔΟΥΜΑΝΗΣ. Αρχικά για μικρό όγκο δεδομένων περίπου 90-110 φοιτητές, δοκιμάστηκε να εκτελεστεί η explain analyse εντολή της sql, με σκοπό να βρεθεί ο χρόνος εκτέλεσης και προγραμματισμού. Χωρίς την παρουσία κάποιου index τα αποτελέσματα έχουν ως εξής:

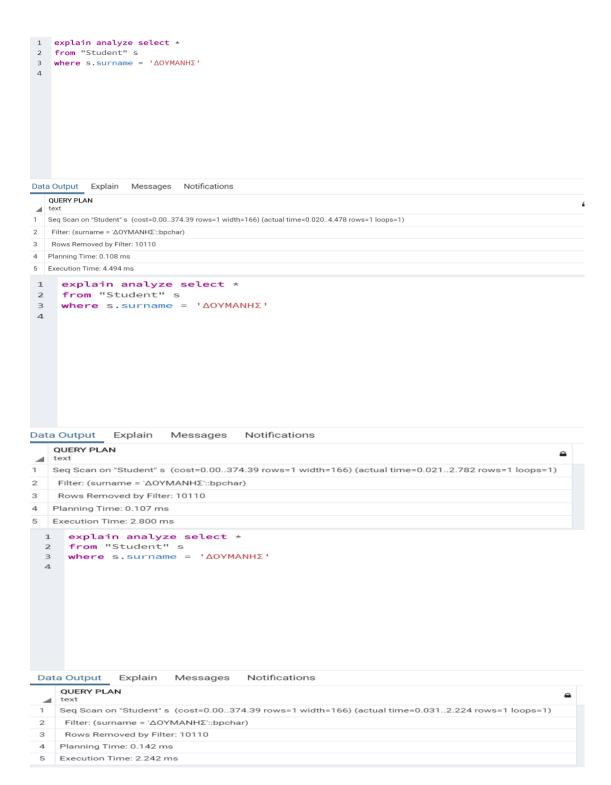
Με την δημιουργία indexes τύπου hash και btree, με ή χωρίς cluster για την 2^{η} περίπτωση, σε μικρό αριθμό δεδομένων, ο "compiler" της sql, αναγνωρίζει ότι η βέλτιστη αναζήτηση γίνεται σειριακά. Καθώς λοιπόν μεγαλύτερο ενδιαφέρον έχουν οι αναζητήσεις για μεγάλο όγκο δεδομένων, παραθέτουμε ενδεικτικά κάποια αποτελέσματα και ο σχολιασμός σε βάθος τον αλγορίθμων θα γίνει παρακάτω.





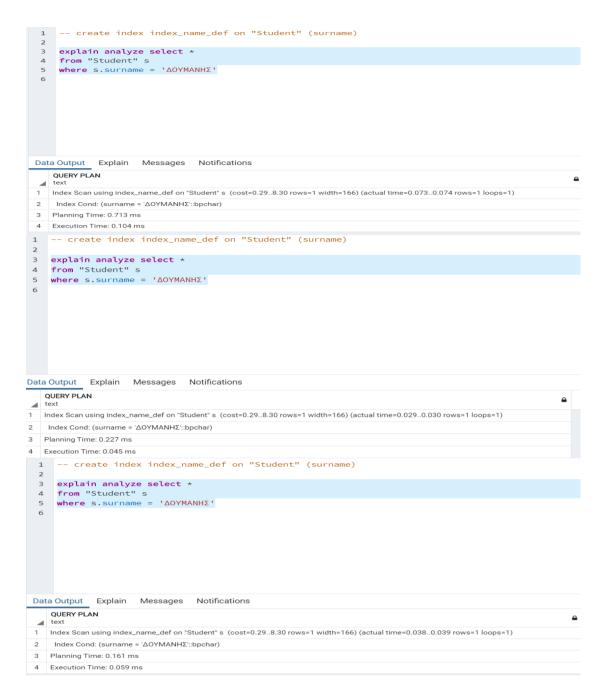
Στη συνέχεια, με χρήση της συνάρτησης insert_mass_data_phase_2(), προσθέτουμε στον πίνακα Student επιπλέον δεδομένα και επαναλαμβάνουμε το πείραμα μας.

Αρχικά λοιπόν, χωρίς την χρήση κάποιου ευρετηρίου, εκτελούμε σειριακή αναζήτηση.



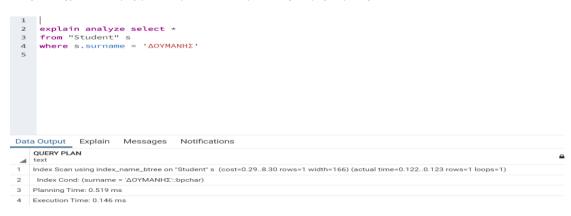
Παρατηρείται ότι το planning time κυμαίνεται στα 0.100 ms, ενώ το execution time χρειάζεται από 2.5 εως και 4.5 ms.

Έπειτα, δοκιμάζουμε να κατασκευάσουμε ένα default index χωρίς την χρήση κάποιου cluster.



Σε αυτή την περίπτωση, αν και το planning time είναι μεγαλύτερο, φαίνεται ότι το execution time είναι σημαντικά μικρότερο.

Στη συνέχεια θα εφαρμόσουμε indexes με τους αλγορίθμους hash και btree.

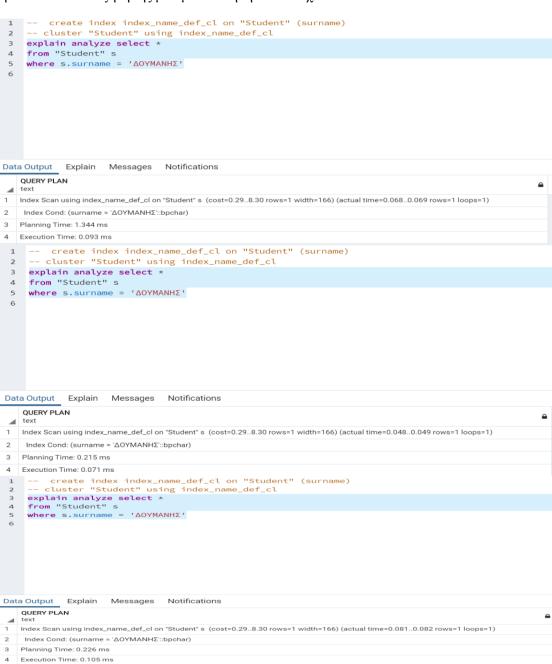


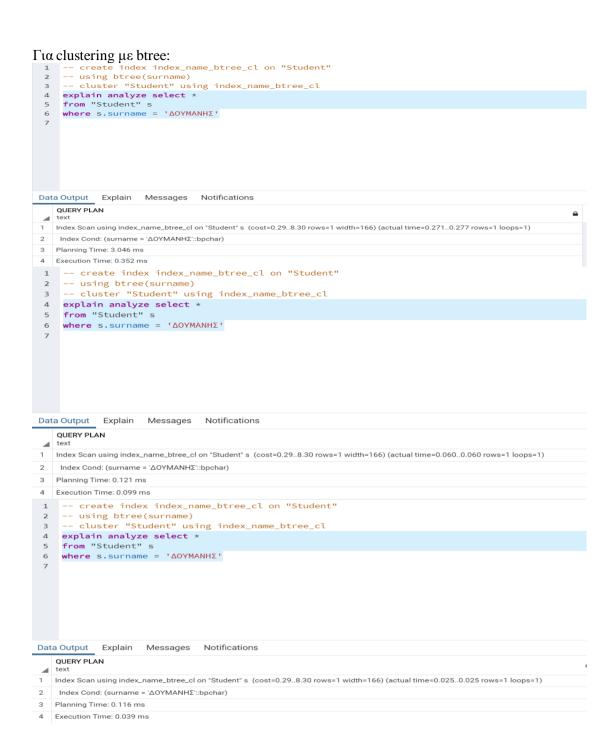
```
explain analyze select *
from "Student" s
where s.surname = 'ΔΟΥΜΑΝΗΣ'
  3
Data Output Explain Messages Notifications
     QUERY PLAN
     Index Scan using index_name_btree on "Student" s (cost=0.29..8.30 rows=1 width=166) (actual time=0.045..0.047 rows=1 loops=1)
      Index Cond: (surname = 'ΔΟΥΜΑΝΗΣ'::bpchar)
 3
     Planning Time: 0.189 ms
     Execution Time: 0.071 ms
     explain analyze select *
from "Student" s
  2
       where s.surname = '\Delta OYMANH\Sigma'
Data Output Explain Messages Notifications
    QUERY PLAN
text
 1 Index Scan using index_name_btree on "Student" s (cost=0.29..8.30 rows=1 width=166) (actual time=0.040..0.041 rows=1 loops=1)
     Index Cond: (surname = 'ΔΟΥΜΑΝΗΣ'::bpchar)
 3 Planning Time: 0.191 ms
 4 Execution Time: 0.066 ms
Btree results.
     -- create index index_name_hash on "Student"
-- using hash(surname)
  1
      explain analyze select *
  4
       from "Student" s
      where s.surname = 'ΔΟΥΜΑΝΗΣ'
  6
Data Output Explain Messages Notifications
    Index Scan using index_name_hash on "Student" s (cost=0.00..8.02 rows=1 width=166) (actual time=0.056..0.056 rows=1 loops=1)
      Index Cond: (surname = 'ΔΟΥΜΑΝΗΣ'::bpchar)
 3
     Planning Time: 2.575 ms
     Execution Time: 0.134 ms
     -- create index index_name_hash on "Student"
-- using hash(surname)
      explain analyze select *
from "Student" s
      where s.surname = 'ΔΟΥΜΑΝΗΣ'
Data Output Explain Messages Notifications
     QUERY PLAN
    Index Scan using index_name_hash on "Student" s (cost=0.00..8.02 rows=1 width=166) (actual time=0.016..0.018 rows=1 loops=1)
     Index Cond: (surname = 'ΔΟΥΜΑΝΗΣ'::bpchar)
    Planning Time: 0.189 ms
 4 Execution Time: 0.090 ms
```



Hash results.

Πριν το σχολιασμό των αποτελεσμάτων, θα εκτελεστεί για το default ευρετήριο και για το btree η διαδικασία αναζήτησης με ομαδοποίηση αντίστοιχα.





Είναι φανερό, πως η χρήση οποιουδήποτε ευρετηρίου, έχει σημαντική βελτίωση στο χρόνο εκτέλεσης της αναζήτησης. Συγκεκριμένα για το πρόβλημα αυτό, καθώς πρόκειται για πρόβλημα αναζήτησης ισότητας, η βέλτιστη μέθοδος είναι η hash search (η οποία μάλιστα δεν μπορεί να εφαρμόσει ομαδοποίηση), btree, θα μπορούσαμε να είχαμε επιλέξει αν το πρόβλημα αφορούσε την εύρεση κάποιου διαστήματος τιμών ή κάποιας ανισότητας. Το σίγουρο είναι ότι για μεγάλο όγκο δεδομένων, η σειριακή αναζήτηση δεν είναι καλή λύση, σε αντίθεση με μικρό αριθμό δεδομένων, όπου η εφαρμογή κάποιου άλλου αλγορίθμου δεν θα είχε κάποια σημαντική βελτιστοποίηση και μάλιστα θα αύξανε το planning time.

Παρακάτω, είναι η βέλτιστη επιλογή του compiler της sql, με την παρουσία όλων τον indexes με ή χωρίς clustering. Η βέλτιστη επιλογή του, είναι αναζήτηση με hashing.

```
1 -- create index index_name_btree on "Student"
2 -- using btree(surname)
3 -- cluster "Student" using index_name_def_cl
4 explain analyze select *
from "Student" s
where s.surname = 'ΔΟΥΜΑΝΗΣ'
7

Data Output Explain Messages Notifications

QUERY PLAN
text
1 Index Scan using index_name_hash on "Student" s (cost=0.00..8.02 rows=1 width=166) (actual time=0.013..0.014 rows=1 loops=1)
2 Index Cond: (surname = 'ΔΟΥΜΑΝΗΣ::bpchar)
3 Planning Time: 0.226 ms
```

4 Execution Time: 0.030 ms