ΣΥΣΤΗΜΑΤΑ ΔΙΑΧΕΙΡΙΣΗΣ ΒΑΣΕΩΝ ΔΕΔΟΜΕΝΩΝ

Εργασία miniDB

Νικολέτα-Χαρά Βλάχου-Σακελλαρίου Π19028
8ο εξάμηνο

Περιεχόμενα

Issue 1 - Enrich WHERE statement

1.1.1 NOT operator - Υλοποίηση

```
# if condition is None, return all rows
# if not, return the rows with values where condition is met for value
if condition is not None:
    # check if a not statement is present in the condition, if so reverse condition
    if'not' in condition:
        print('not is  in condition ', condition)
        condition = condition[4:]
        print(condition)
        column_name, operator, value = self._parse_condition(condition)
        column = self.column_by_name(column_name)
        rows = [ind for ind, x in enumerate(column) if get_op(reverse_op(operator), x, value)]
```

Εικόνα – Προγραμματιστική υλοποίηση λογικής πράξης NOT

Προκειμένου να υλοποιηθεί η λογική πράξη NOT θεωρήθηκε αναγκαίο να τροποποιηθεί το αρχείο table.py. Αναλυτικότερα, στο αρχείο table.py στην μέθοδο _select_where() έχει προστεθεί ο κατάλληλος έλεγχος για την παρουσία της λέξης not , η εύρεση της οποίας γνωστοποιείται στον χρήστη με ένα print statement. Στην συνέχεια, πραγματοποιείται διαχωρισμός μεταξύ του ονόματος της στήλης, της λογικής πράξης και της τιμής με την μέθοδο _parse_condition() από την οποία έχει αφαιρεθεί ο όρος not. Τέλος για κάθε στήλη του πίνακα που έχει επιλεχθεί ,ελέγχεται εάν επαληθεύει την αντίθετη συνθήκη από αυτήν που δόθηκε με την μέθοδο get_op(reverse_op(operator)) ,έτσι ώστε να υπολοποιεί την πράξη NOT.

1.1.2 NOT operator – Παράδειγμα Εκτέλεσης

```
(smdb)> select * from instructor where not salary > 70000
not is  in condition  not salary > 70000
salary > 70000
  id (str) #PK#  name (str)    dept_name (str)      salary (int)
---------------  ------------  -----------------  --------------
         10101  srinivasan    comp. sci.                  65000
         15151  mozart        music                       40000
         32343  el said       history                     60000
         58583  califieri     history                     62000
```

Εικόνα – Παράδειγμα Εκτέλεσης NOT operator

Στο παραπάνω παράδειγμα εκτέλεσης έχει επιλεχθεί ο πίνακας instructor από την βάση δεδομένων που έχει δοθεί. Πραγματοποιείται το αίτημα να επιστραφούν οι στήλες όπου η τιμή salary δεν ξεπερνά τον ακέραιο αριθμό 70000 . Τελικά οι στήλες που ταιριάζουν στο παραπάνω αίτημα επιστρέφονται με επιτυχία.

1.2.1 BETWEEN operator

```
for op_key in ops.keys():
    if 'between' in condition:
        splt = condition.split('between')
        left, right = splt[0].strip(), splt[1].strip()
        return left, 'between', right
```

```
elif 'between' in condition:
    print('between operator found')
    column_name, operator, value = split_condition(condition)
    values = value.split('and')
    start_value = values[0].strip()
    end_value = values[1].strip()
    print('column name:', column_name, 'operator:', operator, 'values:', start_value, " ,", end_value)
    column = self.column_by_name(column_name)
    rows = [ind for ind, x in enumerate(column) if int(start_value) <= x <= int(end_value)]
```

Προκειμένου να υλοποιηθεί η λογική πράξη BETWEEN θεωρήθηκε αναγκαίο να τροποποιηθούν τα αρχεία table.py και misc.py. Στο αρχείο misc.py έχει προστεθεί η παραπάνω λογική συνθήκη η οποία εντωπίζει την ύπαρξη της λογικής πράξης BETWEEN σε μια πρόταση και επιστρέφει τις τιμές μεταξύ των οποίων θα γίνει η πράξη.

1.2.2 BETWEEN operator – Παράδειγμα Εκτέλεσης

```
(smdb)> select * from instructor where salary between 70000 and 90000
between operator found
columnt name: salary operator: between values: 70000  , 90000
  id (str) #PK#  name (str)    dept_name (str)      salary (int)
---------------  ------------  -----------------  --------------
         12121  wu            finance                       90000
         33456  gold          physics                       87000
         45565  katz          comp. sci.                    75000
         76543  singh         finance                       80000
         76766  crick         biology                       72000
         98345  kim           elec. eng.                    80000

(smdb)>
```

Εικόνα – Παράδειγμα Εκτέλεσης BETWEEN operator

Στο παραπάνω παράδειγμα εκτέλεσης έχει επιλεχθεί ο πίνακας instructor από την βάση δεδομένων που έχει δοθεί. Πραγματοποιείται το αίτημα να επιστραφούν οι στήλες όπου η τιμή salary κυμαίνεται ανάμεσα στις ακέραιες τιμές 70000 και 90000 . Τελικά οι στήλες που ταιριάζουν στο παραπάνω αίτημα επιστρέφονται με επιτυχία.

1.3.1 AND operator – Υλοποίηση

```
for op_key in ops.keys():
```

```
if 'and' in condition:
    splt = condition.split('and')
    print('and found ', splt[0], splt[1])
    return splt
```

```
elif 'and' in condition:
    print('and is in condition ')
    conditions = split_condition(condition)
    rows = []
    for con in conditions:
        column_name, operator, value = self._parse_condition(con)
        print('columnt name:', column_name, 'operator:', operator, 'value: ', value)
        column = self.column_by_name(column_name)
        rowsTemp = [ind for ind, x in enumerate(column) if get_op(operator, x, value)]
        rows.append(rowsTemp)
    rowsTemp = []
    # flatten 2d array to 1d array
    for i in rows[0]:
        for j in rows[1]:
            if i == j:
                rowsTemp.append(i)
    print(rowsTemp)
    # remove duplicates
    res = []
    [res.append(x) for x in rowsTemp if x not in res]
    print(res)
    rows = res
```

Εικόνα – Προγραμματιστική υλοποίηση λογικής πράξης AND

Προκειμένου να υλοποιηθεί η λογική πράξη AND θεωρήθηκε αναγκαίο να τροποποιηθεί το αρχείο table.py και misc.py. Αναλυτικότερα, στο αρχείο table.py στην μέθοδο _select_where() έχει προστεθεί ο κατάλληλος έλεγχος για την παρουσία της λέξης and, η εύρεση της οποίας γνωστοποιείται στον χρήστη με ένα print statement. Στην συνέχεια, πραγματοποιείται διαχωρισμός μεταξύ του ονόματος της στήλης, της λογικής πράξης και της τιμής με την μέθοδο _split_condition(). Έπειτα η υλοποίηση συνδυάζει τις γραμμές που πληρούν τις ζητούμενες συνθήκες .Τέλος, όσες γραμμές εμφανίζονται πάνω από μια φορά διαγράφονται.

```
(smdb)> select * from instructor where salary = 40000 and salary<90000
and found  salary = 40000   salary<90000
and is in condition
and found  salary = 40000   salary<90000
columnt name: salary operator: = value:  40000
columnt name: salary operator: < value:  90000
[2]
[2]
  id (str) #PK#  name (str)    dept_name (str)      salary (int)
---------------  ------------  -----------------  --------------
          15151  mozart        music                       40000
```

Εικόνα – Παράδειγμα Εκτέλεσης AND operator

Στο παραπάνω παράδειγμα εκτέλεσης έχει επιλεχθεί ο πίνακας instructor από την βάση δεδομένων που έχει δοθεί. Πραγματοποιείται το αίτημα να επιστραφούν οι στήλες όπου η τιμή salary ικανοποιεί την συνθήκη salary=40000 και την συνθήκη salary<90000. Τελικά οι στήλες που ταιριάζουν στο παραπάνω αίτημα επιστρέφονται με επιτυχία, χωρίς επαναλήψεις.

1.4.1 OR operator – Υλοποίηση

```
if 'or_condition' in condition:
    splt = condition.split('or_condition')
    print('or found ', splt[0], splt[1])
    return splt
```

```
elif ('or_condition' in condition):

    print('or is in condition ')
    conditions = split_condition(condition)
    rows = []
    for con in conditions:
        column_name, operator, value = self._parse_condition(con)
        print('columnt name:', column_name, 'operator:', operator, 'value: ', value)
        column = self.column_by_name(column_name)
        rowsTemp = [ind for ind, x in enumerate(column) if get_op(operator, x, value)]
        rows.append(rowsTemp)

    # flatten 2d array to 1d array
    rows = [j for sub in rows for j in sub]
    print(rows)
    # remove duplicates
    res = []
    [res.append(x) for x in rows if x not in res]
    print(res)
    rows = res
```

Προκειμένου να υλοποιηθεί η λογική πράξη OR θεωρήθηκε αναγκαίο να τροποποιηθεί το αρχείο table.py και misc.py. Αναλυτικότερα, στο αρχείο table.py στην μέθοδο _select_where() έχει προστεθεί ο κατάλληλος έλεγχος για την παρουσία της λέξης or, η εύρεση της οποίας γνωστοποιείται στον χρήστη με ένα print statement. Στην συνέχεια, πραγματοποιείται διαχωρισμός μεταξύ του ονόματος της στήλης, της λογικής πράξης και της τιμής με την μέθοδο _split_condition(). Έπειτα η υλοποίηση

συνδυάζει τις γραμμές που πληρούν τουλάχιστον μια από τις ζητούμενες συνθήκες .Τέλος, όσες γραμμές εμφανίζονται πάνω από μια φορά διαγράφονται.

```
(smdb)> select * from instructor where salary = 40000 or_condition salary= 90000
or found  salary = 40000   salary= 90000
or is in condition
or found  salary = 40000   salary= 90000
columnt name: salary operator: = value:  40000
columnt name: salary operator: = value:  90000
[2, 1]
[2, 1]
  id (str) #PK#  name (str)    dept_name (str)      salary (int)
--------------  ------------  -----------------  --------------
         15151  mozart        music                       40000
         12121  wu            finance                     90000
```

Εικόνα – Παράδειγμα Εκτέλεσης OR operator

Στο παραπάνω παράδειγμα εκτέλεσης έχει επιλεχθεί ο πίνακας instructor από την βάση δεδομένων που έχει δοθεί. Πραγματοποιείται το αίτημα να επιστραφούν οι στήλες όπου η τιμή salary ικανοποιεί είτε την σνυθήκη salary=40000 , είτε την συνθήκη salary<90000. Τελικά οι στήλες που ταιριάζουν στο παραπάνω αίτημα επιστρέφονται με επιτυχία, χωρίς επαναλήψεις.

Issue 2 - Enrich indexing functionality

Προσθήκη δυνατότητας δημιουργίας unique στηλών

```
if 'unique' in args:
    arglist  = args[1:-1].replace(',','')
    arglist = arglist.split(' ')
    print(" Unique condition detected")
    dic['unique'] = arglist[arglist.index('unique')-2]
else:
    dic['unique'] = None
```

```
            if unique_cols is not None:
                self.unique_index =
self.column_names.index(unique_cols)
            else:
                self.unique_index = None
            self.unique = unique_cols
```

```
insert into unique_facts values(A,SECRET);
insert into unique_facts values(B,SECRET2);
insert into unique_facts values(C,SECRET3);
insert into unique_facts values(D,SECRET4);
```

Προκειμένου να είναι δυνατή η δημιουργία ευρετηρίων σε μη primary key στήλες χρειάστηκε να υλοποιηθεί η δυνατότητα δημιουργίας unique στηλών. Για αυτόν τον σκοπό τροποποιήθηκαν τα αρχεία database.py ,mdb.py ,table.py και έγινε η προσθήκη του πίνακα unique_facts ο οποίος διαθέτει unique εγγραφές.

Hash index

```
This list  [1, 2, 3, 4, 5, 6, 7, 8]
Is stored in a hash table, with extendable hashing in buckets of size = 3
Due to the bucket size finding an element will take O(3) instead of O(1)
The hash table is the following:
key 0 Bucket  [8]
key 1 Bucket  [1, 5]
key 10 Bucket  [2]
key 11 Bucket  [3, 7]
key 100 Bucket  [4]
key 101 Bucket  []
key 110 Bucket  [6]
key 111 Bucket  []
The value is successfully found!
8
```

Εικόνα – Hash index με extendible hashing

Εδώ έχει γίνει η υλοποίηση του hash table με extendible hashing. Παραπάνω φαίνεται ένα παράδειγμα μιας λίστας η οποία καταχωρείτε σε ένα πίνακα και στην συνέχεια γίνεται η ανάκτηση της τιμής ‘8’ σε χρονική πολυπλοκότητα O(1). Η δυνατότητα αυτή έχει υλοποιηθεί ως ξεχωριστό module. Η υλοποίηση βρίσκεται στο αρχείο indexHash.py.

Issue 3 - Implement miniDB’s query optimizer

Δημιουργία Sql to RA parser

```
Input sql ->  SELECT *
FROM instructor AS I
INNER JOIN teaches AS T ON I.ID = T.ID
WHERE I.dept_name = “Music” AND T.year = 2009
RA ->  ['σ', 'I.ID', '=', 'T.ID', 'I.dept_name', '=', '“Music”', '^', 'T.year', '=', '2009', 'I', '⋈', 'T', '']
```

Εικόνα – SQL to RA parser

Για τη δημιουργία του παραπάνω μετατροπαία δημιουργήθηκε το ξεχωριστό αρχείο query_optimiser.py. Χωρίζει και επεξεργάζεται εντολές sql και τις μετατρέπει σε σχεσιακή άλγεβρα.