

**Πανεπιστήμιο Πειραιώς
Τμήμα Πληροφορικής**



ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

UNIVERSITY OF PIRAEUS

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

Ατομική Εργασία

Όνομα: Σπυρίδων

Επώνυμο: Σολανάκης

Αριθμός Μητρώου (ΑΜ): Π18141

Εξάμηνο: 5^ο

Ακαδημαϊκό Έτος: 2022-23

Ημερομηνία Παράδοσης: 17/2/2023

Περιεχόμενα

Εισαγωγή	σελ. 3
Θέμα 1ο (α)	σελ. 4
Θέμα 1ο (β)	σελ. 6
Θέμα 2ο (Unique)	σελ. 9
Θέμα 2ο (α)	σελ. 10
Θέμα 2ο (β)	σελ. 12
Βιβλιογραφία	σελ. 17

1. Εισαγωγή

Στο πλαίσιο του μαθήματος «Συστήματα Διαχείρισης Βάσεων Δεδομένων» του 5^{ου} 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.

2. 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.

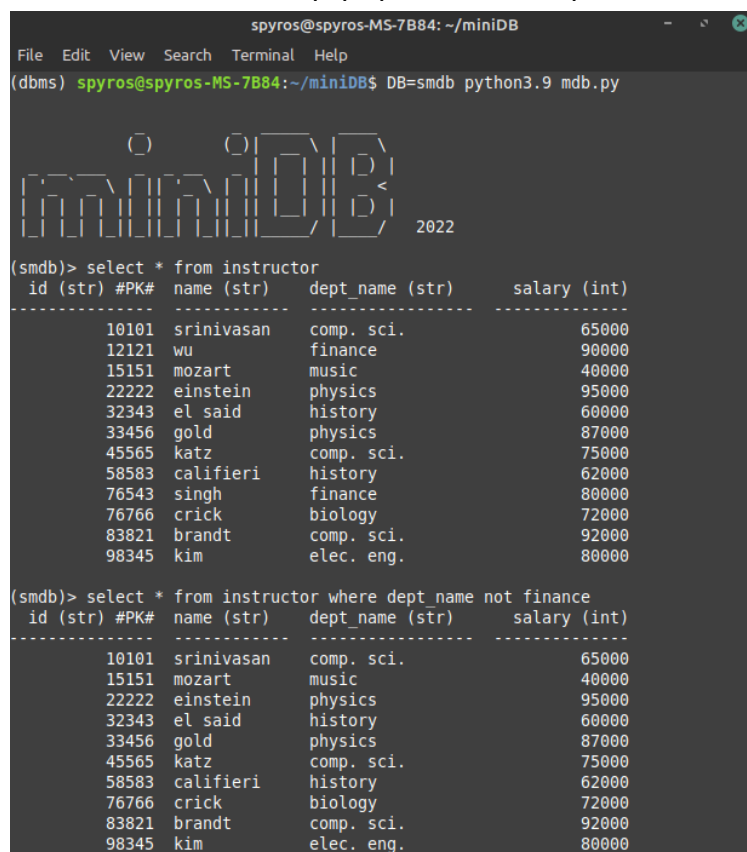
NOT

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



```
spyros@spyros-MS-7B84: ~/miniDB
File Edit View Search Terminal Help
(dbms) spyros@spyros-MS-7B84:~/miniDB$ DB=smbd python3.9 mdb.py

miniDB 2022

(smbd)> select * from instructor
id (str) #PK# name (str) dept_name (str) salary (int)
-----
10101 srinivasan comp. sci. 65000
12121 wu finance 90000
15151 mozart music 40000
22222 einstein physics 95000
32343 el said history 60000
33456 gold physics 87000
45565 katz comp. sci. 75000
58583 califieri history 62000
76543 singh finance 80000
76766 crick biology 72000
83821 brandt comp. sci. 92000
98345 kim elec. eng. 80000

(smbd)> select * from instructor where dept_name not finance
id (str) #PK# name (str) dept_name (str) salary (int)
-----
10101 srinivasan comp. sci. 65000
15151 mozart music 40000
22222 einstein physics 95000
32343 el said history 60000
33456 gold physics 87000
45565 katz comp. sci. 75000
58583 califieri history 62000
76766 crick biology 72000
83821 brandt comp. sci. 92000
98345 kim elec. eng. 80000
```

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`.

Η 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;
```

```
spyros@spyros-MS-7B84: ~/miniDB
File Edit View Search Terminal Help

(smdb)> select * from instructor where salary between 45000 and 75000
id (str) #PK# name (str) dept_name (str) salary (int)
-----
10101 srinivasan comp. sci. 65000
32343 el said history 60000
45565 katz comp. sci. 75000
58583 califieri history 62000
76766 crick biology 72000
```

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.

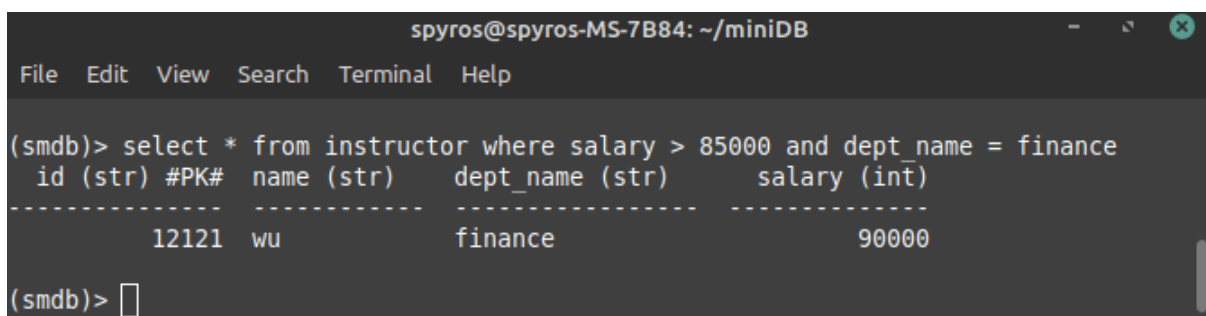
Στο κύριο πρόγραμμα, μετά την κλήση της `interpret()`, γίνεται έλεγχος για το αν έχει επιστραφεί ο τελεστής AND, οπότε για κάθε query plan που επέστρεψε, γίνεται κλήση της `execute_dic()`. Με βάση τα αποτελέσματά της, γίνεται κλήση της μεθόδου `results_rows_headers()`.

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 ;



```
spyros@spyros-MS-7B84: ~/miniDB
File Edit View Search Terminal Help
(smdb)> select * from instructor where salary > 85000 and dept_name = finance
  id (str) #PK#  name (str)    dept_name (str)    salary (int)
-----
      12121  wu          finance           90000
(smdb)> 
```

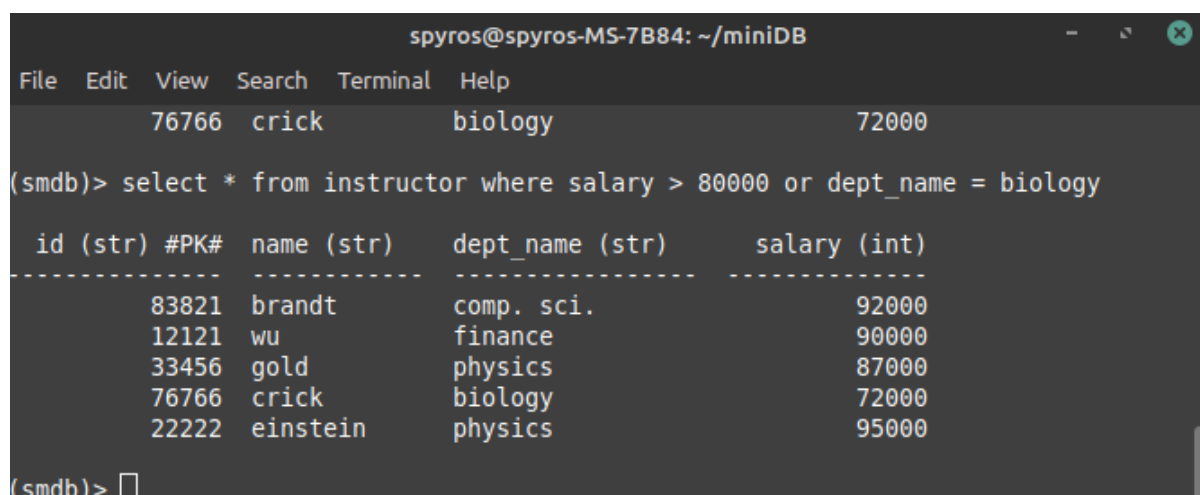
Εικόνα 3: Χρήση του τελεστή AND

OR

Ο τρόπος υλοποίησης 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. Η `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 ;
```



The screenshot shows a terminal window titled "spyros@spyros-MS-7B84: ~/miniDB". The window contains a SQL query and its results. The query is: `(smdb)> select * from instructor where salary > 80000 or dept_name = biology`. The results are displayed in a table with columns: `id (str)`, `#PK#`, `name (str)`, `dept_name (str)`, and `salary (int)`. The results are as follows:

id (str)	#PK#	name (str)	dept_name (str)	salary (int)
76766		crick	biology	72000
83821		brandt	comp. sci.	92000
12121		wu	finance	90000
33456		gold	physics	87000
76766		crick	biology	72000
22222		einstein	physics	95000

Εικόνα 4: Χρήση του τελεστή OR

3. Θέμα 2ο

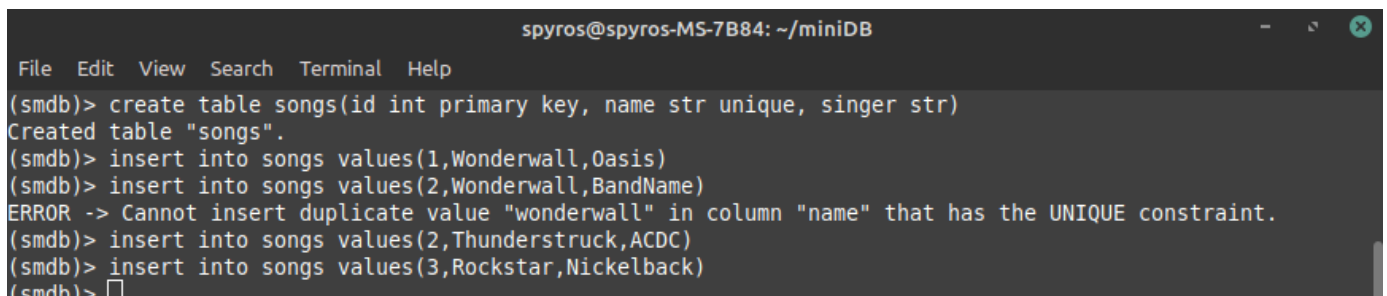
*«Enrich indexing functionality by supporting
(a) BTree index over unique (non-PK) columns (10/50)
(b) Hash index over PK or unique columns (10/50)»*

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);



```
spyros@spyros-MS-7B84: ~/miniDB
File Edit View Search Terminal Help
(smdb)> create table songs(id int primary key, name str unique, singer str)
Created table "songs".
(smdb)> insert into songs values(1,Wonderwall,Oasis)
(smdb)> insert into songs values(2,Wonderwall,BandName)
ERROR -> Cannot insert duplicate value "wonderwall" in column "name" that has the UNIQUE constraint.
(smdb)> insert into songs values(2,Thunderstruck,ACDC)
(smdb)> insert into songs values(3,Rockstar,Nickelback)
(smdb)>
```

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 B+ 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.

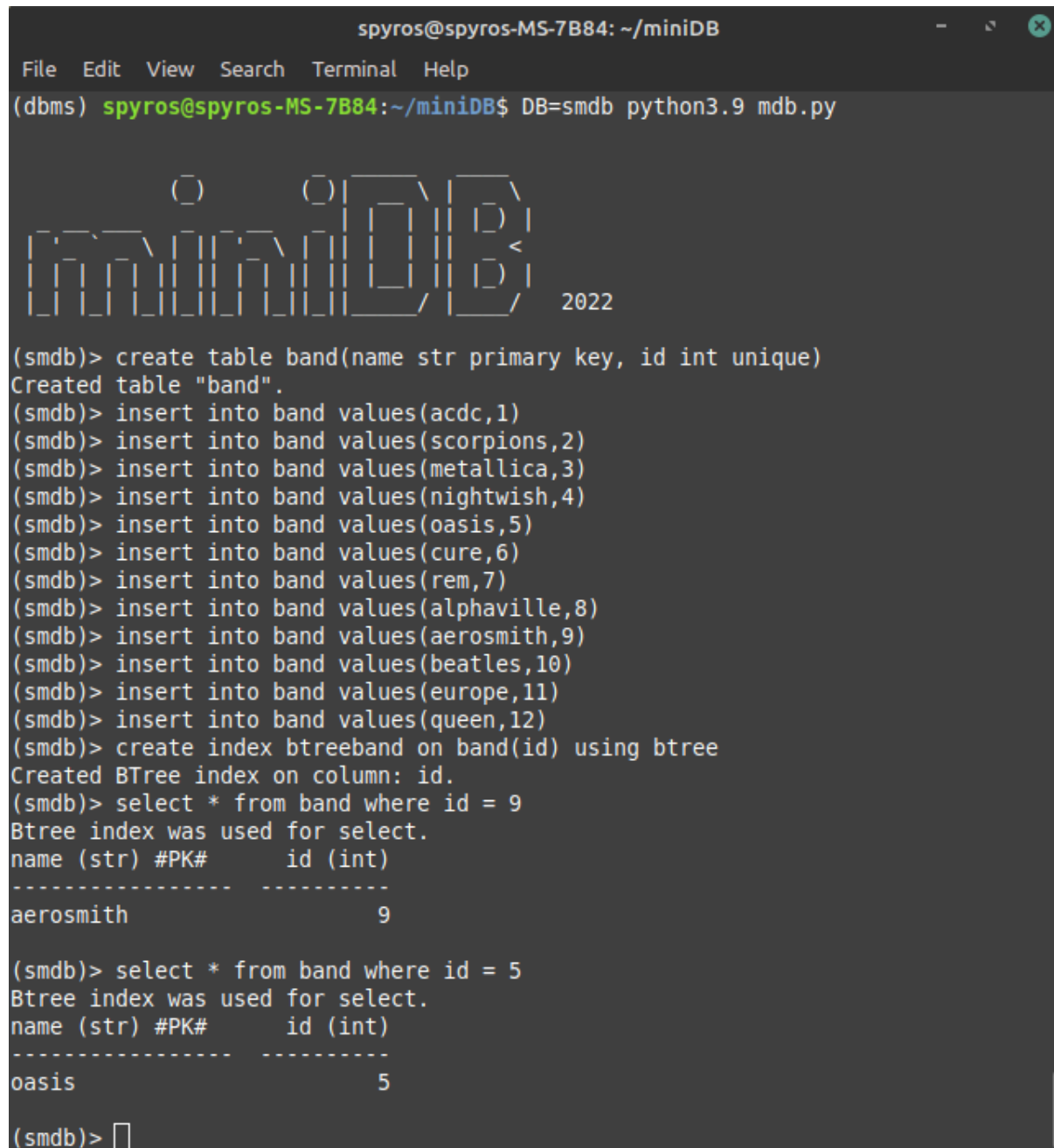
Στο αρχείο `database.py`, προστέθηκε το όρισμα `column_name` στη μέθοδο `create_index()` για το όνομα της στήλης όπου θα δημιουργείται το ευρετήριο. Αν ο χρήστης δεν έχει δηλώσει τη στήλη στην οποία θέλει να φτιάξει το ευρετήριο τότε θα δημιουργηθεί στο πρωτεύον κλειδί του πίνακα, αν υπάρχει. Αν δεν 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 B+ Δένδρου στο PK. Αν όχι, τότε το `index` και η τιμή της εγγραφής εισάγεται στο B+ 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



The screenshot shows a terminal window titled 'spyros@spyros-MS-7B84: ~/miniDB'. The user runs a command to start a database interface: `(dbms) spyros@spyros-MS-7B84:~/miniDB$ DB=smbd python3.9 mdb.py`. Above the terminal output, there is a diagram of a B-tree structure. The tree has a root node with two pointers to leaf nodes. The left leaf node contains the values 'a', 'c', 'd', 'c' and the right leaf node contains 's', 'c', 'o', 'r', 'p', 'i', 'o', 'n', 's'. The year '2022' is written to the right of the diagram. The terminal output shows the following commands and results:

```
(smbd)> create table band(name str primary key, id int unique)
Created table "band".
(smbd)> insert into band values(acdc,1)
(smbd)> insert into band values(scorpions,2)
(smbd)> insert into band values(metallica,3)
(smbd)> insert into band values(nightwish,4)
(smbd)> insert into band values(oasis,5)
(smbd)> insert into band values(cure,6)
(smbd)> insert into band values(rem,7)
(smbd)> insert into band values(alphaville,8)
(smbd)> insert into band values(aerosmith,9)
(smbd)> insert into band values(beatles,10)
(smbd)> insert into band values(europe,11)
(smbd)> insert into band values(queen,12)
(smbd)> create index btreeband on band(id) using btree
Created BTree index on column: id.
(smbd)> select * from band where id = 9
Btree index was used for select.
name (str) #PK#      id (int)
-----
aerosmith          9

(smbd)> select * from band where id = 5
Btree index was used for select.
name (str) #PK#      id (int)
-----
oasis              5

(smbd)>
```

Εικόνα 6: Χρήση ευρετηρίου B+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 $index = (h \% 2^d)$.

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 B+ 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 B+ 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

```
spyros@spyros-MS-7B84: ~/miniDB
File Edit View Search Terminal Help
(smdb)> create table fifa(id int primary key, name str unique, rating int)
Created table "fifa".
(smdb)> insert into fifa values(1,messi,94)
(smdb)> insert into fifa values(2,ronaldo,93)
(smdb)> insert into fifa values(3,ronaldo,93)
ERROR -> Cannot insert duplicate value "ronaldo" in column "name" that has the UNIQUE constraint.
(smdb)> insert into fifa values(3,salah,90)
(smdb)> insert into fifa values(4,mbappe,92)
(smdb)> insert into fifa values(5,courtois,89)
(smdb)> create index hfifa on fifa(id) using hash
Created Hash index on column: id.
```

Εικόνα 7: Δημιουργία ευρετηρίου κατακερματισμού στο Primary Key

```
spyros@spyros-MS-7B84: ~/miniDB
File Edit View Search Terminal Help
(smdb)> select * from fifa where id = 1
Hash index was used for select.
  id (int) #PK#  name (str)      rating (int)
-----
      1  messi              94

(smdb)> select * from fifa where id = 4
Hash index was used for select.
  id (int) #PK#  name (str)      rating (int)
-----
      4  mbappe              92

(smdb)> 
```

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

```
spyros@spyros-MS-7B84: ~/miniDB
File Edit View Search Terminal Help
Hash index was used for select.
  id (int) #PK#  name (str)      rating (int)
-----
         4  mbappe                92

(smdb)> select * from fifa where id>3
Hash index was used for select.
  id (int) #PK#  name (str)      rating (int)
-----
         4  mbappe                92
         5  courtois              89
```

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

```
spyros@spyros-MS-7B84: ~/miniDB
File Edit View Search Terminal Help
(smdb)> create table review(id int primary key, name str unique, rating int)
Created table "review".
(smdb)> insert into review values(1,bioshock,90)
(smdb)> insert into review values(2,bayonetta,85)
(smdb)> insert into review values(3,uncharted,92)
(smdb)> insert into review values(4,nier,88)
(smdb)> insert into review values(5,pokemon,78)
(smdb)> insert into review values(6,fifa,75)
(smdb)> select * from review where name = nier
  id (int) #PK#  name (str)      rating (int)
-----
         4  nier                88

(smdb)> create index hashreview on review(name) using hash
Created Hash index on column: name.
(smdb)> select * from review where name = nier
Hash index was used for select.
  id (int) #PK#  name (str)      rating (int)
-----
         4  nier                88

(smdb)> select * from review where name = bayonetta
Hash index was used for select.
  id (int) #PK#  name (str)      rating (int)
-----
         2  bayonetta            85

(smdb)> █
```

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

4. 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/ce0505fbbed06b947d962>, GitHub Standard Fork & Pull Request Workflow