Βάσεις Δεδομένων - ΠΛΗ302

Αναφορά Εργαστηριακής Εργασίας

Ον/μα : Σεφερλή Ηλιοδώρα, Πάνου Χριστίνα

Α.Μ. : 2017030094, 2017030124

Ομάδα : LAB30244782

Ημ/να : 25/5/2020

Συνοπτική περιγραφή Α' φάσης

3. *Διαχείριση δεδομένων*

3.1 Το πρώτο ερώτημα για τη κατασκευή συναρτήσεων, ζητάει την εισαγωγή νέων πλειάδων στους πίνακες Students, Professor, LabStaff. Για την υλοποίησή του, λοιπόν, του δημιουργήθηκαν τρεις διαφορετικές συναρτήσεις στις οποίες γίνονται οι κατάλληλες εισαγωγές για κάθε πίνακα ξεχωριστά. Έχοντας ως όρισμα τον επιθυμητό αριθμό των πλειάδων, με μία επαναληπτική μέθοδο χρησιμοποιήθηκε η συνάρτηση insert και με χρήση βοηθητικών δοσμένων συναρτήσεων και μίας που κατασκευάστηκε πραγματοποιήθηκε η εισαγωγή των ζητουμένων πραγμάτων. Για τους φοιτητές ήταν χρήσιμος ο υπολογισμός του μέγιστου αριθμού amka που υπάρχει μέχρι τη στιγμή νέας εισαγωγής στον πίνακα, για να συνεχίσει από εκεί η καταμέτρησή τους καθώς και ο αριθμός ΑΜ που είναι ξεχωριστός για κάθε φοιτητή αναλόγως ποια χρονιά εισήχθη. Για τους καθηγητές έγινε το ίδιο για τον αριθμό amka και λαμβάνεται ο αριθμός των συνολικών εργαστηρίων που υπάρχουν για τη χρήση του σε συνάρτηση που παράγει τυχαίους αριθμούς, για να τεθεί ως ανώτατο όριο. Ακριβώς παρόμοια διαδικασία ήταν και για το εργαστηριακό προσωπικό. Για όλα αυτά χρησιμοποιήθηκαν οι συναρτήσεις **random_names**, **random_surname** και **adapt_surname**, για να παραχθούν τυχαία ονοματεπώνυμα και με την τρίτη να προσαρμοστεί το επώνυμο με βάση το φύλο. Επίσης, δημιουργήθηκε η συνάρτηση randomMale, για την λήψη μόνο αρσενικών ονομάτων για τα πατρώνυμα.

3.2 Η υλοποίηση του δεύτερου ερωτήματος είχε δύο σκέλη, ένα για τον υπολογισμό του exam_grade για τη περίπτωση τοποθέτησης βαθμών για φοιτητές που δεν έχουν περαστεί ακόμη και ένα για την εισαγωγή βαθμών εργαστηρίων, το οποίο είχε και μεγαλύτερη πολυπλοκότητα λόγω της αναζήτησης σε προηγούμενα έτη. Αυτό που έγινε, αρχικά, ήταν η δημιουργία ενός πίνακα που αποτελείται από τους φοιτητές που δεν έχουν βαθμολογία στο lab_grade, ενώ είναι εργαστηριακό μάθημα και δεν είναι rejected για το συγκεκριμένο επιθυμητό εξάμηνο. Στη συνέχεια, ο πίνακας αυτός ενώθηκε με τον Register, για να ενωθεί με τον τυχόν φοιτητή που έχει πάρει το μάθημα και σε προηγούμενο έτος. Οπότε, αν ο εργαστηριακός βαθμός είναι μικρότερος του 5 ή δεν υπάρχει εισάγεται σε αυτό που αναζητείται ένας τυχαίος βαθμός ενώ σε αντίθετη περίπτωση εισάγεται ο βαθμός του προηγούμενου έτους.

3.3 Σκοπός αυτού του ερωτήματος είναι η αυτοματοποίηση της εισαγωγής δραστηριοτήτων για συγκεκριμένο μάθημα εξαμήνου σύμφωνα με το πρόγραμμα σπουδών. Συγκεκριμένα, η διαδικασία αναζητά τις διαθέσιμες αίθουσες για διαλέξεις, φροντιστήρια και εργαστήρια και δημιουργεί τις κατάλληλες δραστηριότητες μάθησης. Γι 'αυτό δημιουργήθηκαν οι εξής συναρτήσεις : createlearningActivity, createlearningActivities και random_date. Η συνάρτηση **createlearningactivities** δέχεται ως όρισμα τον κωδικό μαθήματος και με βάση αυτόν γίνεται αναζήτηση των ωρών διαλέξεων, φροντιστηρίων και εργαστηρίων που το υποστηρίζουν. Μέσα σε αυτήν, έπειτα από κατάλληλα if statements, δημιουργούνται με την κλήση της createlearningActivity από μία έως τρεις (ή και καμία, αν το μάθημα δεν διδάσκεται στο τρέχον εξάμηνο) δραστηριότητες ανάλογα

με τις ώρες διαλέξεων, φροντιστηρίων και εργαστηρίων που το μάθημα υποστηρίζει. Η **createlearningactivity** παίρνει ως ορίσματα τον κωδικό του μαθήματος, τον τύπο της δραστηριότητας και του δωματίου. Με βάση αυτά αλλά και τις τιμές που επιστρέφει η random_date δημιουργείται μια σχετική με το μάθημα δραστηριότητα. Η **random_date** επιστρέφει μια τυχαία μέρα, ώρα έναρξης και λήξης με βάση τη ζητούμενη διάρκεια του τύπου της δραστηριότητας, για συγκεκριμένο τύπο δωματίου και υπό τους περιορισμούς να μην συμπίπτει σε χρήση αιθουσών με υπάρχουσα δραστηριότητα. Επίσης, κατασκευάστηκαν για την τυχαία επιλογή συγκεκριμένου τύπου αίθουσας και οι : **random_computer_room_choice**, **random_lab_room_choice** και **random_lecture_room_choice**.

4. *Ανάκτηση δεδομένων και υπολογισμοί*

4.1 Δημιουργήθηκε η συνάρτηση **function_41** για την ανάκτηση ονοματεπωνύμου και ΑΜΚΑ καθηγητών και εργαστηριακού προσωπικού, οι οποίοι έχουν διδάξει σε αίθουσες με χωρητικότητα μεγαλύτερης των 30 ατόμων. Για την δημιουργία του ζητούμενου query χρησιμοποιήθηκαν οι τελεστές : UNION για τον συνδυασμό των αποτελεσμάτων 2 κατάλληλων SELECT statements και INNER JOIN για την αναπαράσταση της τομής 2 συνόλων αποτελεσμάτων. Επίσης, χρησιμοποιήθηκαν τα keywords DISTINCT για την επιστροφή διακριτών τιμών(αποφυγή διπλότυπων) και ORDER BY για την εμφάνιση ταξινομημένων αποτελεσμάτων βάσει name, surname και amka.

4.2 Δημιουργήθηκε η συνάρτηση **function_42** για την εμφάνιση πληροφορίας για τους καθηγητές και τις ώρες γραφείου των μαθημάτων που διδάσκουν το τρέχον εξάμηνο. Ως αποτελέσματα εμφανίζονται το ονοματεπώνυμο του καθηγητή, ο τίτλος του μαθήματος και αντίστοιχες μέρες και ώρες των ωρών γραφείου. Είναι μάλιστα ταξινομημένα αλφαβητικά ως προς το ονοματεπώνυμο του καθηγητή με τη χρήση του keyword ORDER BY. Εδώ έγινε χρήση των τελεστών JOIN και NATURAL JOIN για το συνδυασμό αποτελεσμάτων από περισσότερους των 2 πινάκων με κάποια κοινή στήλη.

4.3 Δημιουργήθηκε η συνάρτηση **maxgrade** για την ανάκτηση της μέγιστης βαθμολογίας για κάθε μάθημα ενός συγκεκριμένου εξαμήνου. Σαν είσοδος δίνονται το εξάμηνο(typical year, typical season) και η επιθυμητή κατηγορία βαθμολογίας, δηλαδή γραπτής εξέτασης, βαθμός εργαστηρίου ή τελική βαθμολογία, ενώ τα αποτελέσματα εμφανίζονται με φθίνουσα σειρά βαθμολογίας(ORDER BY grade DESC). Η υλοποίηση αυτού του query έγινε με την χρήση if statements για καθεμιά από τις περιπτώσεις βαθμολογίας. Επίσης, χρησιμοποιήθηκε το GROUP BY statement για την ομαδοποίηση των αποτελεσμάτων με βάση το course_code και η aggregate function max.

4.4 Δημιουργήθηκε η συνάρτηση **function_44** για την ανάκτηση του αριθμού μητρώου και έτους εγγραφής των φοιτητών, οι οποίοι είναι εγγεγραμμένοι στο τρέχον εξάμηνο σε κάποιο μάθημα που περιλαμβάνει δραστηριότητα που εκτελείται σε αίθουσα με τύπο computer_room. Με βάση το δοσμένο σχήμα, το query

υλοποιήθηκε με NATURAL JOIN μεταξύ των πινάκων Student, Participates, Room και με τις κατάλληλες συνθήκες που τοποθετήθηκαν στην πρόταση WHERE.

4.5 Δημιουργήθηκε η συνάρτηση **function_45** για την ανάκτηση όλων των κωδικών όλων των υποχρεωτικών μαθημάτων με την ένδειξη ΝΑΙ ή ΟΧΙ ανάλογα με τον αν περιλαμβάνουν δραστηριότητες οι οποίες εκπονούνται απογευματινές ώρες (στο διάστημα 16:00-20:00). Χρησιμοποιήθηκε και εδώ το keyword DISTINCT για την αποφυγή διπλότυπων αποτελεσμάτων και ένα CASE statement στο SELECT για την επιλογή της κατάλληλης ένδειξης για κάθε μάθημα.

4.6 Δημιουργήθηκε η συνάρτηση **function_46** ως ακολούθως. Αρχικά, να λαμβάνονται τα δεδομένα του να είναι υποχρεωτικό ένα μάθημα και να έχει εργαστηριακό μέρος από τον πίνακα Courses, το να γίνεται το μάθημα σε αίθουσα εργαστηρίου από το Room και το όλα αυτά να απασχολούν το τωρινό εξάμηνο από το Semester. Οπότε, με κατάλληλες συνδέσεις πινάκων μέσω NATURAL JOIN, αναζητήθηκαν όλα τα μαθήματα που έχουν δηλωθεί οι δραστηριότητές τους στο LearningActivity για το ποια είναι υποχρεωτικά, ποια έχουν μη μηδενικές ώρες εργαστηρίου (δηλαδή έχουν εργαστηριακό μέρος), το εξάμηνο να είναι τωρινό και απασχολούν αίθουσα διαφορετική από lab_room. Για να αποκλειστεί εντελώς η περίπτωση του να έχει ένα μάθημα δηλωθεί σε αίθουσα lab_room, το ερώτημα γίνεται με εξαίρεση με ένα παρόμοιο ερώτημα αλλά με το να δείχνει τα μαθήματα με lab_rooms. Έτσι, με αυτό, τα αποτελέσματα που εξάγονται είναι μόνο τα μαθήματα με εργαστηριακό μέρος που δεν γίνονται καθόλου σε αίθουσα εργαστηρίου.

4.7 Δημιουργήθηκε η συνάρτηση **function_47** με σκοπό να εμφανίζεται όλο το εργαστηριακό προσωπικό και αναλόγως, αν στο τωρινό εξάμηνο είναι εγγεγραμμένος κάποιος σε μία δραστηριότητα, δηλαδή βρίσκεται στο LearningActivity, να υπολογίζεται το άθροισμα τον ωρών (το άθροισμα των διαφορών end_time-start_time). Σε αυτούς που δεν είναι εγγεγραμμένοι στο LearningActivity το εξάμηνο αυτό, τότε θα εμφανιστεί 0 φόρτος εργασίας.

4.8 Δημιουργήθηκε η συνάρτηση **function_48**. Πρώτα φτιάξαμε ένα query στο οποίο θα εμφανίζονταν οι αίθουσες και ο αριθμός των διαφορετικών μαθημάτων που φιλοξενούν. Χρησιμοποιώντας το distinct παίρνουμε μόνο μία φορά το κάθε μάθημα ακόμη και να επαναλαμβάνεται. Οπότε, στη συνέχεια, από αυτό το query κάνουμε SELECT όλα τα δεδομένα με τη προϋπόθεση όμως ο συνολικός αριθμός μαθημάτων που απεικονίζονται να είναι μεγαλύτερος από όλους τους αριθμούς που εμφανίζονται στο query (χρησιμοποιώντας εμφωλευμένο ερώτημα και τη χρήση του ALL).

4.9 Δημιουργήθηκε η συνάρτηση **function_49.** Στο query αυτό χρησιμοποιήθηκε αναδρομή, για την δυνατότητα αναζήτησης σε κάθε περίπτωση των συνεχόμενων ωρών που μπορούν να υπάρξουν σε μία μέρα σε διαφορετική αίθουσα για διαφορετικά μαθήματα. Αυτό πραγματοποιείται με την ένωση του end_time του query που παράγεται κάθε φορά με το start_time της αναδρομικής συνάρτησης. Οπότε, πλέον προκύπτει ο πίνακας roomSchedule που έχει όλες τις ώρες που υπάρχουν σε κάθε αίθουσα συμπεριλαμβανομένου και της ένωσης των ωρών που

ήταν συνεχόμενες. Έπειτα, εφαρμόζοντας NATURAL JOIN με έναν πίνακα που έχει τη max διάρκεια ωρών που υπάρχουν σε κάθε μάθημα σε κάθε μέρα, προκύπτει η μέγιστη διάρκεια συνεχόμενης λειτουργίας κάθε αίθουσας ανά ημέρα εβδομάδας.

4.10 Δημιουργήθηκε η συνάρτηση **function_410**, χρήσει της μεθόδου της διαίρεσης. Συγκεκριμένα, με τη μέθοδο που ειπώθηκε και στο εργαστήριο, δημιουργήθηκε ένα query με τρία υποερωτήματα όπου στο εξωτερικό επιλέγονται όλοι οι καθηγητές που υπάρχουν στο Participates, στη συνέχεια αναζητούνται οι αίθουσες στις οποίες η χωρητικότητα βρίσκεται ανάμεσα στα όρια που θέτονται σαν ορίσματα και τέλος, ουσιαστικά, λαμβάνονται από τον πίνακα Participates όλα τα amka που θα αντιστοιχούσαν σε όλα τα room_id που υπήρχαν στο δεύτερο εσωτερικό query. Ένα από τα δεδομένα στα οποία φαίνεται αυτό είναι για minC=120 και maxC=160.

5. *Λειτουργικότητα με κατασκευή εναυσμάτων (triggers)*

5.1 Δημιουργήθηκε η trigger function **participate_trigger1**, έτσι ώστε κατά την εισαγωγή και ενημέρωση δεδομένων συμμετοχής προσώπων σε δραστηριότητες να ελέγχονται τα παρακάτω:

• Να μην υπάρχει συμμετοχή του προσώπου σε άλλη δραστηριότητα την ίδια μέρα και ώρα.

• Αν το πρόσωπο είναι φοιτητής και συμμετέχει σε δραστηριότητα εργαστηρίου θα πρέπει το άθροισμα των εργαστηριακών ωρών να μην υπερβαίνει τις ώρες εργαστηρίου του συγκεκριμένου μαθήματος όπως ορίζονται από τον οδηγό σπουδών.

Ο έλεγχος του πρώτου για την περίπτωση εισαγωγής γίνεται με κατάλληλα if statements και τον τελεστή EXISTS με βάση τα αποτελέσματα ενός query για τον πίνακα Participates. Για την περίπτωση ενημέρωσης δεδομένων γίνονται κατάλληλοι έλεγχοι ισότητας των amka, start_time, end_time, weekday με τις ειδικές μεταβλητές OLD και NEW.

Ο έλεγχος του δεύτερου γίνεται με την βοήθεια της aggregate function **sum** με όρισμα την αριθμητική διαφορά της ώρας λήξης από αυτήν της έναρξης, που προκύπτουν μετά από NATURAL JOIN μεταξύ των πινάκων LearningActivity και Participates(με κατάλληλες συνθήκες στην πρόταση WHERE). Το αποτέλεσμα του sum καταχωρείται στη μεταβλητή total_hours, η οποία ελέγχεται αν υπερβαίνει τις ώρες εργαστηρίου μέσα κατάλληλα if statements.

Μια βοηθητική boolean μεταβλητή χρησιμοποιείται επιπλέον, για τον καθορισμό της επιτυχούς ή όχι εισαγωγής/ανανέωσης δεδομένων.

5.2 Δημιουργήθηκε η trigger function **learningact_trigger1**, έτσι ώστε κατά την εισαγωγή και ενημέρωση δραστηριοτήτων να ελέγχονται τα παρακάτω:

• Οι τιμές στα πεδία weekday, start_time, end_time θα πρέπει να είναι έγκυρες.

• Η αίθουσα όπου λαμβάνει χώρα η δραστηριότητα που εισάγεται ή ενημερώνεται θα πρέπει να είναι ελεύθερη την συγκεκριμένη ημέρα της εβδομάδας και για τις ώρες διεξαγωγής της δραστηριότητας.

Για τη ζητούμενη λειτουργικότητα χρησιμοποιήθηκαν τα κατάλληλα if statements, ο τελεστής EXISTS και ομοίως με το 5.1 μια μεταβλητή boolean.

5.3 Δημιουργήθηκε η trigger function **fututresemester_trigger1**, η οποία θα ενεργοποιείται κάθε φορά όταν γίνεται insert στον πίνακα Semester. Ουσιαστικά, αυτό που θα κάνει θα είναι, για εξάμηνο που θα έχει τον χαρακτηρισμό 'future', δηλαδή θα αποτελεί μελλοντικό εξάμηνο, θα γίνεται insert στον πίνακα CourseRun και στον πίνακα Supports, τα οποία χρειάζονται να έχουν τα μελλοντικά εξάμηνα. Οπότε, για την insert του καθενός πίνακα θα σχηματίζεται ένα query το οποίο θα επιστρέφει τα επιθυμητά δεδομένα στην εισαγωγή, με συνθήκη στο WHERE να τα παίρνει από το πιο πρόσφατο εξάμηνο, δηλαδή για αυτό με το μεγαλύτερο semester_id για το συγκεκριμένο τύπου εξαμήνου, εαρινό ή χειμερινό.

6. *Λειτουργικότητα με χρήση όψεων (views)*

6.1 Δημιουργήθηκε η όψη **student_view** για την εμφάνιση του πλήθους των φοιτητών ανά μάθημα και ανά εξάμηνο που έχουν περάσει το μάθημα το συγκεκριμένο εξάμηνο και έχουν στο εργαστήριο βαθμό μεγαλύτερο του 8. Για τον ζητούμενο πίνακα πραγματοποιήθηκε JOIN μεταξύ των πινάκων Register και Course(με βάση το course_code) και χρησιμοποιήθηκαν η aggregation function **count**, το GROUP BY statement(με βάση serial_number και course_code) και το keyword ORDER BY για ταξινόμηση με βάση το serial_number.

6.2 Δημιουργήθηκε η όψη **program_view** για την εμφάνιση εβδομαδιαίου προγράμματος τρέχοντος εξαμήνου('present'), ανά αίθουσα διαλέξεων. Η όψη εμφανίζει τον κωδικό της αίθουσας, ημέρα εβδομάδας, ώρα έναρξης και λήξης, ονοματεπώνυμο διδάσκοντα και κωδικό μαθήματος. Για τον ζητούμενο πίνακα πραγματοποιήθηκε JOIN μεταξύ των πινάκων Person, Participates και Room(με βάση το amka και έπειτα το room_id) και χρησιμοποιήθηκαν οι κατάλληλες συνθήκες στην πρόταση WHERE και το keyword ORDER BY για την ταξινόμηση με βάση τα room_id και weekday. Ουσιαστικά γίνεται σύνδεση των πινάκων Participates και Room για την αναζήτηση του ποιοι στο πρόγραμμα αυτό διδάσκουν σε αίθουσες με room_type='lecture_room' και με την ένωση των αποτελεσμάτων με το Person βρίσκεται και το ονοματεπώνυμο των διδασκόντων. Για τον αποκλεισμό της εμφάνισης τον μαθητών στο ερώτημα αυτό, μπήκε ως συνθήκη το amka κάθε ατόμου να μην είναι μέρος του πίνακα Student.

Περιγραφή υλοποίησης ζητουμένων Β' φάσης

3. *Μελέτη απόδοσης ερωτήσεων-φυσικός σχεδιασμός*

Α. Πραγματοποιήθηκε μελέτη του ζητούμενου αιτήματος με βάση το πλάνο και τους χρόνους εκτέλεσης χρήσει της EXPLAIN ANALYZE. Για αρχή η εντολή που χρησιμοποιήθηκε, το QUERY PLAN και η κύμανση των χρόνων φαίνονται παρακάτω.

```
explain analyze select * from "Student" where father_name = 'ΝΕΣΤΩΡ'
```

	QUERY PLAN text
1	Seq Scan on "Student" (cost=0.00..4.38 rows=2 width=165) (actual time=0.098..0.167 rows=2 loops=1)
2	Filter: (father_name = 'ΝΕΣΤΩΡ'::bpchar)
3	Rows Removed by Filter: 108
4	Planning Time: 0.261 ms
5	Execution Time: 0.227 ms

Planning time (ms)	Execution time (ms)	Actual time (ms)
0.158	0.088	0.036…0.066
0.261	0.227	0.098…0.167
0.112	0.066	0.026…0.051
0.086	0.04	0.016…0.029

Στη συνέχεια, με σκοπό την εύρεση κάποιου ευρετηρίου για την επιτάχυνση της εκτέλεσης του αιτήματος επιλέχθηκε ο τύπος του B-tree, του Hash, αλλά και η ομαδοποίηση(clustering) και μελετήθηκε εκ νέου το πλάνο εκτέλεσης.

B-tree :

```
CREATE INDEX student_father_idx ON "Student" USING btree(father_name);
```

	QUERY PLAN text
1	Seq Scan on "Student" (cost=0.00..4.38 rows=2 width=165) (actual time=0.015..0.028 rows=2 loops=1)
2	Filter: (father_name = 'ΝΕΣΤΩΡ'::bpchar)
3	Rows Removed by Filter: 108
4	Planning Time: 0.599 ms
5	Execution Time: 0.038 ms

Planning time (ms)	Execution time (ms)	Actual time (ms)
0.599	0.038	0.015…0.028
0.137	0.089	0.037…0.071
0.107	0.043	0.017…0.031
0.15	0.091	0.026…0.076

Clustering :

```
CREATE INDEX student_father_idx ON "Student" USING btree(father_name);
CLUSTER "Student" USING student_father_idx;
```

	QUERY PLAN text
1	Seq Scan on "Student" (cost=0.00..4.38 rows=2 width=165) (actual time=0.023..0.029 rows=2 loops=1)
2	Filter: (father_name = 'ΝΕΣΤΩΡ'::bpchar)
3	Rows Removed by Filter: 108
4	Planning Time: 0.084 ms
5	Execution Time: 0.038 ms

Planning time (ms)	Execution time (ms)	Actual time (ms)
0.084	0.038	0.023…0.029
0.25	0.101	0.056…0.07
0.124	0.039	0.022…0.028
0.14	0.07	0.04…0.053

Hash :

```
CREATE INDEX student_father_idx ON "Student" USING hash(father_name);
```

	QUERY PLAN text
1	Seq Scan on "Student" (cost=0.00..4.38 rows=2 width=165) (actual time=0.028..0.060 rows=2 loops=1)
2	Filter: (father_name = 'ΝΕΣΤΩΡ'::bpchar)
3	Rows Removed by Filter: 108
4	Planning Time: 0.143 ms
5	Execution Time: 0.078 ms

Planning time (ms)	Execution time (ms)	Actual time (ms)
0.143	0.078	0.028…0.06
0.1	0.054	0.019…0.043
0.688	0.098	0.032…0.062
0.411	0.077	0.034…0.061

Από τη θεωρία ισχύει γενικά ότι η αναζήτηση δείκτη(point query) είναι πολύ γρηγορότερη της απλής αναζήτησης πίνακα, γιατί ανόμοια με τον πίνακα, ένας δείκτης συνήθως περιέχει πολύ λίγες στήλες ανά γραμμή και οι γραμμές είναι ταξινομημένες. Συγκρίνοντας τις τιμές του πρώτου πίνακα με τις αντίστοιχες των πινάκων που αφορούν τη χρήση κάποιου τύπου ευρετηρίου, όντως επιβεβαιώνεται η θεωρία. Ακόμη, όσον αφορά στους τύπους ευρετηρίων που δοκιμάστηκαν, είναι γνωστό ότι η προσπέλαση ενός στοιχείου σε ένα B-tree έχει πολυπλοκότητα O(log(n)), ενώ η αντίστοιχη σε ένα Hash table O(1). Στην πράξη κατά μέσο όρο παρατηρείται ελαφρώς καλύτερη απόδοση με Hash απ ’ό,τι με B-tree. Τέλος, με ομαδοποίηση(clustering) παρατηρείται ελαφρώς καλύτερη απόδοση σε σχέση με το B-tree και παρόμοια με του Hash. Οι διαφορές, ωστόσο που υπάρχουν στους αντίστοιχους χρόνους των 3 μεθόδων είναι πολύ μικρές και δεν αρκούν για να κριθεί κάποια ως βέλτιστη. Αυτό συμβαίνει, γιατί αφορούν μικρό αριθμό δεδομένων.

Στη συνέχεια, αφού σβήστηκαν τα τυχόν προηγούμενα ευρετήρια έγινε εισαγωγή στη βάση εκατοντάδων χιλιάδων δεδομένων(φοιτητών) και μελετήθηκαν εκ νέου τα πλάνα εκτέλεσης όπως φαίνονται παρακάτω για τους διαφορετικούς τύπους ευρετηρίων.

B-tree :

	QUERY PLAN text
1	Bitmap Heap Scan on "Student" (cost=4.32..18.04 rows=5 width=166) (actual time=0.036..0.036 rows=5 loops=1)
2	Recheck Cond: (father_name = 'ΝΕΣΤΩΡ'::bpchar)
3	Heap Blocks: exact=1
4	-> Bitmap Index Scan on stud_fatherb_idx (cost=0.00..4.31 rows=5 width=0) (actual time=0.029..0.029 rows=5 loops=1)
5	Index Cond: (father_name = 'ΝΕΣΤΩΡ'::bpchar)
6	Planning Time: 0.146 ms
7	Execution Time: 0.064 ms

Planning time (ms)	Execution time (ms)	Actual time (ms)
0.146	0.064	0.036…0.036
0.247	0.104	0.056…0.058
0.183	0.086	0.05…0.051
0.179	0.079	0.036…0.037

Clustering :

	QUERY PLAN text
1	Bitmap Heap Scan on "Student" (cost=4.32..18.04 rows=5 width=166) (actual time=0.029..0.029 rows=5 loops=1)
2	Recheck Cond: (father_name = 'ΝΕΣΤΩΡ'::bpchar)
3	Heap Blocks: exact=1
4	-> Bitmap Index Scan on stud_fatherb_idx (cost=0.00..4.31 rows=5 width=0) (actual time=0.024..0.024 rows=5 loops=1)
5	Index Cond: (father_name = 'ΝΕΣΤΩΡ'::bpchar)
6	Planning Time: 0.139 ms
7	Execution Time: 0.058 ms

Planning time (ms)	Execution time (ms)	Actual time (ms)
0.139	0.058	0.029…0.029
0.199	0.078	0.045…0.046
0.103	0.043	0.023…0.024
0.169	0.085	0.049…0.05

Hash :

	QUERY PLAN text
1	Bitmap Heap Scan on "Student" (cost=4.04..17.76 rows=5 width=166) (actual time=0.013..0.015 rows=5 loops=1)
2	Recheck Cond: (father_name = 'ΝΕΣΤΩΡ'::bpchar)
3	Heap Blocks: exact=1
4	-> Bitmap Index Scan on stud_father_idx (cost=0.00..4.04 rows=5 width=0) (actual time=0.006..0.006 rows=5 loops=1)
5	Index Cond: (father_name = 'ΝΕΣΤΩΡ'::bpchar)
6	Planning Time: 0.174 ms
7	Execution Time: 0.041 ms

Planning time (ms)	Execution time (ms)	Actual time (ms)
0.174	0.041	0.013…0.015
0.094	0.029	0.01…0.011
0.116	0.036	0.012…0.013
0.275	0.091	0.033…0.034

Κατά μέσο όρο παρατηρείται καλύτερη απόδοση με Hash απ 'ό,τι με Clustering και ελαφρώς καλύτερη με Clustering απ'ό,τι με B-tree. Χρησιμοποιώντας και τα δύο ευρετήρια, B-tree και Hash, δηλαδή εισάγοντας στο table Student και index hash και index btree με την ομαδοποίηση cluster, το query plan επιλέγει να χρησιμοποιήσει ως βέλτιστη μέθοδο το Hash. Ταυτόχρονα, αυτό γίνεται αντιληπτό από το ότι οι χρόνοι με hash index κυμαίνονται σε μικρότερα διαστήματα σε σχέση με τα άλλα indexes.

B. Αρχικά, για το πρώτο στάδιο του ερωτήματος αυτού παρατηρήθηκε η λειτουργία του query που ζητείται. Τα αποτελέσματα που προέκυψαν φαίνονται παρακάτω:

	QUERY PLAN text
1	Nested Loop (cost=5.74..244.79 rows=1 width=201) (actual time=0.095..0.165 rows=2 loops=1)
2	-> Nested Loop (cost=5.46..244.14 rows=1 width=199) (actual time=0.082..0.140 rows=2 loops=1)
3	-> Bitmap Heap Scan on "Register" r (cost=5.32..235.90 rows=1 width=37) (actual time=0.076..0.131 rows=2 loops=1)
4	Recheck Cond: (course_code = 'HPY 204'::bpchar)
5	Filter: ((register_status = 'pass'::register_status_type) AND (final_grade = '7'::numeric))
6	Rows Removed by Filter: 104
7	Heap Blocks: exact=48
8	-> Bitmap Index Scan on "Register_pkey" (cost=0.00..5.32 rows=120 width=0) (actual time=0.028..0.028 rows=106 loops=1)
9	Index Cond: (course_code = 'HPY 204'::bpchar)
10	-> Index Scan using "Student_pkey" on "Student" s (cost=0.14..8.16 rows=1 width=166) (actual time=0.003..0.003 rows=1 loops=2)
11	Index Cond: (amka = r.amka)
12	-> Index Scan using "Names_pkey" on "Name" n (cost=0.28..0.66 rows=1 width=40) (actual time=0.012..0.012 rows=1 loops=2)
13	Index Cond: (name = s.name)
14	Filter: (sex = 'F'::bpchar)
15	Planning Time: 0.665 ms
16	Execution Time: 0.214 ms

Είναι εμφανές ότι αυτό που κοστίζει περισσότερο χρόνο είναι το nested loop και το bitmap heap scan on "Register", καθώς ο πραγματικός χρόνος εκτέλεσης τους φαίνεται να έχει μεγάλη διάρκεια σε αντίθεση στη συνέχεια με το bitmap index scan και τα index scan στα κλειδιά των πινάκων που χρησιμοποιούνται στο query, που έχουν διαφορά πραγματικού χρόνου σχεδόν μηδενική. Οπότε, θα επικεντρωθούμε στο να μειώσουμε αυτούς τους χρόνους. Επίσης, επειδή καταλαβαίνουμε ότι είναι point query η αναζήτηση που γίνεται, αυτό σημαίνει ότι η χρήση του hash θα είναι πιο χρήσιμη, καθώς το btree χρησιμεύει περισσότερο στα range queries όπως γνωρίζουμε από τις δομές δεδομένων. Αρχικά, λοιπόν, δημιουργούνται ευρετήρια και αναζητείται σε ποια μεταβλητή θα μπορούσε να χρησιμεύσουν για να μειωθεί ο χρόνος εκτέλεσης. Τρέχοντας πάλι το explain analyze του παραπάνω query προκύπτουν τα παρακάτω:

	QUERY PLAN text
1	Nested Loop (cost=5.74..244.79 rows=1 width=201) (actual time=0.098..0.153 rows=2 loops=1)
2	-> Nested Loop (cost=5.46..244.14 rows=1 width=199) (actual time=0.085..0.128 rows=2 loops=1)
3	-> Bitmap Heap Scan on "Register" r (cost=5.32..235.90 rows=1 width=37) (actual time=0.079..0.119 rows=2 loops=1)
4	Recheck Cond: (course_code = 'HPY 204'::bpchar)
5	Filter: ((register_status = 'pass'::register_status_type) AND (final_grade = '7'::numeric))
6	Rows Removed by Filter: 104
7	Heap Blocks: exact=48
8	-> Bitmap Index Scan on "Register_pkey" (cost=0.00..5.32 rows=120 width=0) (actual time=0.026..0.027 rows=106 loops=1)
9	Index Cond: (course_code = 'HPY 204'::bpchar)
10	-> Index Scan using "Student_pkey" on "Student" s (cost=0.14..8.16 rows=1 width=166) (actual time=0.003..0.003 rows=1 loops=2)
11	Index Cond: (amka = r.amka)
12	-> Index Scan using "Names_pkey" on "Name" n (cost=0.28..0.66 rows=1 width=40) (actual time=0.011..0.011 rows=1 loops=2)
13	Index Cond: (name = s.name)
14	Filter: (sex = 'F'::bpchar)
15	Planning Time: 0.658 ms
16	Execution Time: 0.203 ms

Δεν εμφανίζεται να χρησιμοποιείται το ευρετήριο που μόλις εισήχθη, κάτι που σημαίνει ότι δεν βελτιώνει τη λειτουργία με κάποιον τρόπο. Αυτό φαίνεται και από το ότι όσο το τρέξαμε και για άλλες φορές ο χρόνος κυμαινόταν στα ίδια επίπεδα με το προηγούμενο χωρίς ευρετήριο.

	QUERY PLAN text
1	Nested Loop (cost=5.74..244.79 rows=1 width=201) (actual time=0.200..0.304 rows=2 loops=1)
2	-> Nested Loop (cost=5.46..244.14 rows=1 width=199) (actual time=0.171..0.255 rows=2 loops=1)
3	-> Bitmap Heap Scan on "Register" r (cost=5.32..235.90 rows=1 width=37) (actual time=0.158..0.233 rows=2 loops=1)
4	Recheck Cond: (course_code = 'HPY 204'::bpchar)
5	Filter: ((register_status = 'pass'::register_status_type) AND (final_grade = '7'::numeric))
6	Rows Removed by Filter: 104
7	Heap Blocks: exact=48
8	-> Bitmap Index Scan on "Register_pkey" (cost=0.00..5.32 rows=120 width=0) (actual time=0.055..0.055 rows=106 loops=1)
9	Index Cond: (course_code = 'HPY 204'::bpchar)
10	-> Index Scan using "Student_pkey" on "Student" s (cost=0.14..8.16 rows=1 width=166) (actual time=0.009..0.009 rows=1 loops=2)
11	Index Cond: (amka = r.amka)
12	-> Index Scan using "Names_pkey" on "Name" n (cost=0.28..0.66 rows=1 width=40) (actual time=0.023..0.024 rows=1 loops=2)
13	Index Cond: (name = s.name)
14	Filter: (sex = 'F'::bpchar)
15	Planning Time: 1.554 ms
16	Execution Time: 0.409 ms

Στη συνέχεια, δημιουργήθηκε ευρετήριο βάσει course_code και τα αποτελέσματα ήταν τα παρακάτω.

```
create index courseindex on "Register" using hash(course_code)
```

	QUERY PLAN text
1	Nested Loop (cost=5.32..244.38 rows=1 width=201) (actual time=0.114..0.180 rows=2 loops=1)
2	-> Nested Loop (cost=5.05..243.72 rows=1 width=199) (actual time=0.097..0.150 rows=2 loops=1)
3	-> Bitmap Heap Scan on "Register" r (cost=4.90..235.48 rows=1 width=37) (actual time=0.089..0.139 rows=2 loops=1)
4	Recheck Cond: (course_code = 'HPY 204'::bpchar)
5	Filter: ((register_status = 'pass'::register_status_type) AND (final_grade = '7'::numeric))
6	Rows Removed by Filter: 104
7	Heap Blocks: exact=48
8	-> Bitmap Index Scan on courseindex (cost=0.00..4.90 rows=120 width=0) (actual time=0.011..0.011 rows=106 loops=1)
9	Index Cond: (course_code = 'HPY 204'::bpchar)
10	-> Index Scan using "Student_pkey" on "Student" s (cost=0.14..8.16 rows=1 width=166) (actual time=0.003..0.003 rows=1 loops=2)
11	Index Cond: (amka = r.amka)
12	-> Index Scan using "Names_pkey" on "Name" n (cost=0.28..0.66 rows=1 width=40) (actual time=0.013..0.013 rows=1 loops=2)
13	Index Cond: (name = s.name)
14	Filter: (sex = 'F'::bpchar)
15	Planning Time: 0.763 ms
16	Execution Time: 0.241 ms

Παρατηρείται ότι σε αυτή τη περίπτωση γίνεται χρήση του ευρετηρίου που χρησιμοποιήθηκε, αλλά ταυτόχρονα οι χρόνοι που κυμαίνονται στο execution time παραμένουν στο ίδιο μέσο όρο με τα προηγούμενα. Δηλαδή, μετά από αρκετές εκτελέσεις δεν έπεφτε συχνά από το 0.2ms και ήταν πιθανό να φτάσει τα 0.3ms.

Οπότε, δοκιμάστηκε και η εκτέλεση explain analyze χρησιμοποιώντας και την τρίτη μεταβλητή που χρησιμοποιείται στο where clause, τον τελικό βαθμό.

```
create index gradeindex on "Register" using hash(final_grade)
```

	QUERY PLAN text
1	Nested Loop (cost=21.01..33.50 rows=1 width=201) (actual time=0.081..0.102 rows=2 loops=1)
2	-> Nested Loop (cost=20.73..32.84 rows=1 width=199) (actual time=0.066..0.073 rows=2 loops=1)
3	-> Bitmap Heap Scan on "Register" r (cost=20.59..24.61 rows=1 width=37) (actual time=0.060..0.063 rows=2 loops=1)
4	Recheck Cond: ((course_code = 'HPY 204'::bpchar) AND (final_grade = '7'::numeric))
5	Filter: (register_status = 'pass'::register_status_type)
6	Heap Blocks: exact=2
7	-> BitmapAnd (cost=20.59..20.59 rows=1 width=0) (actual time=0.054..0.054 rows=0 loops=1)
8	-> Bitmap Index Scan on "Register_pkey" (cost=0.00..5.32 rows=120 width=0) (actual time=0.031..0.031 rows=106 loops=1)
9	Index Cond: (course_code = 'HPY 204'::bpchar)
10	-> Bitmap Index Scan on gradeindex (cost=0.00..15.02 rows=403 width=0) (actual time=0.020..0.020 rows=409 loops=1)
11	Index Cond: (final_grade = '7'::numeric)
12	-> Index Scan using "Student_pkey" on "Student" s (cost=0.14..8.16 rows=1 width=166) (actual time=0.003..0.003 rows=1 loops=2)
13	Index Cond: (amka = r.amka)
14	-> Index Scan using "Names_pkey" on "Name" n (cost=0.28..0.66 rows=1 width=40) (actual time=0.013..0.013 rows=1 loops=2)
15	Index Cond: (name = s.name)
16	Filter: (sex = 'F'::bpchar)
17	Planning Time: 0.685 ms
18	Execution Time: 0.166 ms

Απευθείας παρατηρείται ότι υπάρχει βελτίωση στις πιο χρονοβόρες διαδικασίες του query που αναφέραμε αρχικά, δηλαδή το δεύτερο nested loop join και το bitmap heap scan. Δηλαδή, ενώ ο χρόνος εκτέλεσης σε ένα από τα προηγούμενα ήταν 0.098..0.165 και τα υπόλοιπα είχαν παρόμοιο χρόνο, με τη χρήση του ευρετηρίου στο final_grade πέφτει στα 0.066...0.073 για τη συγκεκριμένη εκτέλεση του query. Επαναλαμβάνοντας την εκτέλεση παρατηρείται ότι παραμένει χαμηλός ο χρόνος αυτός, καθώς και ότι το execution time μένει σε χαμηλότερα όρια από τα προηγούμενα.

	QUERY PLAN text
1	Nested Loop (cost=21.01..33.50 rows=1 width=201) (actual time=0.075..0.093 rows=2 loops=1)
2	-> Nested Loop (cost=20.73..32.84 rows=1 width=199) (actual time=0.062..0.067 rows=2 loops=1)
3	-> Bitmap Heap Scan on "Register" r (cost=20.59..24.61 rows=1 width=37) (actual time=0.057..0.059 rows=2 loops=1)
4	Recheck Cond: ((course_code = 'HPY 204'::bpchar) AND (final_grade = '7'::numeric))
5	Filter: (register_status = 'pass'::register_status_type)
6	Heap Blocks: exact=2
7	-> BitmapAnd (cost=20.59..20.59 rows=1 width=0) (actual time=0.050..0.050 rows=0 loops=1)
8	-> Bitmap Index Scan on "Register_pkey" (cost=0.00..5.32 rows=120 width=0) (actual time=0.027..0.027 rows=106 loops=1)
9	Index Cond: (course_code = 'HPY 204'::bpchar)
10	-> Bitmap Index Scan on gradeindex (cost=0.00..15.02 rows=403 width=0) (actual time=0.019..0.019 rows=409 loops=1)
11	Index Cond: (final_grade = '7'::numeric)
12	-> Index Scan using "Student_pkey" on "Student" s (cost=0.14..8.16 rows=1 width=166) (actual time=0.002..0.002 rows=1 loops=2)
13	Index Cond: (amka = r.amka)
14	-> Index Scan using "Names_pkey" on "Name" n (cost=0.28..0.66 rows=1 width=40) (actual time=0.012..0.012 rows=1 loops=2)
15	Index Cond: (name = s.name)
16	Filter: (sex = 'F'::bpchar)
17	Planning Time: 0.644 ms
18	Execution Time: 0.142 ms