

ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΒΑΣΕΙΣ ΔΕΔΟΜΕΝΩΝ

ΑΝΑΦΟΡΑ ΔΕΥΤΕΡΟΥ ΜΕΡΟΥΣ ΕΡΓΑΣΤΗΡΙΑΚΗΣ ΑΣΚΗΣΗΣ

Αποστολόπουλος Θεόδωρος, Σπυριδάκης Χρήστος

A.M. 2014030170, 2014030022

Εισαγωγή:

Στην αναφορά του δεύτερου μέρους της εργαστηριακής άσκησης μας ζητήθηκε η μελέτη και ο σχολιασμός της απόδοσης της βάσης δεδομένων που κατασκευάσαμε, όσον αφορά ερωτήματα εισαγωγής (1.1) και ανάκτησης (2.1, 2.2, 2.3) δεδομένων προς και από αυτήν. Είναι αναγκαίο να αναφερθεί ότι τα παρακάτω αποτελέσματα και συμπεράσματα έχουν προκύψει ύστερα από τη εισαγωγή τουλάχιστον 90.000 Persons (30.000 Students, 30.000 Professors, 30.000 labstaff) προκειμένου να είναι εμφανέστερες οι διαφορές στην απόδοση ανάλογα με τη μέθοδο ανάκτησης η εισαγωγής δεδομένων που χρησιμοποιούμε.

Ζήτημα 2:

Αναφορικά με το 2.1 της πρώτης φάσης της εργασίας, δεν δημιουργείται κάποιο ευρετήριο καθώς ο optimizer χτίζει αυτόματα ευρετήριο στα πρωτεύοντα κλειδιά, και στην συγκεκριμένη περίπτωση το amka, στο οποίο γίνονται join οι πίνακες Person και Students, είναι Primary Key και στους δύο πίνακες. Αν ωστόσο δεν υπήρχε η παραπάνω λειτουργία της Postgresql, θα δημιουργούσαμε ευρετήριο με την μέθοδο Hash καθώς όπως αναφέρεται και στην θεωρία έχουμε point query και η μέθοδος Hash είναι πιο αποτελεσματική από αυτή του B-Tree. Όπως διαφαίνεται στον παρακάτω πίνακα, ο optimizer για να εκτελέσει το ερώτημα αξιοποιεί και τα δύο ευρετήρια που έχουν δημιουργηθεί και προκύπτουν οι εξής χρόνοι:

- Planning Time: 1.17ms
- Execution Time: 3.5ms

Previous queries	
<pre>explain analyze SELECT "Person".fname, "Person".surname, "Students".am FROM "Person" JOIN "Students" ON "Person".amka = "Students".amka JOIN "Register" ON "Students".amka = "Register".student_amka JOIN "CourseRun" ON "CourseRun".serial_number = "Register".course_run_id JOIN "Semester" ON "CourseRun".main_in_semester = "Semester".semester_id JOIN "Course" ON "CourseRun".implements_course = "Course".course_code WHERE "Register".register_status = 'approved' AND "Semester".semester_status = 'present' AND "Course".course_code = 'MAB 202' ORDER BY "Person".surname, "Person".fname;</pre>	
Output pane	
Data Output Explain Messages History	
QUERY PLAN	
text	
1	Sort (cost=37.76..37.77 rows=1 width=128) (actual time=3.333..3.334 rows=23 loops=1)
2	Sort Key: "Person".surname, "Person".fname
3	Sort Method: quicksort Memory: 31kB
4	-> Nested Loop (cost=1.55..37.75 rows=1 width=128) (actual time=0.451..3.289 rows=23 loops=1)
5	-> Nested Loop (cost=1.41..29.58 rows=1 width=138) (actual time=0.445..3.243 rows=23 loops=1)
6	-> Nested Loop (cost=1.27..21.42 rows=1 width=143) (actual time=0.438..3.214 rows=23 loops=1)
7	Join Filter: ("Register".course_run_id = "CourseRun".serial_number)
8	Rows Removed by Join Filter: 545
9	-> Nested Loop (cost=1.13..13.24 rows=1 width=132) (actual time=0.389..2.663 rows=284 loops=1)
10	-> Nested Loop (cost=0.84..12.89 rows=1 width=143) (actual time=0.383..1.771 rows=284 loops=1)
11	-> Index Scan using register_status_index on "Register" (cost=0.42..4.44 rows=1 width=15) (actual time=0.378..0.771 rows=284 loops=1)
12	Index Cond: (register_status = 'approved')::register_status
13	-> Index Scan using "Person_pkey" on "Person" (cost=0.42..8.48 rows=1 width=128) (actual time=0.003..0.003 rows=1 loops=284)
14	Index Cond: (amka = "Register".student_amka)
15	-> Index Scan using "Students_pkey" on "Students" (cost=0.29..0.35 rows=1 width=22) (actual time=0.003..0.003 rows=1 loops=284)
16	Index Cond: (amka = "Person".amka)
17	-> Index Scan using coursenum_implencon_index on "CourseRun" (cost=0.14..0.16 rows=1 width=10) (actual time=0.002..0.002 rows=2 loops=284)
18	Index Cond: (implements_course = 'MAB 202')::bpchar
19	-> Index Scan using "Semester_pkey" on "Semester" (cost=0.14..0.15 rows=1 width=4) (actual time=0.001..0.001 rows=1 loops=23)
20	Index Cond: (semester_id = "CourseRun".main_in_semester)
21	Filter: (semester_status = 'present')::semester_status
22	-> Index Only Scan using "Course_pkey" on "Course" (cost=0.14..0.16 rows=1 width=10) (actual time=0.001..0.002 rows=1 loops=23)
23	Index Cond: (course_code = 'MAB 202')::bpchar
24	Heap Fetches: 23
25	Planning time: 1.178 ms
26	Execution time: 3.495 ms

Όσον αφορά το ερώτημα 2.2 κατασκευάσαμε τα εξής ευρετήρια:

```
CREATE INDEX register_student_index on "Register" (student_amka);  
  
CREATE INDEX register_courseRun_index on "Register"(course_run_id);  
  
CREATE INDEX courseRun_ruinsSeme_index on "CourseRun"(ruinsin_semester);  
  
CREATE INDEX courseRun_implemCou_index on "CourseRun"(implements_course);  
  
CREATE INDEX register_status_index on "Register"(register_status);
```

Επιλέξαμε τα συγκεκριμένα attributes για να δημιουργήσουμε B-tree indexes καθώς σε αυτά κάνουμε join και βελτιστοποιούν στο μέγιστο την υλοποίηση μας. Έτσι λοιπόν προέκυψαν οι παρακάτω χρόνοι:

- Planning Time: 0,6 ms
- Execution Time: 0.15 ms

```
explain analyze  
SELECT  
  "Person".amka,  
  "Person".email,  
  "Person".fname,  
  "Person".father_name,  
  "Person".surname  
FROM "Person" JOIN "Students" ON "Person".amka = "Students".amka  
WHERE "Students".am = '2012119147';
```

QUERY PLAN	
1	Nested Loop (cost=0.78..36.75 rows=1 width=242) (actual time=0.032..0.033 rows=1 loops=1)
2	-> Index Scan using "Students_am_key" on "Students" (cost=0.29..8.30 rows=1 width=11) (actual time=0.011..0.011 rows=1 loops=1)
3	Index Cond: (am = '2012119147'::bpchar)
4	-> Index Scan using "Person_pkay" on "Person" (cost=0.42..8.44 rows=1 width=242) (actual time=0.011..0.011 rows=1 loops=1)
5	Index Cond: (amka = "Students".amka)
6	Planning time: 0.610 ms
7	Execution time: 0.156 ms

Συγκρίνοντας τα αποτελέσματά μας με αυτά στα οποία δεν χρησιμοποιούμε ευρετήριο παρατηρούμε ότι ο χρόνος εκτέλεσης έχει μειωθεί σημαντικά.

Τέλος, σχετικά με το 2.3 ερώτημα, όπως ακριβώς και στο 2.1, δεν δημιουργήσαμε δικό μας ευρετήριο καθώς το query που υλοποιεί το ερώτημα κάνει Join μεταξύ των Students και Person πάνω στο πρωτεύον κλειδί amka. Έτσι λοιπόν, το ευρετήριο μας παραδίδεται έτοιμο από τον optimizer και η αναζήτηση στοιχείων με την βοήθειά του μας δίνει τα εξής αποτελέσματα:

- Planning Time: 0.34 ms
- Execution Time: 3400 ms

Materialised:

<pre>explain analyse select * from phase2_materialized</pre>	
Output pane	
Data Output	Explain Messages History
QUERY PLAN	
text	
1	Seq Scan on phase2_materialized (cost=10000000000.00..10000000011.50 rows=150 width=512) (actual time=0.004..0.005 rows=14 loops=1)
2	Planning time: 0.023 ms
3	Execution time: 0.015 ms

Ζήτημα 4:

Όπως αναφέρθηκε και προηγουμένως, εισάγουμε 90.000 στοιχεία στον πίνακα Person της βάσης μας, καλώντας την συνάρτηση insert_people την πρώτη φορά έχοντας δημιουργημένα τα ευρετήρια και έπειτα χωρίς.

<pre>explain analyse select * from insert_people(30000,30000,30000,'2013/10/10')</pre>	
Output pane	
Data Output	Explain Messages History
QUERY PLAN	
text	
1	Function Scan on insert_people (cost=0.25..10.25 rows=1000 width=32) (actual time=1832581.896..1832581.896
2	Planning time: 0.042 ms
3	Execution time: 1832581.913 ms

<pre>explain analyse select * from insert_people(30000, 30000, 30000, '2018/2/2');</pre>	
Output pane	
Data Output	Explain Messages History
QUERY PLAN	
text	
1	Function Scan on insert_people (cost=0.25..10.25 rows=1000 width=32) (actual time=1860989.043..1860989.043 rows=0 loops=1)
2	Planning time: 0.059 ms
3	Execution time: 1860989.072 ms

Παρατηρούμε ότι ο χρόνος που απαιτείται για την εισαγωγή 90.000 στοιχείων δεν διαφέρει σημαντικά είτε χρησιμοποιούμε indexes είτε όχι(1830000- 1860000ms). Αυτό συμβαίνει διότι η συνάρτηση που εισάγει δεδομένα δεν έχει κάποια where συνθήκη που να απαιτεί αναζήτηση ή ανάκτηση δεδομένων από τη βάση. Αντιθέτως, κανονικά θα έπρεπε ο χρόνος της εισαγωγής χωρίς index να είναι ελαφρώς μεγαλύτερος καθώς τα ευρετήρια έχουν το μειονέκτημα του να απαιτούν παραπάνω χρόνο για την δημιουργία τους. Στην περίπτωση μας οι χρόνοι βγήκαν παρόμοιοι, με τον δεύτερο να είναι ελαφρώς

μεγαλύτερος, γεγονός όμως που μπορεί να οφείλεται στο ότι τρέξαμε την εισαγωγή μόνο μία φορά καθώς όπως παρατηρήθηκε από παραπάνω ερωτήματα οι χρόνοι κάθε φορά αυξομειώνονται.