**Πανεπιστήμιο Πειραιώς**

**Τμήμα Πληροφορικής**



**Εργασία στο μάθημα «Συστήματα Διαχείρισης Βάσεων Δεδομένων»**

***Ατομική Εργασία***

**Όνομα:** Σπυρίδων

**Επώνυμο:** Σολανάκης

**Αριθμός Μητρώου (ΑΜ):** Π18141

**Εξάμηνο:** 5ο

**Ακαδημαϊκό Έτος**: 2022-23

**Ημερομηνία Παράδοσης:** 17/2/2023

**Περιεχόμενα**

Εισαγωγή …………………………………………………σελ. 3

Θέμα 1ο (a) ..…………………………………………….σελ. 4

Θέμα 1ο (b) ..…………………………………………….σελ. 6

Θέμα 2ο (Unique) ..…………………………………….σελ. 9

Θεμα 2ο (a) ...…………………………………………...σελ. 10

Θέμα 2ο (b) ...…………………………………………...σελ. 12

Βιβλιογραφία .…………………………………………..σελ. 17

1. **Εισαγωγή**

Στο πλαίσιο του μαθήματος «Συστήματα Διαχείρισης Βάσεων Δεδομένων» του 5ου εξαμήνου, ζητήθηκε η εκπόνηση μιας εργασίας με στόχο την προσθήκη λειτουργιών στο project «miniDB» του αποθετηρίου στο Github που βρίσκεται στο link: https://github.com/DataStories-UniPi/miniDB. Η εργασία αυτή έχει ως σκοπό την εξοικείωση του φοιτητή με τις διαδικασίες ανάλυσης, σχεδιασμού, υλοποίησης λειτουργιών σε ένα ανοικτό λογισμικό, καθώς και με τον τρόπο χρήσης του Git.

1. **Θέμα 1ο**

*«Enrich WHERE statement by supporting (a) NOT and BETWEEN operators (5/50) and (b) AND and OR operators (10/50)»*

**(a)**

Υποστήριξη τελεστών NOT και BETWEEN.

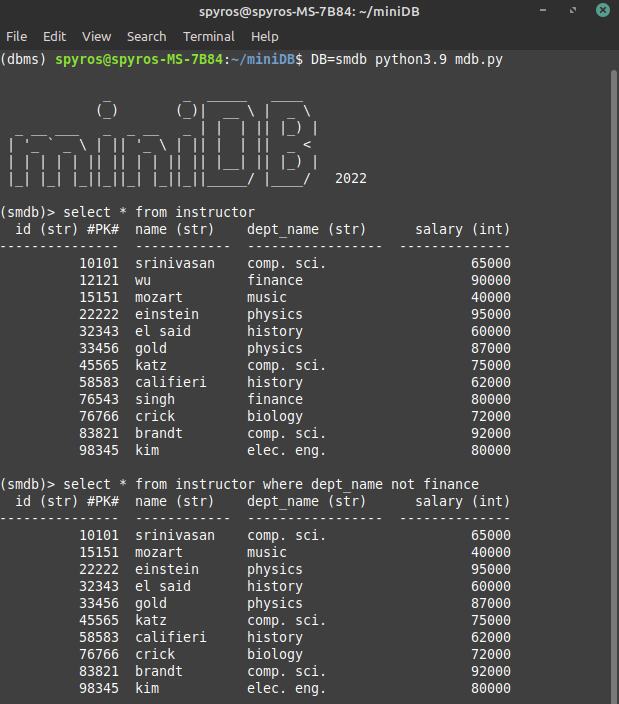
**NOT**

Για τον τελεστή NOT έχουν γίνει προσθήκες στις συναρτήσεις get\_op() και split\_condition() του αρχείου misc.py. Στο λεξικό ops έγινε προσθήκη του key not με την τιμή operator.ne του ενσωματωμένου αρθρώματος (module) operator της Python για τους τελεστές.

*Παράδειγμα χρήσης:*

Select \* from instructor where dept\_name not finance;

**Εικόνα 1:** Χρήση του τελεστή NOT



**BETWEEN**

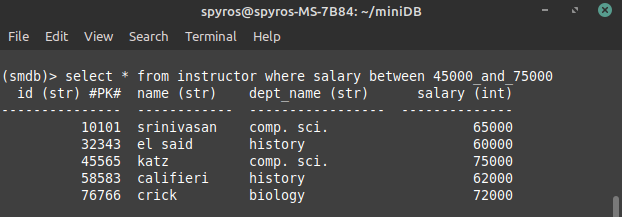
Για τον τελεστή BETWEEN, όπως και στον τελεστή NOT, προστέθηκαν στις συναρτήσεις get\_op() και split\_condition() το κλειδί-string between και ως value το keyword between που θα καλεί τη συνάρτηση between() που δημιουργήθηκε στο αρχείο misc.py.

H συνάρτηση between() δέχεται δύο ορίσματα. Το πρώτο (a) είναι ένας αριθμός ή ένα string που θα ελεγχθεί αν είναι ανάμεσα σε δύο τιμές. Το δεύτερο όρισμα (b) είναι ένα string που περιλαμβάνει τη συνθήκη που ακολουθεί τη δήλωση WHERE σε SQL. Η συνάρτηση between() χωρίζει το b βάσει του keyword “\_and\_” το οποίο θεωρούμε ότι βρίσκεται ανάμεσα στα δύο άκρα της συνθήκης (έστω b[0] και b[1]). Μέσα σε μια δομή try-except επιστρέφεται true αν το a είναι ανάμεσα στα float b[0] και b[1], αλλιώς false. Αν ο κώδικας εγείρει μια εξαίρεση (exception), τότε η μετατροπή σε float απέτυχε, άρα τα b[0] και b[1] είναι strings και θα επιστραφεί true αν το string a είναι ανάμεσα στα strings b[0] και b[1].

Επίσης, έγιναν προσθήκες στον κώδικα της μεθόδου \_parse\_ condition της κλάσης Table στο αρχείο table.py. Πιο συγκεκριμένα προστέθηκε έλεγχος αν ο τελεστής που επιστρέφει η συνάρτηση split\_condition του misc.py είναι ο τελεστής between, οπότε η συνάρτηση \_parse\_ condition θα επιστρέφει τα left, op, right (δηλαδή τη στήλη, τον τελεστή και την τιμή).

*Παράδειγμα χρήσης:*

Select \* from instructor where salary between 45000\_and\_75000;

**Εικόνα 2:** Χρήση του τελεστή BETWEEN

**(b)**

Υποστήριξη τελεστών AND και OR.

**AND**

Στο αρχείο mdb.py, δημιουργήθηκε η συνάρτηση check\_operator() που δέχεται ως όρισμα ένα string με το query που εκτέλεσε ο χρήστης σε SQL. Αφού διασπάσει το string και δημιουργήσει μια λίστα λέξεων, ψάχνει στη λίστα για να βρει τη λέξη “and” οπότε επιστρέφει το string “and”. Διαφορετικά, αν βρει το string “or” το επιστρέφει, αλλιώς επιστρέφει None.

Επιπλέον στο mdb.py, δημιουργήθηκε η συνάρτηση intersect() που δέχεται ως όρισμα ένα string με το query που εκτέλεσε ο χρήστης σε SQL και το διασπά σε απλούστερα queries με απλούστερες συνθήκες στη δήλωση WHERE. Δηλαδή, προσομοιάζει τη λειτουργία της τομής (Intersect) της SQL. Χρησιμοποιώντας το πακέτο dropwhile της βιβλιοθήκης itertools της Python, γίνεται διαχωρισμός της συνθήκης που ακολουθεί τη δήλωση WHERE στις απλούστερες συνθήκες, έστω συνθήκη1 και συνθήκη2. Το σημείο διαχωρισμού των συνθηκών 1 και 2 είναι το string του τελεστή “and”. Καθεμία από αυτές ενώνεται με το πρώτο μέρος του αρχικού SQL query μέχρι τη δήλωση του WHERE και δημιουργούνται νέα απλούστερα SQL queries, τα οποία επιστρέφει η συνάρτηση intersect().

Οι παραπάνω συναρτήσεις που αναλύθηκαν καλούνται μέσα στη συνάρτηση interpret() του αρχείου mdb.py. Πρώτα, καλώντας την check\_operator() ελέγχεται αν στο query υπάρχει η τελεστής AND και αν υπάρχει, καλείται η intersect() για να γίνει ο διαχωρισμός του σε απλούστερα queries. Για καθένα από τα απλούστερα queries, δημιουργείται το query plan του, όπως γίνεται και με τα υπόλοιπα queries. Η interpret() επιστρέφει μια λίστα με τα query plans και τον τελεστή που βρέθηκε στο αρχικό string του query.

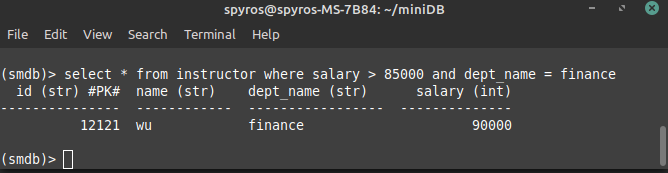
Στο κύριο πρόγραμμα, μετά την κλήση της interpret(), γίνεται έλεγχος για το αν έχει επιστραφεί ο τελεστής AND , οπότε για κάθε query plan που επέστρεψε, γίνεται κλήση της execute\_dic(). Με βάση τα αποτελέσματά της, γίνεται κλήση της μεθόδου results\_rows\_headers().

Η results\_rows\_headers() είναι μια μέθοδος της κλάσης Table στο table.py που επιστρέφει τα rows και τα headers του αποτελέσματος μετά την εκτέλεση ενός query. Λειτουργεί σαν τη μέθοδο show() της κλάσης Table.

Αφού επιστραφούν όλες οι εγγραφές για καθεμία συνθήκη από την results\_rows\_headers(), βρίσκεται η τομή τους, δηλαδή όλες οι κοινές εγγραφές, οι οποίες εμφανίζονται στην οθόνη χρησιμοποιώντας το πακέτο tabulate.

*Παράδειγμα χρήσης:*

Select \* from instructor where salary > 85000 and dept\_name = finance ;

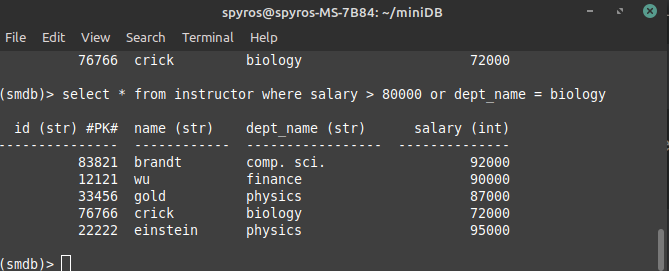
**Εικόνα 3:** Χρήση του τελεστή AND

**OR**

Ο τρόπος υλοποιήσης του τελεστή OR είναι πανομοιότυπος με του τελεστή AND. Αντί της συνάρτησης intersect() δημιουργήθηκε η συνάρτηση union() που επιστρέφει απλούστερα queries με απλούστερες συνθήκες. Το σημείο διαχωρισμού των απλούστερων συνθηκών είναι το string του τελεστή “or”. Προσομοιάζει τη λειτουργία της ένωσης (Union) της SQL. H union() καλείται στη συνάρτηση interpret(), μετά τον έλεγχο για την ύπαρξη του τελεστή “or” στο string. Επιπλέον, στο κύριο πρόγραμμα, αν η interpret() έχει επιστρέψει το string “or” και αφού βρεθούν οι εγγραφές που επιστρέφει το κάθε απλούστερο query (με τον τρόπο που περιγράφηκε στον τελεστή AND), τότε βρίσκεται η ένωση (Union) ανάμεσα σε αυτές. Πιο συγκεκριμένα, τα sets των εγγραφών ενώνονται και αφαιρούνται όσες εμφανίζονται δύο φορές, δηλαδή αφαιρούνται τα διπλότυπα. Μετά, οι εγγραφές εμφανίζονται στην οθόνη με το πακέτο tabulate.

*Παράδειγμα χρήσης:*

Select \* from instructor where salary > 80000 or dept\_name = biology ;

**Εικόνα 4:** Χρήση του τελεστή OR

1. **Θέμα 2ο**

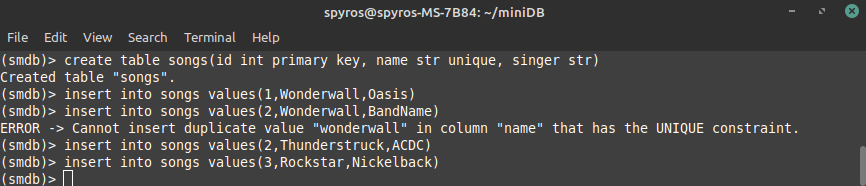
*«Enrich indexing functionality by supporting   
(a) BTree index over unique (non-PK) columns (10/50)   
(b) Hash index over PK or unique columns (10/50)»*

Πριν την επίλυση των ερωτημάτων (a) και (b) έγιναν αλλαγές ώστε να υποστηριχτεί η δήλωση του UNIQUE constraint.

Πιο συγκεκριμένα, έγινε προσθήκη της δήλωσης UNIQUE κατά τη δημιουργία πίνακα στη μέθοδο create\_table() της κλάσης Database στο αρχείο database.py. Επιπλέον, προστέθηκε η δήλωση του UNIQUE constraint στη μέθοδο \_init\_() της κλάσης Table στο table.py. Κατά την προσπάθεια εισαγωγής μιας εγγραφής σε έναν πίνακα, καλείται η μέθοδος \_insert(), όπου γίνεται έλεγχος αν καθεμία στήλη του πίνακα είναι δηλωμένη με το UNIQUE constraint και αν ναι, τότε ελέγχεται αν η τιμή της συγκεκριμένης στήλης της εγγραφής υπάρχει ήδη στον αποθηκευμένο πίνακα της ΒΔ. Αν υπάρχει, τότε εμφανίζεται μήνυμα που να ενημερώνει τον χρήστη για το γεγονός μη δυνατότητας εισαγωγής της εγγραφής και εγείρεται μια εξαίρεση – exception.

*Παράδειγμα χρήσης:*

Create table songs(id int primary key, name str unique, singer str);

**Εικόνα 5:** Δημιουργία πίνακα με στήλη με το UNIQUE constraint. Η προσπάθεια εισαγωγής εγγραφής με διπλότυπο στη UNIQUE στήλη εγείρει εξαίρεση επειδή δεν επιτρέπεται λόγω του περιορισμού.

**(a)**

Υποστήριξη ευρετηρίου Β+ Δένδρου στις στήλες με τον περιορισμό να έχουν μοναδικές τιμές (UNIQUE constraint).

Στο αρχείο mdb.py, έγινε τροποποίηση στη συνάρτηση interpret() ώστε στο λεξικό kw\_per\_action να περιλαμβάνεται στο κλειδί ‘create\_index’ και η στήλη column ως τιμή. Επίσης, στη συνάρτηση create\_query\_plan(), προστέθηκε έλεγχος αν το action είναι ‘create\_index’, οπότε και θα προστεθούν στο dic[‘on’] το string με το όνομα του πίνακα και στο dic[‘column’] το string με το όνομα της στήλης που θα δημιουργηθεί το ευρετήριο.

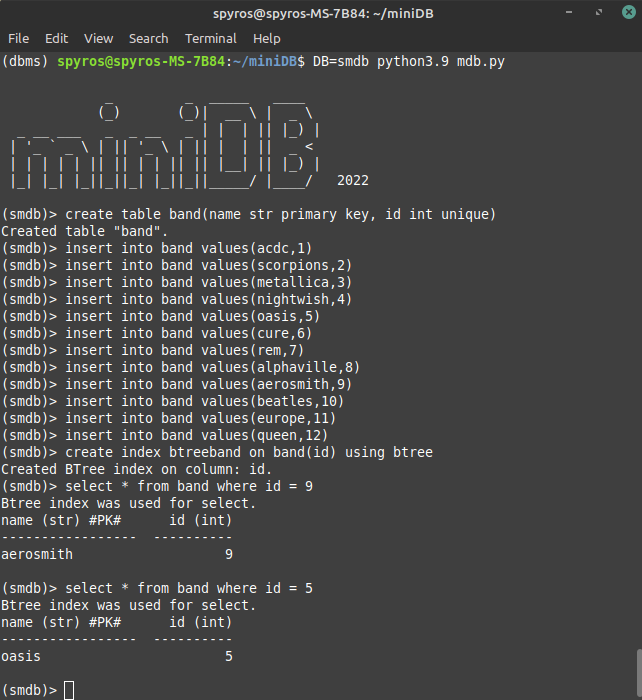
Στο αρχείο database.py, προστέθηκε το όρισμα column\_name στη μέθοδο create\_index() για το όνομα της στήλης όπου θα δημιουργείται το ευρετήριο. Αν ο χρήστης δεν έχει δηλώσει τη στήλη στην οποία θέλει να φτιάξει το ευρετήριο τότε θα δημιουργηθεί στο πρωτεύον κλειδί του πίνακα, αν υπάρχει. Αν δεν υπάρχει, τότε εγείρεται εξαίρεση-exception. Αν έχει δηλωθεί το column\_name, τότε γίνεται έλεγχος αν η στήλη αυτή είναι PK ή UNIQUE οπότε θα μπορεί να υποστηριχθεί από ευρετήριο. Αν δεν είναι τίποτα από τα δύο, τότε εγείρεται εξαίρεση και ο χρήστης ενημερώνεται για την αδυναμία δημιουργίας ευρετηρίου.

Στη μέθοδο \_construct\_index(), προστέθηκε το όρισμα column\_name για τη στήλη που θα υποστηρίζεται από ευρετήριο. Αν το column\_name είναι None, τότε δημιουργείται το ευρετήριο Β+ Δένδρου στο PK. Αν όχι, τότε το index και η τιμή της εγγραφής εισάγεται στο Β+ Δένδρο για κάθε εγγραφή της δηλωμένης στήλης.

*Παράδειγμα χρήσης:*

Create table band(name str primary key, id int unique)

Create index btreeband on band(id) using btree

**

**Εικόνα 6:** Χρήση ευρετηρίου Β+Δένδρου σε στήλη το UNIQUE constraint

**(b)**

Υποστήριξη ευρετηρίου κατακερματισμού στις στήλες που είναι πρωτεύοντα κλειδιά ή δηλωμένες με τον περιορισμό UNIQUE.

Έγινε χρήση του επεκτάσιμου κατακερματισμού της εκδοχής LSB (Least Significant Bit) με συνάρτηση κατακερματισμού που βασίζεται στο modulo %.

Η υλοποίηση του επεκτάσιμου κατακερματισμού βασίζεται στο βιβλίο “Συστήματα Διαχείρισης Βάσεων Δεδομένων” των Ramakrishnan και Gehrke.

Η **κλάση Page** στο page.py αναπαριστά έναν κάδο (ή μια σελίδα). Περιέχει τις παρακάτω μεθόδους.

Η μέθοδος \_init\_() για την αρχικοποίηση του κάδου, αρχικοποιεί μια λίστα map για τις πλειάδες (κλειδί, τιμή) που θα περιέχει ο κάδος και μια μεταβλητή local\_depth που δείχνει το τοπικό βάθος του.

Η μέθοδος full() ελέγχει αν ο κάδος είναι γεμάτος, βάσει του μήκους της λίστας map.

Η μέθοδος put() εισάγει μια πλειάδα (κλειδί, τιμή) στον κάδο. Πριν την εισαγάγει ελέγχει αν ήδη υπάρχει το κλειδί στον κάδο, οπότε διαγράφει την παλιά πλειάδα για να προσθέσει την καινούργια.

Η μέθοδος get() δέχεται ως όρισμα ένα κλειδί και αναζητεί την τιμή που αντιστοιχεί σε αυτό και την επιστρέφει.

Η μέθοδος get\_local\_high\_bit() επιστρέφει το υψηλότερο bit του τοπικού βάθους, κάνοντας ολίσθηση προς τα αριστερά της μονάδας τόσες φορές όσο το τοπικό βάθος.

Η **κλάση ExtendibleHashing** στο extendible\_hashing.py περιέχει τις κύριες λειτουργίες του επεκτάσιμου κατακερματισμού. Ακολουθούν οι μέθοδοι της κλάσης.

Η μέθοδος \_init\_() αρχικοποιεί το ολικό βάθος (έστω d) ίσο με 0 και τον αρχικό κατάλογο να είναι μια λίστα με ένα αντικείμενο της κλάσης Page.

Η μέθοδος get\_page() δέχεται ένα κλειδί, υπολογίζει με την ενσωματωμένη συνάρτηση hash() την τιμή h και μετά επιστρέφει από τον κατάλογο την σελίδα Page με index = (h % 2d).

Η μέθοδος put() υλοποιεί τη λειτουργία προσθήκης μιας πλειάδας (κλειδί, τιμή) στο κατακερματισμένο αρχείο. Πρώτα, καλείται η μέθοδος get\_page(), ώστε να βρεθεί ο κάδος όπου θα πρέπει να μπει η τιμή και μετά καλείται η μέθοδος full() ώστε να ελεγχθεί αν ο κάδος είναι γεμάτος. Η τιμή εισάγεται στον κάδο με τη μέθοδο put() της κλάσης Page. Αν ο κάδος είναι γεμάτος, τότε πρέπει να διασπαστεί. Αν το τοπικό βάθος του κάδου (έστω l) είναι ίσο με το ολικό βάθος (d) του καταλόγου, τότε το μήκος του καταλόγου διπλασιάζεται και το ολικό βάθος αυξάνεται κατά 1. Στη συνέχεια, δημιουργούνται δύο νέα αντικείμενα της κλάσης Page που αναπαριστούν τους δύο νέους κάδους που προκύπτουν από τη διάσπαση του αρχικού. Το τοπικό του βάθος ισούται με το τοπικό βάθος του αρχικού κάδου αυξημένο κατά 1. Μετά, γίνεται εύρεση του υψηλότερου bit του τοπικού βάθους (έστω high\_bit) της σελίδας χρησιμοποιώντας τη μέθοδο get\_local\_high\_bit() της κλάσης Page. Για κάθε πλειάδα της σελίδας υπολογίζεται η τιμή hash (h) του κλειδιού της και με βάση τον τύπο (h % (high\_bit + 1)) γίνεται προσθήκη της εγγραφής σε έναν από τους δύο νέους κάδους που προκύπτουν από τη διάσπαση. Στο τέλος, οι δύο νέοι κάδοι προστίθενται στον κατάλογο.

Η μέθοδος get() δέχεται ένα κλειδί και επιστρέφει την τιμή που αντιστοιχεί σε αυτό.

Για την ένταξη της λειτουργίας του επεκτάσιμου κατακερματισμού στο υπόλοιπο πρόγραμμα έγιναν τα παρακάτω.

Αρχικά, προστέθηκε έλεγχος στη μέθοδο create\_index() της κλάσης Database στο database.py, ώστε αν ο χρήστης έχει δηλώσει μετά το using ότι θέλει ευρετήριο hash, τότε να καλείται η μέθοδος \_construct\_index\_hash() που θα το δημιουργεί.

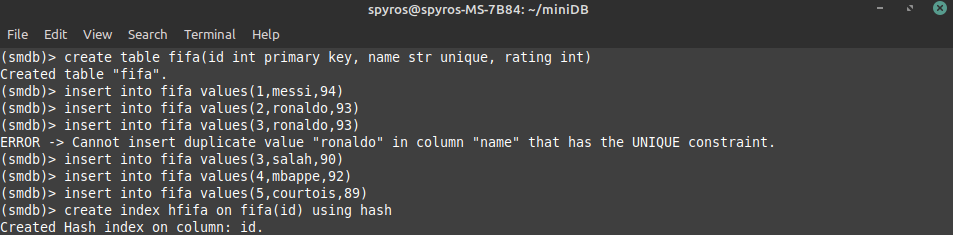
Η \_construct\_index\_hash() έχει ως ορίσματα το όνομα του πίνακα στον οποίο θα δημιουργηθεί το ευρετήριο, το όνομα του ευρετηρίου και η στήλη που θα αφορά. Στην αρχή, δημιουργείται ένα αντικείμενο της κλάσης ExtendibleHashing και μετά για κάθε εγγραφή στη στήλη, εισάγεται στο ευρετήριο μια πλειάδα (κλειδί, δείκτης). Στο τέλος, καλώντας τη μέθοδο \_save\_index() αποθηκεύεται το ευρετήριο στον δίσκο.

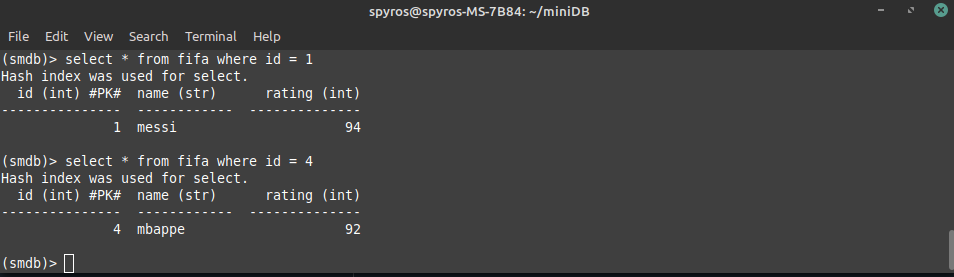
Για τη **λειτουργία της επιλογής** έγιναν αλλαγές στη μέθοδο select() της κλάσης Database στο αρχείο database.py. Έχουν προστεθεί έλεγχοι αν η στήλη που χρησιμοποιείται στη συνθήκη της επιλογής είναι πρωτεύον κλειδί ή δηλωμένη με τον περιορισμό UNIQUE. Μετά τους ελέγχους, γίνεται δοκιμή εκτέλεσης της επιλογής με χρήση ευρετηρίου Β+ Δένδρου και αν προκύψει κάποια εξαίρεση-exception, τότε αυτό το διαχειρίζεται η δομή try-except οπότε θα εκτελεστεί αναζήτηση με το ευρετήριο κατακερματισμού. Για την αναζήτηση με hash ευρετήριο δημιουργήθηκε η μέθοδος \_select\_where\_with\_hash() στην κλάση Table στο table.py.

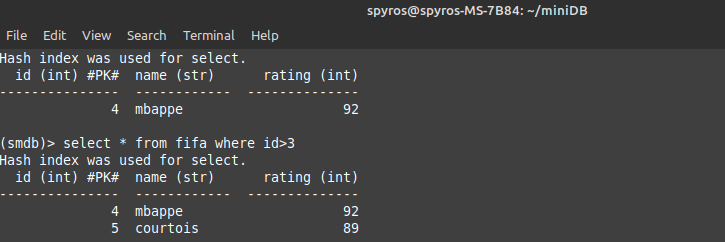
Η μέθοδος \_select\_where\_with\_hash() έχει παρόμοια λειτουργία με την \_select\_where\_with\_btree() για το Β+ Δένδρο. Αφού βρεθούν η στήλη που θα γίνει η αναζήτηση, ο τελεστής και η τιμή της συνθήκης αναζήτησης, γίνεται έλεγχος αν η ερώτηση είναι ταυτότητας ή διαστήματος. Αν ο τελεστής είναι (>, <, <=, >=), επειδή το ευρετήριο κατακερματισμού δεν υποστηρίζει ερωτήσεις διαστήματος, τότε θα γίνει σειριακή αναζήτηση για να βρεθούν οι εγγραφές. Διαφορετικά, χρησιμοποιείται η μέθοδος get() της κλάσης ExtendibleHashing για να βρεθεί ο δείκτης της εγγραφής που ικανοποιεί τη συνθήκη αναζήτησης.

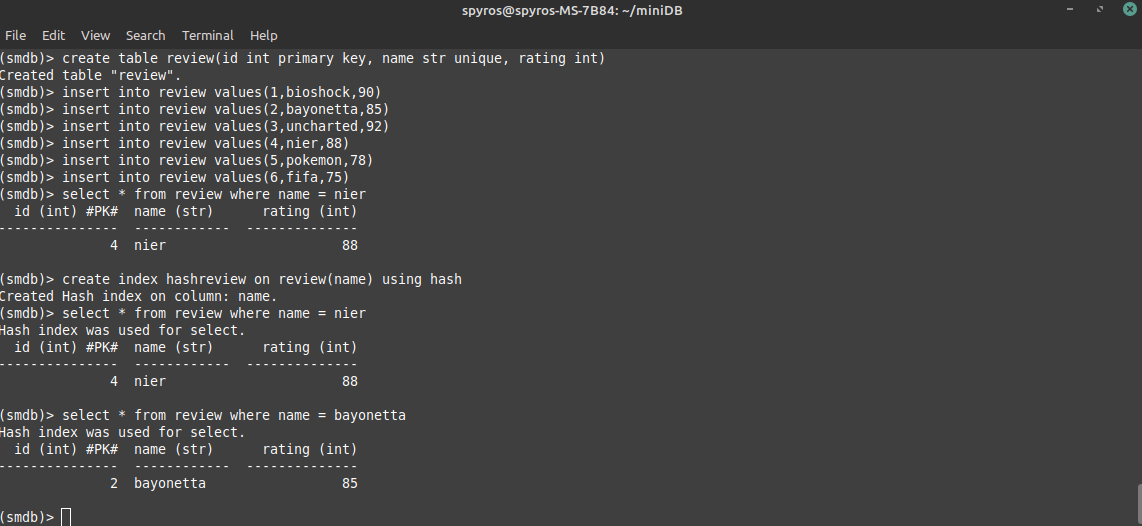
*Παραδείγματα χρήσης:*

Create index hfifa on fifa(id) using hash

**Εικόνα 7:** Δημιουργία ευρετηρίου κατακερματισμού στο Primary Key

**Εικόνα 8:** Αναζήτηση με ερώτηση ταυτότητας στη στήλη του PK που υποστηρίζεται από hash index.

**Εικόνα 9:** Αν γίνει ερώτηση διαστήματος στη στήλη που υποστηρίζεται από ευρετήριο κατακερματισμού τότε θα γίνει σειριακή αναζήτηση για να βρεθεί το αποτέλεσμα

**Εικόνα 10:** Αναζήτηση με ερώτηση ταυτότητας στη στήλη με το UNIQUE constraint που υποστηρίζεται από hash index.

1. **Βιβλιογραφία**

**Βιβλία – Συγγράμματα:**

* «Συστήματα Βάσεων Δεδομένων», A. Silberschatz, H.F. Korth, S. Sudarshan, εκδόσεις Μ. Γκιούρδα
* «Συστήματα Διαχείρισης Βάσεων Δεδομένων», R. Ramakrishnan, J. Gehrke, εκδόσεις Τζιόλα
* «Δομές Δεδομένων, Αλγόριθμοι και Εφαρμογές στη C++», S. Sahni, εκδόσεις Τζιόλα
* Σημειώσεις διαλέξεων και εργαστηρίων μαθήματος, Ι. Θεοδωρίδη, Ι. Κοντούλη, Γ. Θεοδωρόπουλου

**Ιστοσελίδες:**

* <https://en.wikipedia.org/wiki/Extendible_hashing>, Βικιπαίδεια η ελεύθερη Εγκυκλοπαίδεια
* <https://gist.github.com/Chaser324/ce0505fbed06b947d962>, GitHub Standard Fork & Pull Request Workflow