Αναφορά Εργαστηρίου

Τσιαούσης Χρήστος 2016030017 , Τερζής Αλέξανδρος 2013030184

25 Μαΐου 2020

Περίληψη

Το εργαστηριακό μέρος του μαθήματος αποτελείται από την εξοικείωση με την POSTGRESQL σε μια βάση δεδομένων ενός πανεπιστημίου. Σκοπός ήταν η εισαγωγή σχέσεων και πινάκων, η δημιουργία συναρτήσεων, όψεων καθώς και triggers.Μετά την εξοικείωση αυτή, καλούμαστε να δημιουργήσουμε μία απλή διεπαφή για την βάση σε γλώσσα JAVA, να δημιουργήσουμε ενημερώσιμες όψεις, αλλά και να μελετήσουμε την απόδοση ορισμένων ερωτήσεων.

Φάση Α

Επεξήγηση συναρτήσεων του μέρους 3

3.1

Η υλοποίηση αποτελείται από τρεις πανομοιότυπες συναρτήσεις. Θα εξηγήσω αυτη των καθηγητών καθώς χρειάστηκε να δημιουτγηθούν δυο υποσυναρτήσεις για την τυχαία παραγωγή εργαστηρίων και κατάταξης. Όμοια υποσυνάρτηση χρησιμοποιείται και στους εργαστηριακούς. Για την τυχαία παραγωγή εργαστηρίου, η υποσυνάρτηση, είναι απλή, κι επιστρέφει πίνακα μεγέθους N που δίνεται ως όρισμα, όπως στην ήδη υλοποιημένη για τυχαία ονόματα συνάρτηση. Για την τυχαία κατάσταση λόγω του ότι το enumeration έχει εύρος 4, χρειάστηκε η συνάρτηση unnest καθώς και ένα full outter join με τον πίνακα τον ονομάτων για να επιστρέφεται πίνακας μεγέθους N αντί για 4.

Έχοντας δημιουργήσει όλες τις απαραίτητες συναρτήσεις, μπορούμε να δημιουργήσουμε έναν πίνακα μεγέθους N διαλέγοντας όνομα, πατρόνυμο, επίθετο, εργαστήριο και κατάταξη από τις αντίστοιχες συναρτήσεις καθώς και ένα ξεχωριστό e-mail χρησιμοποιόντας την concat και ένα ειδικό sequence που δημιουργήθηκε για τον σκοπό αυτό. Όλη αυτή η λειτουργικότητα βρίσκεται στο συντακτικό WITH *INS* AS (SELECT ...) κι έπειτα εισάγουμε στον πίνακα "Professor" διαλέγοντας από την σχέση *INS*.

3.2

Αρχικά υπολογίζει το semester_id σε μια τοπική μεταβλητή της συνάρτησης βάσει των εισόδων. Εάν αυτό υπάρχει, τότε εισάγει έναν τυχαίο βαθμό εξεταστικής για όλα τα πεδία που είναι nullγια τα μαθήματα του εξαμήνου και για τους φοιτητές που είναι σε κατάσταση approved.Έπειτα, για τον βαθμό εργαστηρίου, ελέγχεται αν είναι μεγαλύτερος του 5 και σε αυτή την περίπτωση δεν τον αλλάζει. Αλλιώς εισάγεται ένας τυχαίος βαθμός με τον ίδιο τρόπο που εισάγεται και ο βαθμός της εξεταστικής. Τέλος, για τον τελικό βαθμό ελέγχεται αν το μάθημα είναι εργαστηριακό μέσω τις τιμής του κελιού lab_hoursαπό τον πίνακα "Course"κι αν είναι, υπολογίζεται ο τελικός βαθμός βάσει του ορισμού του απο τον εργαστηριακό και της τελικής ελέγχοντας αν ικανοποιείται το κριτήριο του ελάχιστου εργαστηριακού βαθμού. Άν το μάθημα δεν είναι εργαστηριακό, τότε εισάγεται ο βαθμός της τελικής εξέτασης.

3.3

Μία από τις πιο περίπλοκες συναρτήσεις στην βάση μας. Παρακαλώ ανατρέξτε στα σχόλια του κώδικα καθώς είναι δύσκολο να την αναλύσω συνοπτικά.Αρχικά ελέγχεται αν ο κωδικός μαθήματος αντιστοιχεί σε μάθημα του συγκεκριμένου εξαμήνου. Αν ο έλεγχος εγκριθεί, δημιουργείται αναδρομικά ένας προσωρινός πίνακας με το πρόγραμμα αιθουσών, δηλαδή ποιές μέρες και ώρες είναι κατηλειμμένη κάθε αίθουσα στο εβδομαδιαίο πρόγραμμα. Έπειτα, όμοια για διαλέξεις, φροντιστήρια και εργαστήρια, υπάρχουν δύο εμφωλευμένοι βρόγχοι επανάληψης που διαλέγουν τυχαία μια αίθουσα συγκεκριμένου τύπου που δεν υπάρχει στον πίνακα του προγράμματος για τις ώρες και τις ημέρες που ορίζονται από τα for loops.Έτσι όταν βρεθεί κάποια ελεύθερη αίθουσα, οι βρόγχοι τερματίζουν και εισάγεται η δραστηριότητα στον πίνακα "LearningActivity".

Επεξήγηση συναρτήσεων του μέρους 4

4.1

Εδώ γίνεται ένα UNION όλων των καθηγητών κι εργαστηριακού προσωπικού και αυτός ο πίνακας JOIN με ένα δεύτερο SUBQUERY το οποίο διαλέγει όλα τα άμκα απο τον "Participates" INNER JOIN με το "Room" χρησιμοποιώντας το room_id και την χωρητικότητα μεγαλύτερη των 30.

4.2

Πάρε όλα τα activity types απο το LearningActivity που είναι office_hours και διάλεξε καθηγητές που έχουν κάνει participate σε αυτά. Βεβαιώσου μέσω της exists ότι το course_code ανήκει στο τρέχον εξάμηνο.

4.3

Βρες το εξάμηνο από τις εισόδους κι αν δεν υπάρχει εκτύπωσε σφάλμα. Αλλιώς ανάλογα τον τύπο βαθμού, εκτύπωσε το course_code και τον μέγιστο βαθμό από τον πίνακα Register ελέγχοντας ότι το μάθημα ανήκει στο εξάμηνο που δόθηκε σαν είσοδος μέσω του πίνακα CourseRun. Αν ο τύπος βαθμού δεν είναι ένας από τους τρεις αποδεκτούς, εκτύπωσε σφάλμα.

4.4

Σε μία τοπική μεταβλητή υπολογίζεται το τρέχον εξάμηνο. Επιστρέφεται ο πίνακας των φοιτητών με φυσική ένωση, δηλαδή στο αμκα, με τον πίνακα ενώς δεύτερου subquery το οποίο παίρνει χωρίς διπλότυπα, κάθε αμκα από τους πίνακες Participates & LearningActivity όπου ο τύπος της αίθουσας είναι computer_room και ο κωδικός μαθήματος ανήκει στο τρέχον εξάμηνο, με χρήση της συνάρτησης exists.

4.5

Αρχικά, το subquery διαλέγει όλους τους κωδικούς μαθημάτων από των πίνακα μαθημάτων, με left outter join της φυσικής ένωσης των πινάκων `μάθημα΄ και `διδακτική δραστηριότητα΄, έτσι ώστε να έχουμε όλους τους κωδικούς των υποχρεωτικών μαθημάτων ανεξάρτητα από το αν υπάρχουν στον πίνακα LearningActivity.Έπειτα, διαλέγονται όλοι οι κωδικοί και η ένδειξη NAI/OXI (χρησιμοποιώντας το συντακτικό case) ανάλογα με το αν το μέγιστο end_time είναι μεγαλύτερο του 16.

4.6

Αρχικά, επειδή χρησιμοποιούνται οι συναρτήσεις exists & except . Στην πρώτη ελέγχεται ότι το μάθημα έχει εργαστηριακό μέρος και στην δεύτερη το μάθημα έχει όντως δραστηριότητες σε αίθουσα τύπου lab_room. Έτσι παίρνω όλους τους κωδικούς μαθημάτων και με φυσική ένωση στον πίνακα μάθημα, εκτυπώνονται κωδικός και τίτλος. Αυτή η ένωση γίνεται επειδή στην exceptδιαλέγω από τον "LearningActivity" και δεν μπορώ να περάσω τίτλο μαθήματος.

4.7

Αρχικά υπολογίζεται σε μια τοπική μεταβλητή το τρέχον εξάμηνο. Για να εμφανίζεται όλο το διδακτικό προσωπικό, κάνω left outter join τον "LabStaff" με το subquery και χρησιμοποιώ την συνάρτηση COALESCE για να τυπώνει τα κενά πεδία ως 0. Στο subquery διαλέγεται το ΑΜΚΑ και το άθροισμα του end_time - start_time από τον πίνακα Participates όπου το ΑΜΚΑ είναι μεγαλύτερο του 3000, για να πάρω μόνο το εργαστηριακό προσωπικό και το εξάμηνο είναι το τρέχον και το room_id ανήκει (exists) σε διδακτική δραστηριότητα του τωρινού εξαμήνου.

4.8

Εδώ έχουμε τα περισσότερα select distinct για την αποφυγή των διπλοτύπων. Από τον πίνακα LearningActivity διαλέγουμε τους κωδικούς αιθουσών κάνοντας count(distinct course_code) κι έτσι βρίσκουμε τις αίθουσες με το σύνολο μαθημάτων που εξυπηρετούν. Επιπλέον θέτουμε σαν όρο μέσω τις having το νουμερο μαθημάτων που εξυπηρετούν να είναι ίσο με το μέγιστο από ένα πανομοιότυπο subquery κι έτσι εκτυπώνονται όλλες οι άιθουσες που εξυπηρετούν τα περισσότερα μαθήματα.

4.9

Τοποθετούμε σε έναν τοπικό πίνακα, μέσω αναδρομής, το πρόγραμμα αιθουσών για όλα τα εξάμηνα, αναδρομικά. Έπειτα διαλέγουμε για κάθε αίθουσα την μεγαλύτερη συνεχόμενη ώρα λειτουργείας της βάση του προγράμματος και χρησιμοποιώντας τις συναρτήσεις min(start_time), max(end_time) & HAVING. Έτσι καταλήγουμε με τις μέγιστες ώρες λειτουργίας της κάθε αίθουσας, ελέγχοντας πάντα ότι το τέλος μίας δραστηριότητας, είναι η αρχή κάποιας άλλης μέσω του κωδικού μαθήματος.

4.10

Αν οι είσοδοι δεν βγάζουν νόημα raise exception αλλιώς επέστρεψε όλα τα αμκα από τον πίνακα person που είναι καθηγητές και κάνουν match (NaturalJoin) στο υποερώτημα που επιστρέφει τα αμκα από τους Participates natural join LearningActivity natural join Room όπου η χωρητικότητα της αίθουσας είναι η ζητούμενη.

Επεξήγηση συναρτήσεων του μέρους 5

5.1

Το trigger function *reject__amka_for_participates* αρχικά υπολογίζει σε μια τοπική μεταβλητή το άθροισμα των ωρών που συμμετέχει το καινούριο άμκα σε δραστηριότητες που εκτελούνται σε αίθουσες τύπου lab ή xomputer_room για το συγκεκριμένο μάθημα. Έπειτα ελέγχει αν το ΑΜΚΑ υπάρχει την ίδια μέρα και σε κοινό εύρος ωρών σε άλλη δραστηριότητα μέσω του πίνακα "Participates" και αν ναι, απορρίπτει την εισαγωγή ή ενημέρωση. Έπειτα γίνεται έλεγχος για τις ώρες. Αν το "Course".lab_hours είναι μικρότερο ή ίσο της τοπκής μεταβλητής, τότε δεν επιτρέπεται

στον φοιτητή να ξεπεράσει τις προβλεπόμενες ώρες εργαστηρίου. Σε οποιαδήποτε άλλη περίπτωση επιτρέπεται η εισαγωγή και ενημέρωση.

5.2

Αρχικά δημιουργώ το trigger στον πίνακα LearningActivity και πριν την εισαγωγή ή ενημέρωση στοιχείων καλώ την trigger function *reject_wrong_la* για τον έλεγχο. Αυτή λειτουργέι ως εξής: αν η αρχή της καινούριας δραστηριότητας είναι πριν τις 8 το πρωί ή το τέλος της μετά τις 8 το απόγευμα καθώς και αν η ημέρα τις εβδομάδας δεν ανήκει στο {1,5} τότε δίνει μύνημα σφάλματος. Επίσης αν η καινούρια αίθουσα που πρόκειται να εισαχθεί ή ενημερωθεί υπάρχει είδη στον πίνακα δραστηριοτήτων για την συγκεκριμένη μέρα και υπάρχει επικάλυψη στο διάστημα ωρών της καινούριας, πάλι δεν επιτρέπεται η εισαγωγή. Αλλιώς η εισαγωγή προχωράει κανονικά και η *reject_wrong_la* επιστρέφει το καινούριο στοιχείο.

5.3

Το trigger function ελέγχει αν η κατάσταση του εξαμήνου είναι μελλοντική, κι αν είναι ελέγχει εαν είναι εαρινό ή χειμερινό. Και στις δύο περιπτώσεις διαλέγει από τον πίνακα "CourseRun" τον μέγιστο κωδικό εξαμήνου μέσω τις συνάρτησης HAVING και μέσω του πίνακα "Course" τα μαθήματα χειμερινού ή εαρινού αντίστοιχα και τα εισάγει στον πίνακα "CourseRun" με κωδικό εξαμήνου το NEW.semester_id.

Επεξήγηση συναρτήσεων του μέρους 6

6.1

Το subqueryεπιστρέφει τον κωδικό μαθήματος, τον κωδικό εξαμήνου και τον βαθμό εργαστηρίου από τον πίνακα "Register"και ενώνεται με τον πίνακα "CourseRun"πάνω στους κωδικούς μαθήματος και εξαμήνου και αθροίζεται το πλήθος των κελιών όπου ο βαθμός εργαστηρίου είναι μεγαλύτερος του 8.

6.2

Το subqueryείναι μια αναδρομική σχέση, παρόμοια με αυτήν στο ερώτημα 3.3 με την διαφορά ότι επιστρέφει επιπλέον τον κωδικό μαθήματος. Αυτό ενώνεται με ένα επιπλέον subqueryεπάνω στον πίνακα "CourseRun"το οποίο επιστρέφει το αμκα του κύριου καθηγητή του μαθήματος για το συγκεκριμένο εξάμηνο. Τελικά, καταλήγουμε με την ένωση αυτών, να έχουμε έναν πίνακα με τα κελιά ημέρα, ώρα εκκίνησης, ώρα λήξης, κωδικός αίθουσας, κωδικός μαθήματος και αμκα. Κι έτσι διαλέγουμε τα πρώτα πέντε κελιά κι αντί για το αμκα, προβάλουμε το ονοματεπώνυμο μέσω του πίνακα "Professor".

Αντιστοίχηση ερωτημάτων με ονόματα συναρτήσεων

3.1	CreateStudents__3_1 — CreateProfessors__3_1 — CreateLabStaffs__3_1
3.2	InsertGrades__3_2
3.3	AutoFill__3_3
4.1	BiggerThan30People__4_1
4.2	OfficeHoursOfProfs__4_2
4.3	MaxGrade__4_3
4.4	ParticipatesInPCRoom__4_4
4.5	NoonHours__4_5
4.6	ObligatoryWithoutLabRoom__4_6
4.7	LabAssistantsTotal__4_7
4.8	MostUsedRoom__4_8
4.9	totalTime__4_9
4.10	rangeCapacity__4_10
5.1	reject__amka_for_participates
5.2	reject_wrong_la
5.3	new_semester_5_3
6.1	student_view__6_1
6.2	weekly_view__6_2

Φάση Β

Επεξήγηση του μέρους 1

Δομή του προγράμματος

Το πρόγραμμα χωρίζεται σε δύο αρχεία, το "Run.java"και το "DbApp.java". Στο πρώτο μπορεί κανείς να βρει την κύρια μέθοδο, καθώς και μεθόδους για εκτύπωση των μενού. Στο δεύτερο υλοποιείται η διασύνδεση με το JDBC APIτης POSTGRESQL.

DbApp explanation

- Σύνδεση στην βάση

  Μία απλή συνάρτηση που δέχεται τη διεύθυνση και το όνομα της βάσης, καθώς και το όνομα και τον κωδικό του χρήστη που θέλει να συνδεθεί. Στην περίπτωση που κάτι δεν πάει καλά, εκτυπώνεται μύνημα σφάλματος. Αξίζει να σημειωθεί ότι με το που γίνει η σύνδεση, θέτω να μην γίνονται αυτόματες συνδιαλλαγές.

- Επικύρωση συνδιαλλαγών

  Γίνεται μία απλή κλήση της συνάρτησης rollback().

- Ακύρωση συνδιαλλαγών

  Γίνεται μία απλή κλήση της συνάρτησης commit().

- Εκτύπωση εγγεγραμμένων φοιτητών σε μάθημα εξαμήνου

  Παίρνει τα κατάλληλα ορίσματα, και αρχικά υπολογίζει σε μια τοπική μεταβλητή τον κωδικό του εξαμήνου, κάνοντας χρήση των αντικειμένων Statment & ResultSettου JDBC. Έπειτα κλείνει το ResultSetγια να γίνει εκ νέου ένα executeQuery("...")που θα επιστρέψει το ζητούμενο.

- Μενού βαθμολογίας

  Δημιουργείται ένα ArrayList από αντικείμενα τύπου Savepoint έτσι ώστε να έχουμε την απαιτούμενη λειτουργικότητα. Έπειτα υπολογίζεται το semester_id όπως την παραπάνω συνάρτηση και γίνεται μία δεύτερη εκτέλεση του statment που επιστρέφει τα μαθήματα του εξαμήνου με τις βαθμολογίες του φοιτητή όπως ορίζεται στην εκφώνηση. Μετά υπάρχει ένας βρόγχος που κάθε φορά που διαλέγει ο χρήστης ένα στοιχείο για αλλαγή, πριν από οποιαδήποτε ενέργεια, γίνεται pus hστην ArrayList η κατάσταση των συνδιαλλαγών. Η λίστα είναι υλοποιημένη σαν FIFO stack. Διαλέγοντας ο χρήστης το στοιχείο, του ζητέιται να εισάγει τους βαθμούς οι οποίοι έπειτα μέσω ενόςPreparedStatement γίνεται η ανανέωση στα ζητούμενα κελιά του πίνακα. Παρακαλώ, ανατρέξτε στον κώδικα.

Επεξήγηση του μέρους 2

- lab_schedule_B_2_2

  Όπως ορίζεται από την POSTGRESQL για να ειναί μια όψη ανανεώσιμη, θα πρέπει να αναφέρεται σε εναν μόνο πίνακα και να μην έχει τα στοιχεία GROUP BY, HAVING, LIMIT, OFFSET, DISTINCT, WITH, UNION, INTERSECT, EXCEPT. Επίσης η χρησιμότητα των όψεων, εκτός του να γλυτώνει κανείς χρόνο γράφοντας όλο το ερώτημα, έιναι η μερική προβολή κρίσιμων πινάκων σε unauthorised users. Έτσι, επιλέγουμε τον πίνακα **"Participates"** σε μία όψη που θα εμφανίζει μόνο αυτούς που είναι responcible για μία δραστηριότητα, αλλά όχι όλους τους συμμετέχοντες. Το όνομα της όψης είναι **visible_participants_B_2_1** και η συμπεριφορά της είναι η αναμενόμενη. Κάνοντας δηλαδή αλλαγές στην όψη, οι αλλαγές περνάνε στον πίνακα που αναφέρεται και ελέγχονται τα triggers του πίνακα αυτού.

- lab_schedule_B_2_2

  Δεν υλοποιήθηκε. Παρ΄ όλ΄ αυτά, έγινε κατανοητό ότι οι όψεις, δεν είναι παρά ένα αποθηκευμένο SELECT statment το οποίο μπορούμε να διαχειριστούμε και ως πίνακα, κι έτσι λόγω της σύγχυσης που μπορεί να γίνει πάνω σε εισαγωγές κι ενημερώσεις όταν αυτό εκφράζεται από πολλαπλούς πίνακες, πρέπει χειροκίνητα αυτός που οργανώνει την βάση να προβλέψει το πως θα γίνονται όλα αυτά μέσω triggers.

Επεξήγηση του μέρους 3

Α. Εύρεση φοιτητών που έχουν πατρόνυμο 'ΠΥΡΡΟΣ'

Το ερώτημα που θέσαμε στην βάση είναι *EXPLAIN ANALYSE SELECT * FROM "Student" WHERE father_name = 'ΠΥΡΡΟΣ'*

2.1.1 No index

Στην απλή αυτή περίπτωση, έχοντας 110 εγγεγραμμένους φοιτητές, και μετά από 5 συνεχόμενες εκτελέσεις. Ο μέσος όρος του χρόνου εκτέλεσης του ερωτήματος προκύπτει **0.0644ms**.

Αυξάνοντας τον αριθμό των φοιτητών να είναι 10210 παρατηρούμε μία δραματική αύξηση του χρόνου εκτέλεσης, με μέσο όρο **2.7892ms**.

2.1.2 B-tree Index

Αφού δημιουργήσαμε τον δείκτη στον πίνακα των φοιτητών *using btree(father_name)*παρατηρήσαμε μία απροσδόκητη συμπεριφορά, καθώς ο χρόνος εκτέλεσης φάνηκε να μεγαλώνει και να είναι τελικά **0.0764ms**.Το αποτέλεσμα αυτό μπορεί να ωφείλεται στο γεγονός ότι το balanced treeέχει μια γραμμική συμπεριφορά στην βελτίωση της απόδοσης όσο αυξάνονται οι σειρές του πίνακα, κι έτσι ο χρόνος της πραγματικής εκτέλεσης είναι περίπου ίδιος -με και χωρίς τον δείκτη του δέντρου- και χάνεται χρόνος στον σχεδιασμό εκτέλεσης του ερωτήματος. Επιπλέον αξίζει να σημειωθεί ότι κάθε βάση έχει κάποια εσωτερικά-εικονικά ευρετήρια για την απάντηση των ερωτήσεων και συνήθως αυτά είναι b-trees.

Αυξάνονταστους φοιτητές στους 10210 παρατηρούμε μία εξαιρετική απόδοση του ευρετηρίου **0.1136ms**.Αυτό συμβαίνει λόγω της σχετικά σταθερής απόδοσης του δέντρου όσο αυξάνεται το μέγεθος του πίνακα.

2.1.3 Hash Index

Εδώ έχουμε μια αρκετά καλύτερη απόδοση με τον μέσω όρο να ανέρχετε στα **0.0417ms**. Φαίνεται η αλγοριθμική υπεροχή της μεθόδου αυτής, αν και η διαφορά θα ήταν μεγαλύτερη σε κάποιον πίνακα με περισσότερες σειρές. Παρ΄ όλ΄ αυτά η μέθοδος αυτή δεν χρησιμοποιείται συχνά σε πίνακες με πολλές εγγραφές και ανανεώσεις καθώς με την εισαγωγή ή τροποποίηση κελιών πρέπει να επαναδημιουργείται το ευρετήριο.

Για μέγεθος πίνακα 10210 αυτή η μέθοδος έχει εξ ίσου καλή και σταθερή απόδοση, αλλά λίγο χειρότερη απο αυτή του ευρετηρίου με δεντρο. Ο χρόνος εκτέλεσης ανέρχεται στα **0.1512ms**.Αυτό μας δείχνει ότι το δέντρο είναι πιο αποδοτικό για μεγάλα νούμερα σε σχέση με το hash.

2.1.4 Clustering

Χρησιμοποιήθηκε το συντεκτικό *CLUSTER "Student" using std_btree_index;*και φάνηκε οι χρόνοι εκτέλεσης να μειώνονται συγκριτικά με τις πρώτες δύο μεθόδους αλλά όχι αρκετά για να υπερισχύσουν του hashing.Πιο συγκεκριμένα, είχαμε μέσο όρο εκτέλεσης του ερωτήματος **0.0623ms**.

Για πίνακα μεγέθους 10210 αυτή η μέθοδος έχει την καλύτερη απόδοση απ όλες και ο μ.ο. του χρόνου εκτέλεσης είναι **0.0812ms**.Αυτό συμβαίνει επειδή εκμεταλλευόμαστε το πλήθος των κελιών για αποτελεσματικότερη αναζήτηση.

2.1.5 Αναλυτικοί πίνακες

Πίνακας με πλήθος φοιτητών 110.

	No-INDEX(ms)	B-tree(ms)	Hash(ms)	Cluster(ms)
1	0.098	0.090	0.061	0.053
2	0.053	0.058	0.065	0.067
3	0.049	0.075	0.039	0.093
4	0.055	0.063	0.055	0.054
5	0.067	0.096	0.062	0.087
average	0.0644	0.0764	0.0546	0.0623

Πίνακας με πλήθος φοιτητών 10210.

	No-INDEX(ms)	B-tree(ms)	Hash(ms)	Cluster(ms)
1	3.572	0.125	0.150	0.187
2	2.156	0.093	0.134	0.068
3	2.078	0.117	0.200	0.067
4	2.936	0.109	0.158	0.043
5	3.398	0.124	0.128	0.042
average	2.7892	0.1136	0.1512	0.0810

Άξιο παρατήρησης είναι ότι στις πρώτες εκτελέσεις είναι σημαντικά αργότερη η εκτέλεση. Αυτό μας δείχνει πως στο υπόβραθρο τις βάσης, είναι προγραμματισμένη η μελέτη της χρονικής τοπικότητας των ερωτημάτων. Δηλαδή όταν γίνει μια ερώτηση, είναι πολύ πιθανό να ξανασυμβεί σε κοντινό χρονικό διάστημα. Με αυτό τον τρόπο για παράδειγμα, συναρτήσεις που τρέχουν βρόγχους επανάληψης μπορούν να είναι πολύ αποδοτικότερες.

Β. Εύρεση φοιτητριών που έχουν περάσει το 'ΠΛΗ 302΄ με βαθμό 8

Αρχικά βλέπουμε, χωρίς κανένα index και με τέσσερις εκτελέσεις της EXPLAIN ANALYSE, ότι ο μέσος χρόνος εκτέλεσης είναι **0.298ms** και βλέπουμε ότι στον πίνακα "Register" γίνονται δύο σαρώσεις, ενώ στους πίνακες "Student" & "Name" από μία. Λόγω και του πολύ μεγαλύτερου, αναλογικά, μεγέθους του πρώτου πίνακα συγκριτικά με τους άλλους δύο, θα επιλέξουμε εκεί να δημιουργήσουμε τα ευρετήριά μας. Παρατηρούμε όμως, ότι οι κόμβοι επανάληψης τρέχουνε 3 φορές

για τους πίνακες “Student” & “Name” ενώ για τον “Register” μονάχα μία. Αυτό δημιουργή ένα δίλλημα απόφασης για το τι indexes είναι προτιμότερο να χρησιμοποιηθούν.

Για την απόφαση του που θα τα βάλουμε τελικά, χρησιμοποιήθηκε η εντολή EXPLAIN (ANALYSE,BUFFERS). Εν τέλη, αποκομιζόμενοι και τις γνώσεις μας από το προηγούμενο ερώτημα, καταλήγουμε να έχουμε σταθερά δύο HASH indexes στους πίνακες “Student” & “Name” καθώς είδαμε ότι για το μέγεθος τους αυτή είναι η αποδοτικότερη μέθοδος κι έτσι οι παρακάτω τίτλοι, θα αναφέρονται στην μέθοδο indexing του πίνακα “Register”.

No-Index

Βάζοντας τα δύο indexes στους πίνακες φοιτητών και ονομάτων, βλέπουμε κατευθείαν μια μείωση στον χρόνο εκτέλεσης. Πιο συγκεκριμένα το ερώτημα από 0.298ms πλέον ολοκληρώνεται σε περίπου **0.227ms**.

B-tree

Φαίνεται η διαφορά να είναι πολύ μικρή καθώς ο μλεσος χρόνος εκτέλεσης είναι **0.214ms**.

HASH

Εδω η διαφορά μεγαλώνει, καθώς όπως είδαμε για μικρότερου μεγέθους πίνακες αυτή είναι η αποδοτικότερη μέθοδος και ο μέσος χρόνος είναι **0.187ms**.

Clustering

Παρατηρούμε μια περίεργη συμπεριφορά και μια αλλαγή στην δομή εκτέλεσης, καθώς αλλάζει το δέντο πλάνου του ερωτήματος και γίνεται ένα επιπλέον sort using merge sort(amka) το οποίο καθυστερεί το αποτέλεσμα. Αυτό ίσως να μπορεί να διορθωθεί αν αλλάξουμε το index που χρησιμοποιήθηκε για το Clustering (btree(amka)) παρ΄ όλ΄ αυτά, είναι μια μέρα πριν την παράδωση και δεν προλαβαίνουμε. Ο μέσος χρόνος εκτέλεσης για την μέθοδο αυτή ήταν **0.429ms**.

Σημειώσεις

Λόγω της χρήσης λίνουξ, το configurationτων προγραμμάτων έγινε χειροκίνητα και οι εκδόσεις είναι ελαφρώς διαφορετικές. Πιο συγκεκριμένα, η έκδοση της POSTGREsql είναι η 12.3 καθώς και του PGADMIN4 είναι η 4.21. **Για οποιοδήποτε πρόβλημα συμβατότητας, παρακαλώ επικοινωνήστε άμεσα στο tsiao98@gmail.com.**

Το εργαστήριο του μαθήματος προσέφερε μια αρκετά καλή κατανόηση των λειτουργιών μιας βάσης δεδομένων και θα αποτελέσει ένα πολύ καλό θεμέλιο για οποιαδήποτε ενασχόληση με servers & distributed systems.Ένα μεγάλο ευχαριστώ στους εργαστηριακούς βοηθούς για την άριστη συνεργασία και βοήθεια.