Πανεπιστήμιο Πειραιώς

Τμήμα Πληροφορικής

Μάθημα Προπτυχιακών Σπουδών:

Συστήματα Διαχείρισης Βάσεων Δεδομένων

Ακαδημαϊκό Έτος: 2022 – 2023

Εξάμηνο: 5ο

Τελική Εργασία

<u>Ομάδα Εργασίας:</u>

Θοδωρής Κοξάνογλου Π20094

Αποστόλης Σιαμπάνης Π20173

Περιεχόμενα

Issues

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

Currently, miniDB does not support the use of (a) the simple NOT and BETWEEN operators as well as (b) complex where conditions combined with AND or OR operators. The target of this issue is to add this functionality, both at mSQL level and internally, i.e., in table.py and database.py.

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

Currently, miniDB supports BTree indexing over the primary key of a table. The target of this issue is to (a) add support for BTree index over other columns as well, if they are declared as 'unique', and (b) add support for Hash index over the PK or a unique column of a table (implement extendible hashing, either MSB or LSB variant, by using a hash function based on the modulo operator "%" – dict is a hash struct in python that can be used in your implementation). The required changes affect the SQL parser mdb.py as well as the maintenance of the meta_table "meta_indexes". In either case, as a preparatory action, the CREATE TABLE statement should be enriched to support the declaration of a column as 'unique'.

#3 - **Implement miniDB's query optimiser** by **(a)** building equivalent query plans based on respective RA expressions (**10/50**) and **(b)** choosing the optimal query execution plan using a cost-based evaluation (**20/50**)

Currently, miniDB generates a single query plan (dictionary) that corresponds to the 'obvious' translation of a user's query. The target of this issue is to support cost-based query optimization. In detail: (a) build a module that is responsible for creating multiple equivalent query plans using relational algebraic equivalence transformation rules and (b) implement a simple query evaluator that uses per column statistics (number of distinct values, min-max values, histograms, etc.) to evaluate the cost of each query plan; the best plan should then be fed into miniDB for execution.

Important note: the query optimizer should, on the one hand, maintain statistics metadata and, on the other hand, make **the least necessary** changes to the internal miniDB files (table.py and database.py). Use PostgreSQL and/or SQLite as a frame of reference when necessary.

Issue #1

Προσθέσαμε την δυνατότητα επεξεργασίας των conditions στα actions: select, update_table, delete_from της μεθόδου *create_query_plan*

(a) Simple NOT and BETWEEN operators

<u>Στο αρχείο mdb.py:</u>

Παρόμοια με την μέθοδο *evaluate_from_clause*, δημιουργήσαμε την *evaluate_where_clause* για την σωστή διαχείριση των conditions.

```
def evaluate_where_clause(dic):
    '''
    Evaluate the part of the query that is supplied as the 'where' argument
    '''
    if dic['where'] is None:
        return dic
    where_split = split_statement(dic['where'])
    #find the indices of the not and between keywords
    not_idx = [i for i,word in enumerate(where_split) if word=='not' and not in_paren(where_split,i)]
    between_idx = [i for i,word in enumerate(where_split) if word=='between' and not in_paren(where_split,i)]
    not_between_idx = [i for i,word in enumerate(where_split) if word=='not' and where_split[i+1]=='between' and not in_paren(where_split,i)]

    operators = {'>=': '<',
                 '<=': '>',
                 '!=': '=',
                 '>': '<=',
                 '<': '>=',
                 '=': '!=',
                 }
```

Εικόνα 1: def evaluate_where_clause

Στην περίπτωση που έχουμε condition με **‘ not ‘**, μετατρέπουμε τον operator του condition στον αντίθετό του και αφαιρούμε το **‘ not ‘** από το where statement. Αυτή η μετατροπή μπορεί να συμβεί και για πολλαπλά **‘ not ‘** στο where statement.

```
# for every not keyword, delete not and change the operator to the opposite one
while not_idx:
    not_idx = not_idx[0]
    condition_right = ' '.join(where_split[not_idx+1:not_idx+4])
    where_split_right = ' '.join(where_split[not_idx+4:])
    where_split_left = ' '.join(where_split[:not_idx])

    for key, value in operators.items():
        if key in condition_right:
            dic['where'] = where_split_left + ' ' + condition_right.replace(key, value) + ' ' + where_split_right
            break
    where_split = split_statement(dic['where'])
    not_idx = [i for i,word in enumerate(where_split) if word=='not' and not in_paren(where_split,i)]
```

Εικόνα 2: def evaluate_where_clause handles NOT

Όταν έχουμε condition με **‘ between ‘** μετατρέπουμε το condition από “column **BETWEEN** value1 **AND** value2” σε “column **>=** value1 **AND** column **<=** value2”. Αυτή η μετατροπή μπορεί να συμβεί και για πολλαπλά **‘ between ’** στο where statement.

```
# for every between keyword, delete between and change it to >= and <=
while between_idx:
    between_idx = between_idx[0]
    column_name = where_split[between_idx-1]
    value1= where_split[between_idx+1]
    value2= where_split[between_idx+3]

    where_split_right = ' '.join(where_split[between_idx+4:])
    where_split_left = ' '.join(where_split[:between_idx-1])

    dic['where'] = where_split_left + ' ' + column_name + " >= " + value1 + " and " + column_name + " <= " + value2 + ' ' + where_split_right

    where_split = split_statement(dic['where'])
    between_idx = [i for i,word in enumerate(where_split) if word=='between' and not in_paren(where_split,i)]

return dic
```

Εικόνα 3: def evaluate_where_clause handles BETWEEN

Επιπλέον, έχουμε προσθέσει έλεγχο καθώς γίνονται οι μετατροπές των conditions να μην κόβονται τα values των columns της μορφής **“word word”**, δηλαδή values που περιέχουν white spaces – είναι η μέθοδος *split_statement*.

```
def split_statement(statement):
    '''
    Split a statement into a list of words, but keep the words in quotes together.
    '''
    result = []
    current = ""
    in_quote = False
    for char in statement:
        if char == "\"":
            in_quote = not in_quote
            current += char
        elif char == " " and not in_quote:
            result.append(current)
            current = ""
        else:
            current += char
    result.append(current)
    return result
```

Εικόνα 4: def split_statement

Στο αρχείο misc.py:

Προσθέσαμε τον operator **‘!=’** – **not equal** – ώστε να διαχειριστούμε την περίπτωσή που ο χρήστης χρησιμοποιήσει conditions της μορφής: **“column != value”**.

```
ops = {'>=': operator.ge,
       '<=': operator.le,
       '!=': operator.ne,
       '=': operator.eq,
       '>': operator.gt,
       '<': operator.lt
       }
```

Εικόνα 5: misc.py, added not equal operator '!='

Παραδείγματα Εκτέλεσης Κώδικα

```
(smdb)> 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
```

*Εικόνα 6:Select * from instructor*

```
(smdb)> select * from instructor where not salary > 50000
  id (str) #PK#  name (str)    dept_name (str)      salary (int)
---------------  ------------  -----------------  --------------
          15151  mozart        music                       40000
```

*Εικόνα 7: Select * from instructor where not salary > 50000*

```
(smdb)> select * from instructor where salary between 50000 and 70000
  id (str) #PK#  name (str)    dept_name (str)      salary (int)
---------------  ------------  -----------------  --------------
          10101  srinivasan    comp. sci.                  65000
          32343  el said       history                     60000
          58583  califieri     history                     62000
```

*Εικόνα 8: select * from instructor where salary between 50000 and 70000*

(b) Complex where conditions combined with AND or OR operators

Στο αρχείο table.py:

Στις μεθόδους: *_update_rows, delete_where, _select_where* αντικαταστήσαμε την προηγούμενη υλοποίηση του ενός condition με μία νέα μέθοδο, την *_parse_multiple_conditions*, που επιστρέφει απευθείας τα rows του table που ζητούνται να ανακτηθούν από τον χρήστη. Η μέθοδος αυτή επεξεργάζεται το string της **dic['where']**. Αρχικά, αναγνωρίζει τους **' or '** operators στο string και προσθέτει τα conditions που βρίσκονται αριστερά και δεξιά των operators αυτών σε μια προσωρινή λίστα *sub_conditions_rows_list*. Όταν τελειώσει ο διαχωρισμός των sub-conditions εκτελείται η μέθοδος *get_rows_for_and* για την αναγνώριση των **' and '** operators.

```python
def _parse_multiple_conditions(self, conditions):
    '''
    Get the rows that satisfy "where" statement for "or" and "and" operators.

    Args:
        conditions_list_of_or: list of strings.
        rows_for_or: set of integers.
        sub_conditions_rows_list: list of sets of integers.
    '''
    conditions_temp = conditions
    # check if there is any or operator
    if ' or ' in conditions_temp:
        conditions_list_of_or = conditions_temp.split(' or ')
        rows_for_or = set(range(0))
        sub_conditions_rows_list = []

        for sub_conditions in conditions_list_of_or:
            sub_conditions_rows_list.append(self.get_rows_for_and(sub_conditions.split(' and ')))
        # get the rows that satisfy the or conditions
        for rows in sub_conditions_rows_list:
            rows_for_or = rows_for_or.union(rows)

        return rows_for_or
    else:
        return self.get_rows_for_and(conditions_temp.split(' and '))
```

Εικόνα 9: def _parse_multiple_conditions

Η *get_rows_for_and* επεξεργάζεται τα sub-conditions της *_parse_multiple_condition* και υπολογίζει τα rows που επαληθεύουν κάθε condition ξεχωριστά. Όταν τελειώσει η αναζήτηση όλων των sub-conditions, προστίθενται τα κοινά rows σε μία νέα λίστα, η οποία επιστρέφεται πίσω στη parse_multiple_conditions για την τελική επεξεργασία των rows που επιστρέφει η βάση στο χρήστη.

```python
def get_rows_for_and(self, conditions_list_of_and):
    '''
    Get the rows that satisfy "and" conditions in the list of conditions and then return them.

    Args:
        conditions_columns_of_and: list of strings.
        splitted_conditions_list_of_and: list of strings.
        rows_for_and: set of integers.
    '''
    conditions_columns_of_and = []
    splitted_conditions_list_of_and = []
    rows_for_and = set(range(len(self.data)))
    # get the columns and the conditions for each and condition
    for sub_conditions in conditions_list_of_and:
        conditions_columns_of_and.append(split_condition(sub_conditions)[0])
        splitted_conditions_list_of_and.append(self._parse_condition(sub_conditions))
    # get the rows that satisfy the and conditions
    for index in range(len(conditions_list_of_and)):
        column = self.column_by_name(conditions_columns_of_and[index])
        rows_for_and = rows_for_and.intersection([ind for ind, x in enumerate(column) if get_op(splitted_conditions_list_of_and[index][1], x, splitted_conditions_list_of_and[index][2])])

    return rows_for_and
```

Εικόνα 10: def get_rows_for_and

Παραδείγματα Εκτέλεσης Κώδικα

```
(smdb)> 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
```

*Εικόνα 11: Select * from instructor*

```
(smdb)> select * from instructor where salary > 72000 and id <= 50000
  id (str) #PK#  name (str)    dept_name (str)      salary (int)
---------------  ------------  -----------------  --------------
          12121  wu            finance                     90000
          22222  einstein      physics                     95000
          33456  gold          physics                     87000
          45565  katz          comp. sci.                  75000
```

*Εικόνα 12:Select * from instructor where salary > 72000 and id <= 50000*

```
(smdb)> select * from instructor where salary > 72000 or id <= 50000
  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
          76543  singh         finance                     80000
          83821  brandt        comp. sci.                  92000
          98345  kim           elec. eng.                  80000
```

*Εικόνα 13:Select * from instructor where salary > 72000 or id <= 50000*

```
(smdb)> select * from instructor where salary < 72000 and id between 10101 and 32343 or id = 98345
  id (str) #PK#  name (str)    dept_name (str)      salary (int)
---------------  ------------  -----------------  --------------
          10101  srinivasan    comp. sci.                  65000
          15151  mozart        music                       40000
          98345  kim           elec. eng.                  80000
          32343  el said       history                     60000
```

*Εικόνα 14: Select * from instructor where salary > 72000 and id between 10101 and 32343 or id = 98345*

Issue #2

(a) BTree index over unique (non-PK) columns

Στο αρχείο mdb.py:

Στην μέθοδο create_query_plan προσθέσαμε δύο επιπλέον ελέγχους για τα **action**: **“create table”** και **“create index”** ώστε να μπορεί η βάση να διαχειρίζεται τα **unique columns**. Η λογική που ακολουθήσαμε είναι παρόμοια με αυτή που χρησιμοποιήθηκε για το **primary key**.

```
if action=='create table':
    args = dic['create table'][dic['create table'].index('('):dic['create table'].index(')')+1]
    dic['create table'] = dic['create table'].removesuffix(args).strip()

    arg_nopk_nounique = args.replace('primary key', '').replace('unique', '')[1:-1]
    arglist = [val.strip().split(' ') for val in arg_nopk_nounique.split(',')]
    dic['column_names'] = ','.join([val[0] for val in arglist])
    dic['column_types'] = ','.join([val[1] for val in arglist])
    if 'primary key' in args:
        arglist = args[1:-1].split(' ')
        dic['primary key'] = arglist[arglist.index('primary')-2]
    else:
        dic['primary key'] = None

    if 'unique' in args:
        unique_list = [i for i, x in enumerate(arglist) if x=='unique' or x=='unique,']
        dic['unique'] = ','.join(arglist[i-2] for i in unique_list)
    else:
        dic['unique'] = None

if action=='create index':
    args = dic['on'].split(' ', 1)
    if len(args) == 2:
        dic['on'] = args[0]
        dic['column'] = args[1].strip().replace('(','').replace(')','').replace(' ','')
    else:
        dic['column'] = None
```

Εικόνα 15: def create_query_plan

Στο αρχείο database.py:

Προσθέσαμε στις μεθόδους *create_table* και *import table* το attribute **unique** ώστε να μπορέσει να προστεθεί το αναγνωριστικό αυτό στις σωστές στήλες του πίνακα.

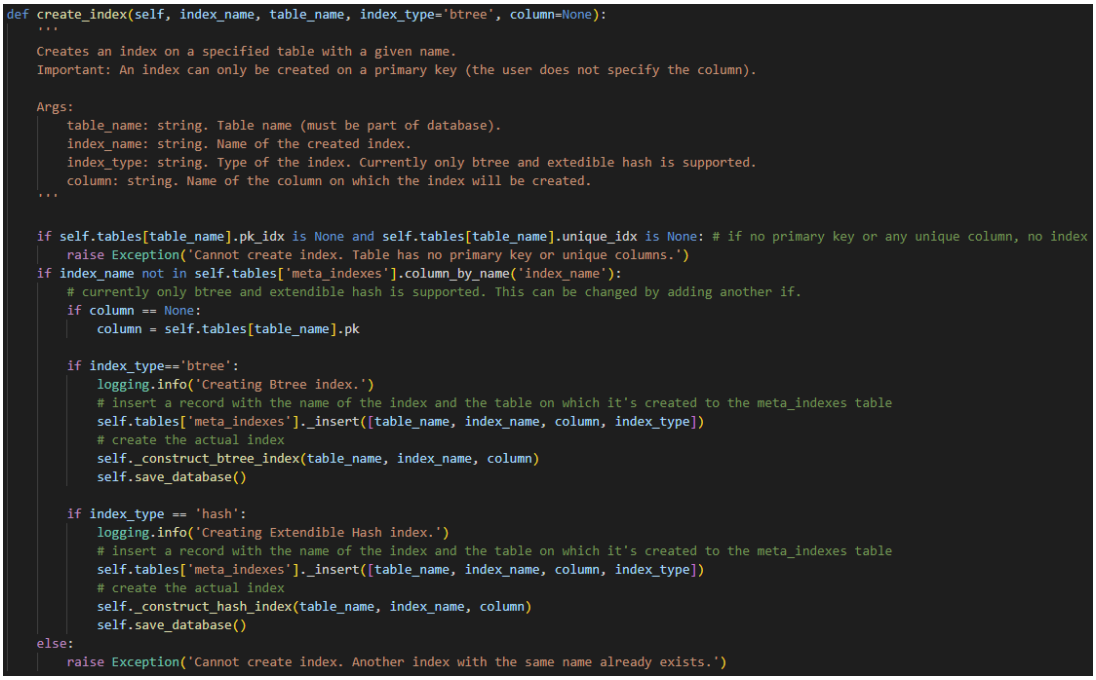
```
def import_table(self, table_name, filename, column_types=None, primary_key=None, unique=None):
    '''
    Creates table from CSV file.

    Args:
        filename: string. CSV filename. If not specified, filename's name will be used.
        column_types: list. Types of columns. If not specified, all will be set to type str.
        primary_key: string. The primary key (if it exists).
        unique: list. List of unique columns (if it exists).
    '''
    file = open(filename, 'r')

    first_line=True
    for line in file.readlines():
        if first_line:
            colnames = line.strip('\n')
            if column_types is None:
                column_types = ",".join(['str' for _ in colnames.split(',')])
            self.create_table(name=table_name, column_names=colnames, column_types=column_types, primary_key=primary_key, unique=unique)
            lock_ownership = self.lock_table(table_name, mode='x')
            first_line = False
            continue
        self.tables[table_name]._insert(line.strip('\n').split(','))

    if lock_ownership:
        self.unlock_table(table_name)
    self._update()
    self.save_database()
```

Εικόνα 16: def import_table

```
def create_table(self, name, column_names, column_types, primary_key=None, unique=None, load=None):
    '''
    This method create a new table. This table is saved and can be accessed via db_object.tables['table_name'] or db_object.table_name

    Args:
        name: string. Name of table.
        column_names: list. Names of columns.
        column_types: list. Types of columns.
        primary_key: string. The primary key (if it exists).
        unique: list. List of unique columns (if it exists).
        load: boolean. Defines table object parameters as the name of the table and the column names.
    '''
    # print('here -> ', column_names.split(','))
    self.tables.update({name: Table(name=name, column_names=column_names.split(','), column_types=column_types.split(','), primary_key=primary_key, unique=unique, load=load)})
    # self._name = Table(name=name, column_names=column_names, column_types=column_types, load=load)
    # check that new dynamic var doesnt exist already
    # self.no_of_tables += 1
    self._update()
    self.save_database()
    # (self.tables[name])
    if self.verbose:
        print(f'Created table "{name}".')
```

Εικόνα 17: def create_table

Έχουμε προσθέσει ως argument στη μέθοδο *create_index* το column στο οποίο ζήτησε ο χρήστης να δημιουργηθεί το **index**. Το column μπορεί να είναι είτε **Primary key** είτε **unique**.



```
def create_index(self, index_name, table_name, index_type='btree', column=None):
    '''
    Creates an index on a specified table with a given name.
    Important: An index can only be created on a primary key (the user does not specify the column).

    Args:
        table_name: string. Table name (must be part of database).
        index_name: string. Name of the created index.
        index_type: string. Type of the index. Currently only btree and extedible hash is supported.
        column: string. Name of the column on which the index will be created.
    '''

    if self.tables[table_name].pk_idx is None and self.tables[table_name].unique_idx is None: # if no primary key or any unique column, no index
        raise Exception('Cannot create index. Table has no primary key or unique columns.')
    if index_name not in self.tables['meta_indexes'].column_by_name('index_name'):
        # currently only btree and extendible hash is supported. This can be changed by adding another if.
        if column == None:
            column = self.tables[table_name].pk

        if index_type=='btree':
            logging.info('Creating Btree index.')
            # insert a record with the name of the index and the table on which it's created to the meta_indexes table
            self.tables['meta_indexes']._insert([table_name, index_name, column, index_type])
            # create the actual index
            self._construct_btree_index(table_name, index_name, column)
            self.save_database()

        if index_type == 'hash':
            logging.info('Creating Extendible Hash index.')
            # insert a record with the name of the index and the table on which it's created to the meta_indexes table
            self.tables['meta_indexes']._insert([table_name, index_name, column, index_type])
            # create the actual index
            self._construct_hash_index(table_name, index_name, column)
            self.save_database()
    else:
        raise Exception('Cannot create index. Another index with the same name already exists.')
```

Εικόνα 18: def create_index

<u>Στο αρχείο table.py:</u>

Έχουμε προσθέσει το argument **unique** στη μέθοδο *_insert* για την σωστή αποθήκευση και ανάγνωση των πεδίων του πίνακα. Επιπλέον, προσθέσαμε την δυνατότητα **να μην αποθηκεύονται** τα διπλότυπα δεδομένα στη βάση.

```python
def _insert(self, row, insert_stack=[]):
    '''
    Insert row to table.

    Args:
        row: list. A list of values to be inserted (will be casted to a predifined type automatically).
        insert_stack: list. The insert stack (empty by default).
    '''
    try:
        if len(row)!=len(self.column_names):
            raise ValueError(f'## ERROR -> Cannot insert {len(row)} values. Only {len(self.column_names)} columns exist')

        for i in range(len(row)):
            # for each value, cast and replace it in row.
            try:
                row[i] = self.column_types[i](row[i])
            except ValueError:
                if row[i] != 'NULL':
                    raise ValueError(f'## ERROR -> Value {row[i]} of type {type(row[i])} is not of type {self.column_types[i]}.')
            except TypeError as exc:
                if row[i] != None:
                    print(exc)

            # if value is to be appended to the primary_key column, check that it doesnt already exist (no duplicate primary keys)
            if i==self.pk_idx and row[i] in self.column_by_name(self.pk):
                raise ValueError(f'## ERROR -> Value {row[i]} already exists in primary key column.')
            elif i==self.pk_idx and row[i] is None:
                raise ValueError(f'## ERROR -> The value of the primary key cannot be None.')

            # if value is to be appended to a unique column, check that it doesnt already exist (no duplicate unique values)
            if self.unique is not None:
                for j in range(len(self.unique)):
                    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 {self.unique[j]} column.')
                    elif i==self.unique_idx[j] and row[i] is None:
                        raise ValueError(f'## ERROR -> The value of the unique column {self.unique[j]} cannot be None.')
    except ValueError as exc:
        print(exc)
        return

    # if insert_stack is not empty, append to its last index
    if insert_stack != []:
        self.data[insert_stack[-1]] = row
    else: # else append to the end
        self.data.append(row)
    # self._update()
```

Εικόνα 19: def _insert

Παραδείγματα Εκτέλεσης Κώδικα

Σημείωση: Για την προβολή των παραδειγμάτων αλλάξαμε προσωρινά από την ***smallRelationsInsertFile.sql*** δύο SQL query.

Από:

- create table department (dept_name str primary key, building str, budget int);
- create table course (course_id str primary key, title str, dept_name str, credits int);

Σε:

- create table department (dept_name str primary key, building str, budget int unique);
- create table course (course_id str primary key, title str, dept_name str unique, credits int);

```
(smdb)> explain create table course (course_id str primary key, title str, dept_name str unique, credits int);
{'create table': 'course',
 'column_names': 'course_id,title,dept_name,credits',
 'column_types': 'str,str,str,int',
 'primary key': 'course_id',
 'unique': 'dept_name'}
```

*Εικόνα 20: explain create table course (course_id str primary key, title str, dept_name str **unique**, credits int);*

```
(smdb)> explain create index dept_name on course (dept_name) using hash
{'create index': 'dept_name',
 'on': 'course',
 'using': 'hash',
 'column': 'dept name'}
```

Εικόνα 21: explain create index dept_name on course (dept_name) using hash;

```
(smdb)> select * from meta_indexes
table_name (str)    index_name (str)    column_name (str)    index_type (str)
-----------------   -----------------   ------------------   -----------------
course              dept_name           dept_name            hash
department          department          budget               btree
```

*Εικόνα 22: select * from meta_indexes (μπορούν να δημιουργηθούν και indexes πάνω σε υποψήφια κλειδιά - unique)*

(b) Hash index over PK or unique columns

Στο αρχείο mdb.py:

Στην μέθοδο *create_query_plan* προσθέσαμε δύο ελέγχους για τα action **"create table"** και **"create index"** ώστε να μπορούμε να επεξεργαστούμε το τελικό **dictionary** του SQL query.

Το στιγμιότυπο θα το βρείτε εδώ.

Στο αρχείο database.py:

Στον πίνακα **meta_indexes** προστέθηκαν οι στήλες **column_name** (για να αποθηκεύουμε το όνομα της στήλης που χρησιμοποιήθηκε για το index) και το **index_type** (για να αποθηκεύουμε τον τύπο του index).

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

Εικόνα 23: database.py

Έχουμε προσθέσει έλεγχο για τα conditions πριν εκτελέσουμε την κατάλληλη αναζήτηση. Ελέγχουμε αν υπάρχουν conditions με columns που περιλαμβάνουν ευρετήριο, καθώς και τι operator έχουν τα conditions. Αν δεν έχουμε columns που να περιλαμβάνουν ευρετήριο, τότε εκτελείται αναζήτηση σωρού πάνω σε ερώτηση ταυτότητας ή διαστήματος.

```
#select correct method select_where depending on the number of conditions, their operators and the existance of indexes on their column
if len(condition_list) == 0:
    table = self.tables[table_name]._select_where(columns, condition, distinct, order_by, desc, limit)
else:
    if self._has_index(table_name) and '!=' not in condition_list[0] and (conditions_columns[0]==self.tables[table_name].column_names[self.tables[table_name].pk_idx] \
                                                                          or (self.tables[table_name].unique is not None and conditions_columns[0] in self.tables[table_name].unique)):
        index_name,index_type = self.select('index_name,index_type', 'meta_indexes', f'table_name={table_name} and column_name={conditions_columns[0]}', return_object=True).data[0]
        if index_name is not None and index_type == 'btree':
            bt = self._load_idx(index_name)
            table = self.tables[table_name]._select_where_with_btree(columns, bt, condition, distinct, order_by, desc, limit)
        elif index_name is not None and '=' in condition_list[0] and index_type == 'hash':
            ht = self._load_idx(index_name)
            table = self.tables[table_name]._select_where_with_hash(columns, ht, condition, distinct, order_by, desc, limit)
        else:
            table = self.tables[table_name]._select_where(columns, condition, distinct, order_by, desc, limit)
    else:
        table = self.tables[table_name]._select_where(columns, condition, distinct, order_by, desc, limit)
```

Εικόνα 24: def select

Στην *create_index* προστέθηκε η δυνατότητα δημιουργίας hash index ανάλογα με το SQL query του χρήστη.

Το στιγμιότυπο θα το βρείτε εδώ.

Η μέθοδο *_construct_hash_index* δημιουργεί το Extensible hash index του πίνακα πάνω στην στήλη που επέλεξε ο χρήστης. (Προϋπόθεση: η στήλη να είναι είτε **Primary key** είτε **unique**)

```python
def _construct_hash_index(self, table_name, index_name, column_name=None):
    '''
    Construct a extendible hash on a table and save.

    Args:
        table_name: string. Table name (must be part of database).
        index_name: string. Name of the created index.
        column_name: string. Name of the column on which the index will be created.
    '''
    ht = Hash(4) # 4 is the bucket size

    if column_name is None:
        column_name = self.tables[table_name].pk

    if column_name == self.tables[table_name].pk or column_name in self.tables[table_name].unique:
        # for each record in the primary key of the table, insert its value and index to the extendible hash
        for idx, key in enumerate(self.tables[table_name].column_by_name(column_name)):
            if key is None:
                continue
            record = (key, idx)
            ht.insert(record)
    # save the extendible hash
    self._save_index(index_name, ht, column_name)
```

Εικόνα 25: def _construct_hash_index

Επιπλέον, αλλάξαμε την ονομασία του *_construct_index* σε *_construct_btree_index*, ώστε η ονομασία της μεθόδου να αντικατοπτρίζει τις λειτουργίες της μεθόδου.

```python
def _construct_btree_index(self, table_name, index_name, column_name=None):
    '''
    Construct a btree on a table and save.

    Args:
        table_name: string. Table name (must be part of database).
        index_name: string. Name of the created index.
        column_name: string. Name of the column on which the index will be created.
    '''
    bt = Btree(3) # 3 is arbitrary

    if column_name is None:
        column_name = self.tables[table_name].pk

    if column_name == self.tables[table_name].pk or (self.tables[table_name].unique is not None and column_name in self.tables[table_name].unique):
        # for each record in the primary key of the table, insert its value and index to the btree
        for idx, key in enumerate(self.tables[table_name].column_by_name(column_name)):
            if key is None:
                continue
            bt.insert(key, idx)
    # save the btree
    self._save_index(index_name, bt, column_name)
```

Εικόνα 26: def _construct_btree_index

Στην *_save_index* έχουμε προσθέσει το argument **column_name** για την σωστή αποθήκευση του πίνακα στο column name.

```python
def _save_index(self, index_name, index, column_name):
    '''
    Save the index object.

    Args:
        index_name: string. Name of the created index.
        index: obj. The actual index object (btree object).
        column_name: string. Name of the column on which the index will be created.
    '''
    try:
        os.mkdir(f'{self.savedir}/indexes')
    except:
        pass

    index_info = {'index': index, 'column_name': column_name}
    with open(f'{self.savedir}/indexes/meta_{index_name}_index.pkl', 'wb') as f:
        pickle.dump(index_info, f)
```

Εικόνα 27: def _save_index

Προσθέσαμε επιπλέον ελέγχους που εκτυπώνουν σωστά τις στήλες των πινάκων που είναι **Primary key** ή **Unique**. (Αναλυτικά εδώ)

Στη *_select_where_with_hash* ελέγχουμε αν οι στήλες που επιλέχθηκαν έχουν hash index. Αν υπάρχει ευρετήριο, εκτελείται η μέθοδο *find* της class **Hash** ώστε ο βρούμε την πλειάδα που αναζητεί ο χρήστης.

```python
def _select_where_with_hash(self, return_columns, ht, condition, distinct=False, order_by=None, desc=True, limit=None):
    '''
    Returns the table of a select where condition is met, using an extendible hash.
    '''
    # if * return all columns, else find the column indexes for the columns specified
    if return_columns == '*':
        return_cols = [i for i in range(len(self.column_names))]
    else:
        return_cols = [self.column_names.index(colname) for colname in return_columns]

    column_name, operator, value = self._parse_condition(condition)

    # if the column in condition is not a primary key and unique, abort the select
    if self.unique is not None and column_name != self.column_names[self.pk_idx] and column_name not in self.unique:
        print('Column is not PK or unique. Aborting')
    elif self.unique is None and column_name != self.column_names[self.pk_idx]:
        print('Column is not PK. Aborting')

    # hash find
    if operator == '=':
        rows = [ht.find(value)]
    else:
        raise ValueError('Hash table only supports equality operator')

    dict = {(key):([[self.data[i][j] for j in return_cols] for i in rows] if key=="data" else value) for key,value in self.__dict__.items()}

    dict['column_names'] = [self.column_names[i] for i in return_cols]
    dict['column_types']   = [self.column_types[i] for i in return_cols]

    s_table = Table(load=dict)

    s_table.data = list(set(map(lambda x: tuple(x), s_table.data))) if distinct else s_table.data

    if order_by:
        s_table.order_by(order_by, desc)

    if isinstance(limit,str):
        s_table.data = [row for row in s_table.data if row is not None][:int(limit)]

    return s_table
```

Εικόνα 284: _select_where_with_hash

<u>Στο αρχείο extendiblehash.py:</u>

Έχουμε δημιουργήσει ένα νέο αρχείο **extendiblehash.py** το οποίο περιέχει όλες τις μεθόδους της Hash Index.

Έχουμε προσθέσει δύο κλάσεις Bucket και Hash.

Η Bucket Class δημιουργεί αντικείμενα τύπου bucket σε ένα Hash Index.

```
class Bucket:
    def __init__(self, local_depth, hash, size):
        '''
        Initialize a bucket with the given parameters.

        Args:
            local_depth: the depth of the bucket
            hash: the hash value of the bucket
            size: the size of the bucket
        '''
        self.local_depth = local_depth
        self.hash = hash
        self.size = size
        self.records = []
```

Εικόνα 295: class Bucket def __init__

Η Hash Class δημιουργεί αντικείμενα Hash. Κάθε μία μέθοδος στην Hash υλοποιεί και κάποια λειτουργία της index:

```
class Hash:
    def __init__(self, bucket_size = 4, global_depth = 1, directory_size = 2):
        '''
        Initialize an exendible hash table with the given parameters.

        Args:
            bucket_size: the size of each bucket
            global_depth: the depth of the hash table
            directory_size: the size of the directory
        '''
        self.bucket_size = bucket_size
        self.global_depth = global_depth
        self.directory = {}
        self.directory_size = directory_size
        self.directory = {i: Bucket(1, f"{i:b}", bucket_size) for i in range(2)}
        self.buckets_to_split = []
```

Εικόνα 306: class Hash def __init__

- *insert*: Εισάγει στο hash table το **record = (key, idx)**, όπου key η τιμή του πεδίου στο πρωτεύων/ υποψήφιο κλειδί του πίνακα στη πλειάδα **idx**.

```
def insert(self, record):
    '''
    Insert the given record into the hash table.

    Args:
        record: the record to be inserted
    '''
    key, idx = record
    pointer = int(self.binary_hash_value(key)[-self.global_depth:],2)
    if len(self.directory[pointer].records) < self.bucket_size:
        self.directory[pointer].records.append(record)
    else:
        if self.global_depth == self.directory[pointer].local_depth:
            pointer = self.expand_directory(pointer,key)
            self.rehash_bucket()
        elif self.global_depth > self.directory[pointer].local_depth:
            pointer = self.bucket_split(pointer,key)
            self.rehash_bucket()
        self.directory[pointer].records.append(record)
```

Εικόνα 317: class Hash def insert

- *find:* Εύρεση πλειάδας πίνακα σύμφωνα με το **key** πεδίο του πίνακα.

```
def find(self,key):
    '''
    Find the record with the given key.

    Args:
        key: the key to be found
    '''
    remainder_lsb = self.get_remainder(key)
    pointer = int(self.get_binary(remainder_lsb,self.global_depth)[-self.global_depth:],2)
    for record in self.directory[pointer].records:
        if key == record[0]:
            return record[1]#return index
    return -1
```

Εικόνα 28: class Hash def find

- *get_binary:* Επιστρέφει το key σε δυαδική μορφή για την εύρεση του κατάλληλου **bucket**.

```
def get_binary(self, key, depth):
    '''
    Convert the given key to a binary string of the given depth.

    Args:
        key: the key to be converted
        depth: the depth of the binary string
    '''
    return f"{key:0{depth}b}"
```

Εικόνα 29: class Hash def get_binary

- *binary_hash_value:* Επιστρέφει το **key** σε δυαδική μορφή σύμφωνα με το **global_depth** του **directory**.

```
def binary_hash_value(self, key):
    '''
    Convert the given key to a binary string of the global depth.

    Args:
        key: the key to be converted
    '''
    if isinstance(key, str):
        key = abs(sum([ord(c) for c in key]))
    return f"{key & ((1 << self.global_depth) - 1):b}"
```

Εικόνα 30: class Hash def binary_hash_value

- *get_remainder:* Επιστρέφουμε το υπόλοιπο του **key % self.directory_size** δηλαδή επιστρέφουμε την μάσκα του κλειδιού με το **LSB**.

```
def get_remainder(self, key):
    '''
    Get the remainder of the given key.

    Args:
        key: the key to be converted
    '''

    if (isinstance(key, str)):
        key = abs(sum([ord(c) for c in key]))
    remainder_lsb = key % self.directory_size
    return remainder_lsb
```

Εικόνα 321: class Hash def get_remainder

- *rehash_bucket:* αναταξινομεί όλα τα **records** που βρίσκονται μέσα στο **bucket**.

```
def rehash_bucket(self):
    '''
    Rehash the records in the bucket to be split.
    '''
    bucket = self.buckets_to_split
    bucket_records = bucket.records.copy()
    bucket.records.clear()
    for key, index in bucket_records:
        remainder_lsb = self.get_remainder(key)
        new_ptr = int(self.get_binary(remainder_lsb,self.global_depth)[-self.global_depth:],2)
        bucket = self.directory[new_ptr]
        bucket.records.append((key, index))
        bucket.hash = self.get_binary(remainder_lsb,self.global_depth)[-self.global_depth:]
```

Εικόνα 332: class Hash def rehash_bucket

- *bucket_split:* χωρίζει **buckets** τα οποία έχουν γίνει **overflow** σε δύο καινούργια **buckets**.

```
def bucket_split(self, pointer, key):
    '''
    Split the bucket at the given pointer.

    Args:
        pointer: the pointer to the bucket to be split
        key: the key to be inserted
    '''
    self.directory[pointer].local_depth += 1
    self.buckets_to_split = self.directory[pointer]
    remainder_lsb = self.get_remainder(key)
    key_in_binary = self.get_binary(remainder_lsb, self.global_depth -1)
    for i in [0, 1]:
        new_ptr = "{}".format(i) + key_in_binary[-(self.global_depth-1):][-self.global_depth:]
        new_ptr_int = int(new_ptr,2)
        new_bucket = Bucket(self.directory[pointer].local_depth,new_ptr,self.bucket_size)
        self.directory[new_ptr_int] = new_bucket
    remainder_lsb = self.get_remainder(key)
    new_ptr = int(key_in_binary[-self.global_depth:],2)
    return new_ptr
```

Εικόνα 343: class Hash def bucket_split

- *expand_directory:* διπλασιάζει το **directory** όταν αυξάνεται το **global_depth** κατά ένα.

```
def expand_directory(self, pointer, key):
    '''
    Expand the directory by doubling its size.

    Args:
        pointer: the pointer to the bucket to be split
        key: the key to be inserted
    '''
    self.global_depth += 1
    self.directory_size *= 2
    new_ptr = self.bucket_split(pointer,key)
    for key_ in self.directory.copy():
        bucket = self.directory[key_]
        if bucket.local_depth < self.global_depth:
            hash_ = bucket.hash
            for key in range(self.directory_size):
                if self.directory.get(key) is None:
                    binary_ver = self.get_binary(key,bucket.local_depth)
                    if binary_ver[-bucket.local_depth:] == hash_:
                        self.directory[key] = bucket
    return new_ptr
```

Εικόνα 354:class Hash def expand_directory

- *delete:* Διαγράφει ένα στοιχείο από το **Extendible Hash Index**, σε περίπτωση που διαγραφθεί η αντίστοιχη πλειάδα από τον πίνακα της βάσης. (Κατά την διαγραφή των εγγραφών από το bucket που αντιστοιχεί στην συγκεκριμένη υλοποίηση υπολείπεται της μεθόδου merge_bucket που είναι υπεύθυνη για την συγχώνευση των buckets.)

```
def delete(self,key):
    '''
    Delete the record with the given key from the bucket.

    Args:
        key: the key to be deleted
    '''
    remainder_lsb = self.get_remainder(key)
    pointer = int(self.get_binary(remainder_lsb,self.global_depth)[-self.global_depth:],2)
    for record in self.directory[pointer].records:
        if key == record[0]:
            self.directory[pointer].records.remove(record)
            # if len(self.directory[pointer].records) == 0:
            #       CREATE METHOD: self.merge_bucket(pointer)
            return
    return -1
```

Εικόνα 365: class Hash def delete

Παραδείγματα Εκτέλεσης Κώδικα

```
ht: <extendiblehash.Hash object at 0x00000221DEC03F70>
 special variables
 function variables
  bucket_size: 4
 buckets_to_split: <extendiblehash.Bucket object at 0x00000221DEBCE1F0>
 directory: {0: <extendiblehash.Buck...1DEBCED90>, 1: <extendiblehash.Buck...1DEBCE460>, 2: <extendiblehash.Buck...1DEBCE880>, 3: <extendiblehash.Buck...1DEBCE460>}
  special variables
  function variables
  0: <extendiblehash.Bucket object at 0x00000221DEBCED90>
   special variables
    hash: '00'
    local_depth: 2
   records: [('finance', 3), ('biologp', 7)]
    size: 4
  1: <extendiblehash.Bucket object at 0x00000221DEBCE460>
   special variables
    hash: '1'
    local_depth: 1
   records: [('biology', 0), ('elec. eng.', 2), ('music', 5), ('physics', 6)]
    size: 4
  2: <extendiblehash.Bucket object at 0x00000221DEBCE880>
   special variables
    hash: '10'
    local_depth: 2
   records: [('comp. sci.', 1), ('history', 4), ('bioltgyfogp', 8)]
    size: 4
  3: <extendiblehash.Bucket object at 0x00000221DEBCE460>
   special variables
    hash: '1'
    local_depth: 1
   records: [('biology', 0), ('elec. eng.', 2), ('music', 5), ('physics', 6)]
    size: 4
   len(): 4
  directory_size: 4
  global_depth: 2
```

Εικόνα 37: create index dept_name on course (dept_name) using hash;

```
(smdb)> select * from course
course_id (str) #PK#    title (str)              dept_name (str) #UQ#      credits (int)
----------------------  -----------------------  ----------------------  --------------
bio-101                 intro. to biology        biology                              4
cs-101                  intro. to computer science comp. sci.                         4
ee-181                  intro. to digital systems elec. eng.                          3
fin-201                 investment banking       finance                              3
his-351                 world history            history                              3
mu-199                  music video production    music                                3
phy-101                 physical principles      physics                              4
301                     genetics                 biologp                             85
30uy1                   genetics                 bioltgyfogp                          0
```

*Εικόνα 38: select * from course*

Για το SQL query: **select * from course where dept_name = “biology”;**

```
miniDB > database.py > Database > select
383                 index_name,index_type = self.select('index_name,index_type', 'meta_indexes', f'table_name={table_name} and column_name={conditions_columns[0]}', return_object=True).data[0]
384                 if index_name is not None and index_type == 'btree':
385                     bt = self._load_idx(index_name)
386                     table = self.tables[table_name]._select_where_with_btree(columns, bt, condition, distinct, order_by, desc, limit)
387                 elif index_name is not None and '=' in condition_list[0] and index_type == 'hash':
388                     ht = self._load_idx(index_name)
389                     table = self.tables[table_name]._select_where_with_hash(columns, ht, condition, distinct, order_by, desc, limit)
```

Εικόνα 39: Εντοπίζει την ύπαρξη index hash πάνω στο unique: dept_name

```
miniDB > table.py > Table > _select_where_with_hash
362             print('Column is not PK. Aborting')
363
364         # hash find
365         if operator == '=':
366             rows = [ht.find(value)]
367         else:
368             raise ValueError('Hash table only supports equality operator')
369
370         dict = {(key):([[self.data[i][j] for j in return_cols] for i in rows] if key=="data" else value) for key,value in self.__dict__.items()}
371
```

Εικόνα 40: Εντόπισε τα rows που υποστηρίζουν την συνθήκη του where statement.

```
> return_cols: [0, 1, 2, 3]
  return_columns: '*'
> rows: [0]
> self: <miniDB.table.Table object at 0x00000221DEBDFC10>
  value: 'biology'
```

*Εικόνα 41: οι τιμές που επέστρεψε από το hash index: **rows: [0]***

```
(smdb)> select * from course where dept_name = "biology";
course_id (str) #PK#    title (str)       dept_name (str) #UQ#      credits (int)
----------------------  ----------------  ----------------------  --------------
bio-101                 intro. to biology biology                              4
```

Εικόνα 42: το αποτέλεσμα του query

Επιπλέον Χαρακτηριστικά που προσθέσαμε στον πηγαίο κώδικα

Αρχείο mdb.py

- Έχουμε προσθέσει στη evaluate_where_clause την διαχείριση NOT BETWEEN condition. Όταν έχουμε condition με **'not between'** μετατρέπουμε το condition από "column **NOT BETWEEN** value1 **AND** value2" σε "column < value1 **AND** column > value2". Αυτή η μετατροπή μπορεί να συμβεί και για πολλαπλά **' not between '** στο where statement.

```
#for ever not between, delete not, between keywords and change the condition to "column < value1 and column > value2"
while not_between_idx:
    not_between_idx = not_between_idx[0]
    column_name = where_split[not_between_idx-1]
    value1= where_split[not_between_idx+2]
    value2= where_split[not_between_idx+4]

    where_split_right = ' '.join(where_split[not_between_idx+5:])
    where_split_left = ' '.join(where_split[:not_between_idx-1])
    dic['where'] = where_split_left + ' ' + column_name + " < " + value1 + " or " + column_name + " > " + value2 + ' ' + where_split_right

    where_split = split_statement(dic['where'])
    not_between_idx = [i for i,word in enumerate(where_split) if word=='not' and where_split[i+1]=='between' and not in_paren(where_split,i)]
    not_idx = [i for i,word in enumerate(where_split) if word=='not' and not in_paren(where_split,i)]
    between_idx = [i for i,word in enumerate(where_split) if word=='between' and not in_paren(where_split,i)]
```

Εικόνα 43: def evaluate_where_clause

Παραδείγματα Εκτέλεσης Κώδικα

```
(smdb)> select * from department where budget not between 90000 and 100000
dept_name (str) #PK#    building (str)      budget (int) #UQ#
----------------------  ----------------  ------------------
elec. eng.              taylor                         85000
finance                 painter                       120000
history                 painter                        50000
music                   packard                        80000
physics                 watson                         70000
```

*Εικόνα 44: select * from deparment where budget not between 90000 and 100000;*

- Προσθέσαμε στην *main* μία συνθήκη ελέγχου έλλειψης SQL query στο terminal του προγράμματος, ώστε να το πρόγραμμα να μην τερματίζεται αυτόματα αλλά να δημιουργεί μία νέα γραμμή που θα περιμένει το input του χρήστη.

```
print(art)
session = PromptSession(history=FileHistory('.inp_history'))
while 1:
    try:
        line = session.prompt(f'({db._name})> ', auto_suggest=AutoSuggestFromHistory()).lower()
        if line=='': continue
        elif line[-1]!=';':
            line+=';'
    except (KeyboardInterrupt, EOFError):
        print('\nbye!')
        break
```

Εικόνα 45: main method

Παραδείγματα Εκτέλεσης Κώδικα

```
(smdb)>
(smdb)>
```

Εικόνα 46: Enter key pressed the program does not crash when the user hit enter

Αρχείο database.py

- Στη μέθοδο *create_index* η βάση ελέγχει ποιο χαρακτηριστικό του πίνακα έχει επιλεγεί για να δημιουργηθεί index. Αν ο χρήστης της βάσης δεν δηλώσει κάποιο χαρακτηριστικό στο SQL query, το πρόγραμμα παίρνει αυτόματα την στήλη primary key.

```
if self.tables[table_name].pk_idx is None and self.tables[table_name].unique_idx is None: # if no primary key or any unique column, no index
    raise Exception('Cannot create index. Table has no primary key or unique columns.')
if index_name not in self.tables['meta_indexes'].column_by_name('index_name'):
    # currently only btree and extendible hash is supported. This can be changed by adding another if.
    if column == None:
        column = self.tables[table_name].pk
```

Εικόνα 47: def create_index

Αρχείο table.py

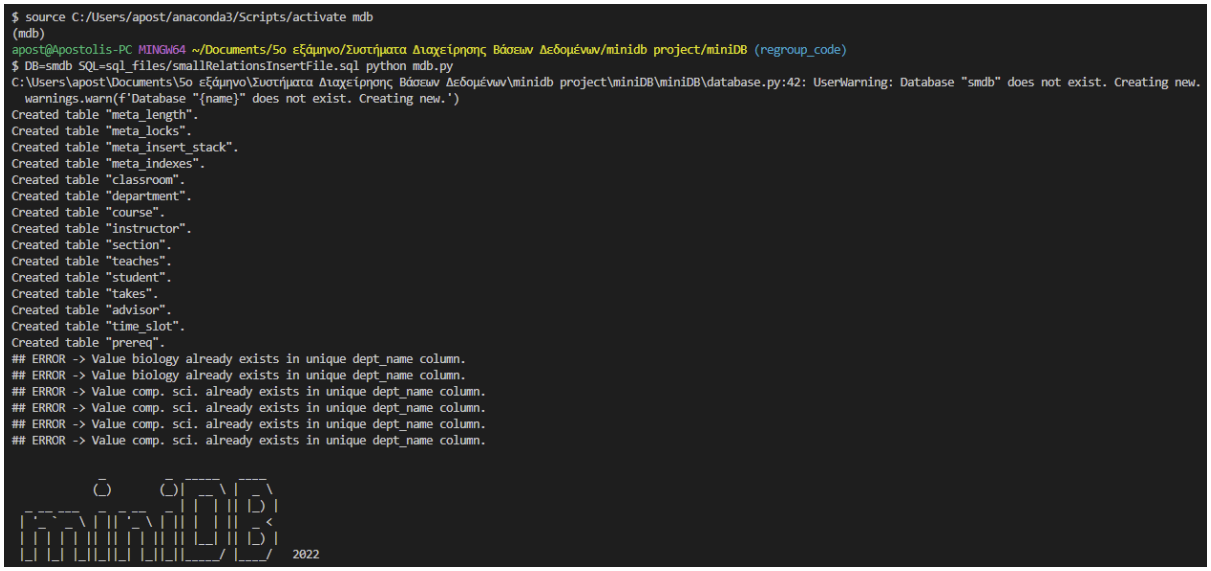
- Μέσω της μεθόδου _insert το πρόγραμμα εκτυπώνει στο terminal πιθανά σφάλματα στα values της πλειάδας που επιθυμεί ο χρήστης να προσθέσει σε έναν πίνακα της βάσης. Στην περίπτωση που ο χρήστης επιθυμεί να προσθέσει διαφορετικών τύπων δεδομένα από αυτά των χαρακτηριστικών του πίνακα ή παραπάνω δεδομένα από αυτά που δέχεται ο πίνακας, τότε το SQL query ακυρώνεται και δεν προστίθεται κάποια επιπλέον πλειάδα στον πίνακα.

```
def _insert(self, row, insert_stack=[]):
    '''
    Insert row to table.

    Args:
        row: list. A list of values to be inserted (will be casted to a predifined type automatically).
        insert_stack: list. The insert stack (empty by default).
    '''
    try:
        if len(row)!=len(self.column_names):
            raise ValueError(f'## ERROR -> Cannot insert {len(row)} values. Only {len(self.column_names)} columns exist')

        for i in range(len(row)):
            # for each value, cast and replace it in row.
            try:
                row[i] = self.column_types[i](row[i])
            except ValueError:
                if row[i] != 'NULL':
                    raise ValueError(f'## ERROR -> Value {row[i]} of type {type(row[i])} is not of type {self.column_types[i]}.')
            except TypeError as exc:
                if row[i] != None:
                    print(exc)

            # if value is to be appended to the primary_key column, check that it doesnt already exist (no duplicate primary keys)
            if i==self.pk_idx and row[i] in self.column_by_name(self.pk):
                raise ValueError(f'## ERROR -> Value {row[i]} already exists in primary key column.')
            elif i==self.pk_idx and row[i] is None:
                raise ValueError(f'## ERROR -> The value of the primary key cannot be None.')

            # if value is to be appended to a unique column, check that it doesnt already exist (no duplicate unique values)
            if self.unique is not None:
                for j in range(len(self.unique)):
                    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 {self.unique[j]} column.')
                    elif i==self.unique_idx[j] and row[i] is None:
                        raise ValueError(f'## ERROR -> The value of the unique column {self.unique[j]} cannot be None.')
    except ValueError as exc:
        print(exc)
        return
```

Εικόνα 48: def _insert

Παραδείγματα Εκτέλεσης Κώδικα

Για την δημιουργία της βάσης δεδομένων η οποία περιέχει ως unique column την στήλη dept_name του πίνακα course.



Εικόνα 49: database creation error for unique columns data

Αρχικά, έγινε προσπάθεια εισαγωγής στοιχείου στην στήλη του primary key όπου ήδη υπήρχε ως τιμή.

Στην συνέχεια, έγινε προσπάθεια εισαγωγής στοιχείου στην στήλη που έχει οριστεί ως unique όπου υπήρχε ήδη ως τιμή.

Επίσης, έγινε προσπάθεια εισαγωγής κειμένου σε στήλη που έχει οριστεί ως στήλη που δέχεται μόνο ακέραιους αριθμούς.

Τέλος, έγινε μια επιτυχής εισαγωγή δεδομένου στον πίνακα course.

```
(smdb)> insert into course values (BIO-101,Genetics,Biology,four);
## ERROR -> Value bio-101 already exists in primary key column.
(smdb)> insert into course values (BIO-102,Genetics,Biology,four);
## ERROR -> Value biology already exists in unique dept_name column.
(smdb)> insert into course values (BIO-102,Genetics,mathematics,four);
## ERROR -> Value four of type <class 'str'> is not of type <class 'int'>.
(smdb)> insert into course values (BIO-102,Genetics,mathematics,5);
```

Εικόνα 50: try to insert false data into table: Error messages example for insert values

- Στη συνάρτηση *show* προσθέσαμε επιπλέον ελέγχους που εκτυπώνουν σωστά τις στήλες των πινάκων που είναι **Primary key** ή **Unique**.

```python
def show(self, no_of_rows=None, is_locked=False):
    '''
    Print the table in a nice readable format.

    Args:
        no_of_rows: int. Number of rows.
        is_locked: boolean. Whether it is locked (False by default).
    '''
    output = ""
    # if the table is locked, add locked keyword to title
    if is_locked:
        output += f"\n## {self._name} (locked) ##\n"
    else:
        output += f"\n## {self._name} ##\n"

    # headers -> "column name (column type)"
    headers = []
    for col, tp in zip(self.column_names, self.column_types):
        if self.pk == col:
            headers.append(f'{col} ({tp.__name__}) #PK#')
        elif self.unique is not None and col in self.unique:
            headers.append(f'{col} ({tp.__name__}) #UQ#')
        else:
            headers.append(f'{col} ({tp.__name__})')
    # detect the rows that are no tfull of nones (these rows have been deleted)
    # if we dont skip these rows, the returning table has empty rows at the deleted positions
    non_none_rows = [row for row in self.data if any(row)]
    # print using tabulate
    print(tabulate(non_none_rows[:no_of_rows], headers=headers)+'\n')
```

Εικόνα 51: def show

Παραδείγματα Εκτέλεσης Κώδικα

```
(smdb)> select title,dept_name,course_id from course
title (str)               dept_name (str) #UQ#    course_id (str) #PK#
------------------------  --------------------  ---------------------
intro. to biology         biology               bio-101
intro. to computer science comp. sci.           cs-101
intro. to digital systems  elec. eng.           ee-181
investment banking        finance               fin-201
world history             history               his-351
music video production     music                 mu-199
physical principles       physics               phy-101
genetics                  mathematics           bio-102
```

Εικόνα 52: print correct PK and UNIQUE column