ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Εργασία Συστήματα Διαχείρισης Βάσεων Δεδομένων

Π19006 Αλεξανδρής Ιωάννης
Π19127 Παγίδας Νίκος
Π19128 Παλατσίδης Αλέξανδρος

Υλοποιήθηκαν τα δύο πρώτα issues. Δηλαδή προστέθηκε η λειτουργία του NOT, BETWEEN, AND και OR για το πρώτο issue και BTree index σε unique στήλες και Hash index σε primary key και unique στήλες του πίνακα για το δεύτερο issue. Για το δεύτερο issue προστέθηκε και η δήλωση μιας στήλης ενός πίνακα ως unique.

Στο fork που έγινε φαίνονται αλλαγές από ένα άτομο επειδή τα μέλη της ομάδας διαθέτουν δύο windows υπολογιστές και έναν mac υπολογιστή έτσι για την διευκόλυνση μας η εργασία έγινε από έναν υπολογιστή ενώ βρισκόμασταν είτε από κοντά είτε από κάποια πλατφόρμα επικοινωνίας (Discord, MS Teams).

#1 (a)

Για την προσθήκη της λειτουργίας not έγινε αλλαγή στο αρχείο table.py όπου στις συναρτήσεις _select_where, _select_where_with_btree, _select_where_with_hash, _update_rows και _delete_where προστέθηκε ο έλεγχος αν υπάρχει το not στο condition. Στην περίπτωση που υπάρχει τρέχει ο παρακάτω κώδικας:

```
column_name, operator, value = self._parse_condition(condition.replace('not ', ''))
column = self.column_by_name(column_name)
rows = [ind for ind, x in enumerate(column) if not get_op(operator, x, value)]
```

Το συγκεκριμένο κομμάτι κώδικα είναι από το _select_where στην γραμμή 517. Σε κάθε σημείο της εργασίας είναι διαφορετικό για να καλύψει τις ανάγκες της εφαρμογής αλλά η λογική είναι η ίδια.
Η μόνη διαφορά που έγινε σε σχέση με τον αρχικό κώδικα είναι ότι στο _parse_condition δίνεται το condition χωρίς το not και στην τελευταία γραμμή υπάρχει το not στην συνθήκη και έτσι αντιστρέφεται το τι μπαίνει στο rows.

```
(smdb)> select * from student
  id (str) #PK#  name (str)    dept_name (str)      tot_cred (int)
--------------  ------------  -----------------  ----------------
         00128  zhang         comp. sci.                      102
         12345  shankar       comp. sci.                       32
         19991  brandt        history                          80
         23121  chavez        finance                         110
         44553  peltier       physics                          56
         45678  levy          physics                          46
         54321  williams      comp. sci.                       54
         55739  sanchez       music                            38
         70557  snow          physics                           0
         76543  brown         comp. sci.                       58
         76653  aoi           elec. eng.                       60
         98765  bourikas      elec. eng.                       98
         98988  tanaka        biology                         120
(smdb)> select * from student where not dept_name = history
  id (str) #PK#  name (str)    dept_name (str)      tot_cred (int)
--------------  ------------  -----------------  ----------------
         00128  zhang         comp. sci.                      102
         12345  shankar       comp. sci.                       32
         23121  chavez        finance                         110
         44553  peltier       physics                          56
         45678  levy          physics                          46
         54321  williams      comp. sci.                       54
         55739  sanchez       music                            38
         70557  snow          physics                           0
         76543  brown         comp. sci.                       58
         76653  aoi           elec. eng.                       60
         98765  bourikas      elec. eng.                       98
         98988  tanaka        biology                         120
```

Για την προσθήκη της λειτουργίας not between και between έγινε αλλαγή στο αρχείο table.py όπου στις συναρτήσεις _select_where, _select_where_with_btree, _select_where_with_hash, _update_rows και _delete_where προστέθηκε ο έλεγχος αν υπάρχει το not between ή between

στο condition. Στην περίπτωση που υπάρχει το not between τρέχει ο παρακάτω κώδικας:

```
column_name, values = self._parse_condition(condition.replace('not ', ''))
column = self.column_by_name(column_name)
rows = [ind for ind, x in enumerate(column) if not get_op_between(x, values[0], values[1])]
```

Το συγκεκριμένο κομμάτι κώδικα είναι από το _select_where στην γραμμή 509. Σε κάθε σημείο της εργασίας είναι διαφορετικό για να καλύψει τις ανάγκες της εφαρμογής αλλά η λογική είναι η ίδια. Στο _parse_condition δίνεται το condition χωρίς το not και στην τελευταία γραμμή υπάρχει το not στην συνθήκη και έτσι αντιστρέφεται το τι μπαίνει στο rows. Αυτή την φορά όμως χρησιμοποιείται η συνάρτηση get_op_between που ελέγχει αν η τιμή της γραμμής είναι μεταξύ των τιμών που έδωσε ο χρήστης. Εδώ είναι το not between άρα στο τέλος αντιστρέφεται η συνθήκη με το not.
Το between είναι το αντίστοιχο χωρίς την αντιστροφή της συνθήκης.
Επίσης έγιναν αλλαγές στο αρχείο misc.py όπου προστέθηκε η συνάρτηση split_condition_between που χωρίζει το condition στα values μεταξύ των οποίων θα γίνει η σύγκριση και στο column πάνω στο οποίο γίνεται η σύγκριση και η συνάρτηση get_op_between που κάνει την σύγκριση.

```
def split_condition_between(condition):
    splt = condition.split(' between ') # Split at between and get then get the right and left values around and and then get
    if len(splt) > 1:
        values = splt[1].split(' and ')[0:2]
        condition_column = splt[0]

        if values[0] == '"' == values[-1]:  # If the value has leading and trailing quotes, remove them.
            values = values.strip('"')
        elif ' ' in values:  # If it has whitespaces but no leading and trailing double quotes, throw.
            raise ValueError(
                f'Invalid condition: {condition}\nValue must be enclosed in double quotation marks to include whitespaces.')

        if values[0].find('"') != -1 or values[1].find(
                '"') != -1:  # If there are any double quotes in the value, throw. (Notice we've already removed the leading
            raise ValueError(f'Invalid condition: {condition}\nDouble quotation marks are not allowed inside values.')

        return condition_column, values
```

```python
def get_op_between(a, b, c):  # Funtion the takes checks if a is between b and c
    try:
        a = type(b)(a)  # Confirm that they are of the same type
        if b <= c: # Check if b and c are in the right order
            if b <= a <= c:
                return True
            else:
                return False
        else:
            if c <= a <= b:
                return True
            else:
                return False
    except TypeError:
        return False
```

#1 (b)

Για το and όπως και το or η διαδικασία είναι αρκετά πιο περίπλοκη. Αρχικά διαχωρίστηκαν οι συνθήκες που συνδέονται με and στην λίστα _and και τα or στην λίστα _or με τον παρακάτω κώδικα:

```
_or = [] # Same as above
_and = []
if condition is not None:
    if ' or ' in condition:
        _or = condition.split(' or ')
        for i in _or:
            if ' and ' in i:
                _and.append(i.split(' and '))
    elif ' and ' in condition:
        _and = [condition.split(' and ')]
    if ' and ' in condition:
        for i in range(len(_and)):
            end = len(_and[i])
            j = 0
            while j < end:
                if 'between ' in _and[i][j]:
                    _and[i][j:j + 2] = [' and '.join(_and[i][j:j + 2])]
                    end -= 1
                j += 1
```

Επίσης αν υπάρχει κάποιο between στο condition θα βρίσκεται και στην _and αφού το between χρησιμοποιεί την λέξη and.

Ο παρακάτω κώδικας κάνει την λειτουργία του and και του or που πάλι είναι διαφορετικός σε κάθε σημείο του κώδικα για να καλύπτει τις ανάγκες της κάθε εφαρμογής.
Η λογική για τις συνθήκες που συνδέονται με and είναι να πάρουμε το αποτέλεσμα της κάθε συνθήκης, να το προσθέσουμε με το αποτέλεσμα της επόμενης συνθήκης και να κρατάμε σε κάθε πρόσθεση τα κοινά χαρακτηριστικά.
Η λογική για τις συνθήκες που συνδέονται με or που μπορεί να είναι πολλές συνθήκες ενωμένες με and είναι η πρόσθεση του αποτελέσματος της δεξιάς και της αριστερής συνθήκης αφαιρώντας τα διπλότυπα.
Αν υπάρχουν and σε μια συνθήκη που συνδέεται με μία άλλη με το or πάντα γίνονται πρώτα τα and και μετά τα or.

```
rows = []
if _or and not _and:
    for i in _or:
        if 'not ' in i:
            column_name, operator, value = self._parse_condition(i.replace('not ', ''))
            column = self.column_by_name(column_name)
            index_rows = [ind for ind, x in enumerate(column) if not get_op(operator, x, value)]
        else:
            column_name, operator, value = self._parse_condition(i)
            column = self.column_by_name(column_name)
            index_rows = [ind for ind, x in enumerate(column) if get_op(operator, x, value)]
        rows += index_rows
    rows = list(set(rows))
elif _and:
    rows = []
    for j in _and:
        and_rows = []
        count = 0
        for i in j:
            if 'not between ' in i:
                column_name, values = self._parse_condition(i.replace('not ', ''))
                column = self.column_by_name(column_name)
                index_rows = [ind for ind, x in enumerate(column) if
                              not get_op_between(x, values[0], values[1])]
            elif 'between ' in i:
                column_name, values = self._parse_condition(i)
                column = self.column_by_name(column_name)
                index_rows = [ind for ind, x in enumerate(column) if
                              get_op_between(x, values[0], values[1])]
            elif 'not ' in i:
                column_name, operator, value = self._parse_condition(i.replace('not ', ''))
                column = self.column_by_name(column_name)
                index_rows = [ind for ind, x in enumerate(column) if not get_op(operator, x, value)]
            else:
                column_name, operator, value = self._parse_condition(i)
                column = self.column_by_name(column_name)
                index_rows = [ind for ind, x in enumerate(column) if get_op(operator, x, value)]
            if count == 0:
```

```
            if count == 0:
                and_rows = index_rows
            else:
                and_rows = set(and_rows).intersection(index_rows)
            count += 1
        rows += and_rows
    if _or:
        for i in _or:
            if ' and ' not in i or i.count(' and ') == i.count('between '):
                if 'not between ' in i:
                    column_name, values = self._parse_condition(i.replace('not ', ''))
                    column = self.column_by_name(column_name)
                    index_rows = [ind for ind, x in enumerate(column) if
                                  not get_op_between(x, values[0], values[1])]
                elif 'between ' in i:
                    column_name, values = self._parse_condition(i)
                    column = self.column_by_name(column_name)
                    index_rows = [ind for ind, x in enumerate(column) if
                                  get_op_between(x, values[0], values[1])]
                elif 'not ' in i:
                    column_name, operator, value = self._parse_condition(i.replace('not ', ''))
                    column = self.column_by_name(column_name)
                    index_rows = [ind for ind, x in enumerate(column) if not get_op(operator, x, value)]
                else:
                    column_name, operator, value = self._parse_condition(i)
                    column = self.column_by_name(column_name)
                    index_rows = [ind for ind, x in enumerate(column) if get_op(operator, x, value)]
                rows += index_rows
rows = list(set(rows))
```

Παρακάτω είναι κάποια παραδείγματα ερωτήσεων πάνω στον πίνακα student που φαίνεται η λειτουργία του and και του or.

```
(smdb)> select * from student where not dept_name = history and tot_cred = 58
  id (str) #PK#  name (str)    dept_name (str)      tot_cred (int)
---------------  ------------  -----------------  ----------------
          76543  brown         comp. sci.                       58
```

```
(smdb)> select * from student where not dept_name = history or tot_cred = 58
  id (str) #PK#  name (str)    dept_name (str)      tot_cred (int)
---------------  ------------  -----------------  ----------------
          00128  zhang         comp. sci.                      102
          12345  shankar       comp. sci.                       32
          23121  chavez        finance                         110
          44553  peltier       physics                          56
          45678  levy          physics                          46
          54321  williams      comp. sci.                       54
          55739  sanchez       music                            38
          70557  snow          physics                           0
          76543  brown         comp. sci.                       58
          76653  aoi           elec. eng.                       60
          98765  bourikas      elec. eng.                       98
          98988  tanaka        biology                         120
(smdb)> select * from student where dept_name = history and name = brandt or tot_cred = 58
  id (str) #PK#  name (str)    dept_name (str)      tot_cred (int)
---------------  ------------  -----------------  ----------------
          76543  brown         comp. sci.                       58
          19991  brandt        history                          80
```

#2 (a)

Για την προσθήκη της λειτουργίας btree indexes στα unique columns έπρεπε να προστεθούν τα unique columns.
Προστέθηκε η δήλωση των στηλών ως unique στο create table.
Πρώτα όμως στο mdb.py προστέθηκε στο dic το dic[‘unique’].

```
dic['unique'] = None
if 'unique' in args:  # Make a 'unique' item in dic if there are unique columns
    list = args.removesuffix(')').removeprefix('(').strip().split(',') # Remove parenthesis and split them by comma
    for i in list:
        if len(i.split(' ')) == 4:  # If there are enough items there can be a unique declaration
            if i.split(' ')[3] == 'unique':  # unique declaration needs to be last
                if dic['unique'] is None:  # If found add the columns in dic['unique']
                    dic['unique'] = i.split(' ')[1] + ','
                else:
                    dic['unique'] += i.split(' ')[1]
    dic['unique'] = dic['unique'].strip(',') # If there is only one unique column the last comma needs t be removed
```

Έπειτα στο database.py πλέον τα tables δημιουργούνται με ένα επιπλέον πεδίο unique.

```
if unique is not None:  # If there are unique columns make self.unique and make table given unique
    self.tables.update({name: Table(name=name, column_names=column_names.split(','), column_types=column_types.split(','), unique=unique.split(','), pr
    self.unique = unique.split(',')
else:  # Same as before where there is no unique columns
    self.tables.update({name: Table(name=name, column_names=column_names.split(','), column_types=column_types.split(','), unique=None, primary_key=pr
    self.unique = None
```

Επίσης στο table.py στην συνάρτηση _insert προστέθηκε έλεγχος έτσι ώστε όταν μπαίνει μία τιμή σε μία στήλη που είναι unique δεν θα πρέπει να

υπάρχει άλλη τέτοια τιμή στην στήλη.

```
if self.unique is not None:# Check that there are no duplicates in unique columns
    if i in self.unique_idx:
        for j in range(len(self.unique_idx)):# Checks if value is already in the column
            if i == self.unique_idx[j] and row[i] in self.column_by_name(self.unique[j]):
                raise ValueError(f'## ERROR -> Value {row[i]} already exists in unique column.')
```

Τα self.unique και self.unique_idx αρχικοποιούνται εδώ και χρησιμοποιούνται στους ελέγχους.

```
if unique is not None:# Get the unique columns
    self.unique = unique
    self.unique_idx = []
    for i in range(len(unique)):# Get the indexes of the unique columns
        self.unique_idx.append(column_names.index(unique[i]))
else:# If no unique columns
    self.unique = None
```

Για τα ευρετήρια το πρώτο πράγμα που έγινε ήταν να προστεθούν δύο ακόμη στήλες στο meta_indexes (index_type,column_name) στο αρχείο database.py για να διατηρούνται περισσότερες πληροφορίες για τα ευρετήρια. Στο mdb.py στην create_query_plan διαχωρίζεται το όνομα του πίνακα και η στήλη πάνω στην οποία θα δημιουργηθεί το ευρετήριο.

```
self.create_table('meta_indexes', 'table_name,index_name,index_type,column_name', 'str,str,str,str')
```

```
if action == 'create index':
    dic['column'] = dic['on'].split('(')[1].strip().removesuffix(')').strip() # The index column is specified in parenthesis
    dic['on'] = dic['on'].split('(')[0].strip()# The table where the index will be created
return dic
```

Έπειτα στο database.py στην συνάρτηση create_index γίνονται έλεγχοι ότι το ευρετήριο θα είναι πάνω σε πρωτεύον κλειδί ή unique πεδίο και ότι δεν υπάρχει άλλο ευρετήριο στον πίνακα και αν πληρούνται οι προϋποθέσεις δημιουργείται btree index με την construct_index ή αντίστοιχα hash index με την construct_hash_index.

Οι έλεγχοι που γίνονται μέσα στο database.py στην create_index (Η μεταβλητή table είναι το αντικείμενο table που δημιουργήθηκε όταν δημιουργήθηκε το table και αποθηκεύτηκε στο self.tables).

```
table = self.tables[table_name] # Get the table object of the table
if table.pk is not None: # Check if the column on which to create index is a primary key or unique
    if column_name not in table.pk:
        if table.unique is not None: # Check if the column is unique
            if column_name not in table.unique:
                raise ValueError(f'## ERROR -> Can\'t make index on non unique or non primary key field.')
        else: # There is no unique column
            raise ValueError(f'## ERROR -> Can\'t make index on non unique or non primary key field.')
elif table.unique is not None: # Check if the column is unique if there is no primary key
    if column_name not in table.unique:
        raise ValueError(f'## ERROR -> Can\'t make index on non unique or non primary key field.')
else: # If there is no primary key or unique column there can't be an index
    raise ValueError(f'## ERROR -> Can\'t make index on non unique or non primary key field.')
```

Η διαδικασία δημιουργίας btree index δεν αλλάζει στην _construct_index.
Η δημιουργία ενός πίνακα με unique πεδίο.

```
(smdb)> create table Teacher (id int primary key, age int unique, subject str)
Created table "teacher".
(smdb)> insert into Teacher values (2, 25, Math)
(smdb)> select * from Teacher
  id (int) #PK#    age (int)  subject (str)
---------------  -----------  ---------------
              1           25  history
(smdb)> insert into Teacher values (1, 25, History)
(smdb)> select * from Teacher
  id (int) #PK#    age (int)  subject (str)
---------------  -----------  ---------------
              1           25  history
```

Όπως φαίνεται επειδή η στήλη age είναι unique δεν δέχτηκε την τιμή 25 για age δεύτερη φορά.

Τέλος τροποποιήθηκε στο αρχείο table.py η συνάρτηση _select_where_with_btree για να μπορούν να υπάρχουν στο where τα not, between, and και or όπως και στο _select_where που τρέχει όταν ο πίνακας δεν έχει ευρετήρια. Η διαφορά είναι στο πως γίνονται τα select στο _select_where_with_btree. Για το not between γίνεται το εξής:

```
if 'not between ' in condition:
    column_name, values = self._parse_condition(condition.replace('not ', ''))
    column = self.column_by_name(column_name)
    if column_name == index_column: # If column name is the index column do btree searce
        if values[0] <= values[1]: # Check if left value is smaller than the right value
            rows = bt.find('<', values[0]) # Find values greater than the smallest value in values
            rows = list(set(rows).intersection(bt.find('>', values[1]))) # Add the values smaller th
        else:
            rows = bt.find('<', values[1]) # Find values greater than the smallest value in values
            rows = list(set(rows).intersection(bt.find('>', values[0]))) # Add the values smaller th
    else:
        rows = [ind for ind, x in enumerate(column) if not get_op_between(x, values[0], values[1])]
```

Στο παραπάνω τμήμα κώδικα ελέγχεται αν η στήλη της συνθήκης είναι πάνω στην στήλη που δημιουργήθηκε το index και αν είναι αληθής παίρνει τα rows μεγαλύτερα από την μικρότερη τιμή και τα προστέθει στα rows που είναι μικρότερα από την μικρότερη τιμή χωρίς τα διπλότυπα. Επίσης ελέγχεται ποια τιμή είναι η μεγαλύτερη και ποια είναι η μικρότερη και αν είναι ίσες ώστε να είναι σωστό το διάστημα στο οποίο θα ψάξουμε με το bt.find.

Για το between γίνεται το το παρακάτω με την διαφορά ότι το διάστημα είναι αντιστραμένο.

```
elif 'between ' in condition:
    column_name, values = self._parse_condition(condition)
    column = self.column_by_name(column_name)
    if column_name == index_column:
        if values[0] <= values[1]:  # Check if left value is smaller than the right value
            rows = bt.find('>=', values[0]) # Find values smaller than or equal to the biggest v
            rows = list(set(rows).intersection(bt.find('<=', values[1])))
        else:
            rows = bt.find('>=', values[1])
            rows = list(set(rows).intersection(bt.find('<=', values[0])))
    else:
        rows = [ind for ind, x in enumerate(column) if get_op_between(x, values[0], values[1])]
```

Στο παρακάτω γίνεται το not όπου ελέγχεται το operator και το αντιστρέφει στο bt.find.

```
elif 'not ' in condition:
    column_name, operator, value = self._parse_condition(condition.replace('not ', ''))
    column = self.column_by_name(column_name)
    if column_name == index_column:
        if operator == '>':
            rows = bt.find('<=', value)
        elif operator == '<':
            rows = bt.find('>=', value)
        elif operator == '>=':
            rows = bt.find('<', value)
        elif operator == '<=':
            rows = bt.find('>', value)
        else:
            rows = bt.find('<', value)
            rows += bt.find('>', value)
            rows = list(set(rows))
    else:
        rows = [ind for ind, x in enumerate(column) if not get_op(operator, x, value)]
```

Στο παρακάτω γίνεται η συνθήκη χωρίς το not between, between ή not όπου γίνεται απλά το bt.find.

```
else:
    column_name, operator, value = self._parse_condition(condition)
    column = self.column_by_name(column_name)
    if column_name == index_column:
        rows = bt.find(operator, value)
    else:
        rows = [ind for ind, x in enumerate(column) if get_op(operator, x, value)]
```

Τέλος σε κάθε ένα από αυτά αν η στήλη πάνω στην οποία είναι η συνθήκη δεν είναι η στήλη πάνω στην οποία είναι το index γίνεται το ίδιο με το select χωρίς index.

#2 (b)

Για την προσθήκη της λειτουργίας hash index χρειάστηκε παραπάνω δουλειά από ότι για τα btree indexes.
Μαζί με τις αλλαγές που έγιναν για την υποστήριξη btree index πάνω σε unique columns προστέθηκε η συνάρτηση _construct_hash_index στην οποία γίνεται η παραγωγή του ευρετηρίου με την χρήση μιας hash function που βρίσκει το υπόλοιπο της διαίρεσης του αθροίσματος των ord() τιμών κάθε χαρακτήρα της τιμής της εγγραφής με τον αριθμό των εγγραφών του πίνακα.

Έπειτα προσθέτει στο dictionary hash_map με κλειδί το παραπάνω αποτέλεσμα το index και την τιμή της εγγραφής.

```
def _construct_hash_index(self, table_name, index_name, column_name, index_type=None):
    column_length = len(self.tables[table_name].data)
    hash_map = {}
    hash_map['Rows Length'] = len(self.tables[table_name].data)# Add the number of the table columns
    for index, value in enumerate(self.tables[table_name].column_by_name(column_name)):
        if value is None:# If the value is None don't make hash index for that value
            continue
        sum = 0# The hash sum
        value = str(value)# Make the column item a string so that it is iterable
        for i in value:# Add the ord() sum of each character
            sum += ord(i)
        hash = sum % column_length# Get the modulo value with the amount of columns

        if hash not in hash_map:# If there is no hash item in hash_map with the same hash make new item in hash_map with the index
            hash_map[hash] = [[index, value]]
        else:# If there is a hash item with the same hash add the item to the hash_map with the same hash
            hash_map[hash].append([index, value])
    self._save_index(index_name, hash_map)# Save the hash_map
```

Επίσης προστέθηκε στο αρχείο table.py η συνάρτηση _select_with_hash για να μπορεί να γίνει το select με hash όταν ο πίνακας έχει hash index.
Το συγκεκριμένο select είναι παρόμοιο με αυτό που υπάρχει και για το select χωρίς ευρετήριο με την διαφορά ότι όταν υπάρχει ερώτηση ταυτότητας πάνω στην στήλη του index τρέχει ο παρακάτω κώδικας ή κάποια έκδοση το παρακάτω κώδικα.

```
sum = 0 # Hash find sums the chara
value = str(value) # Make the valu
for key in value:
    sum += ord(key)
index = sum % hash['Rows Length']
for keys in hash[index]: # if the
    rows.append(keys[0])
```

Στο συγκεκριμένο κομμάτι κώδικα βρίσκουμε την hash τιμή της εγγραφής και αποθηκεύουμε στην λίστα rows τις τιμές του dictionary hash με κλειδί την hash τιμή.

```
(smdb)> select * from advisor
  s_id (str) #PK#    i_id (str)
-----------------  ------------
            00128         45565
            12345         10101
            23121         76543
            44553         22222
            45678         22222
            76543         45565
            76653         98345
            98765         98345
            98988         76766
```

Δημιουργία hash index πάνω στο πρωτεύον κλειδί.

```
(smdb)> create index hash_index on advisor (s_id) using hash
(smdb)> select * from meta_indexes
table_name (str)    index_name (str)    index_type (str)    column_name (str)
-----------------  -----------------  -----------------  ------------------
advisor            hash_index         s_id               hash
```

Select πάνω στο πρωτεύον κλειδί.

```
(smdb)> select * from advisor where s_id = 00128 or s_id = 12345
  s_id (str) #PK#    i_id (str)
-----------------  ------------
            00128         45565
            98988         76766
```