Readme

wx641&kw2669

Solutions

- Storage
  - all_table_dict = {} : store all result table
    - key: table name
    - value: result table
      - 2D array
      - first row as column name
  - hash_index_dict = {} : store all hash indexes
    - key: tablename_columnname
    - value: index of this tablename_columnname
  - Btree_index_dict = {}: store all Btree indexes
    - key: tablename_columnname
    - value: index of this tablename_columnname
  - col_dict = {} : record the type of index
    - key: tablename_columnname
    - value: hash or btree
- Algorithms
  - read standard input line by line
  - delete all comment and space
  - use re package to identify each operation
  - parse each operation to cetain elements, such as table name, column name, terms
  - deal with the terms according to the operation
  - after each operation, store the result table in all_table_dict
- File I/O
  - terminal command : python parseinput.py --filename input.txt
    - input.txt is the filename
    - we use the package argparse to deal with stdin input
  - inputfromfile file name type: filename.txt
  - outputtofile file name type: tablename.txt
    - first row: column name
  - 'sampleoutput.txt': record the running time of each operation

Functions

- Inputfromfile:
  - inputfromfile(items):parse string command to meaningful elements
  - txt_to_matrix(filename): Function that read the file line by line and store the data of one file into an array
- select:
  - find_col(arr,s): return the entire column by table name and column name
  - find_col_num(arr,s): return the column number by table name and column name
  - relop(s,loc): return flag of relop with column located before relop

relop	1	2	3	4	5	6
flag	<=	>=	!=	=	<	>

  - col_content(data, arithhop, col, const, i): calculate attribute [arithhop constant] part
  - parse_terms(terms) : parse attribute[arithhop constant] part
  - parse_condition(ori_table, s, res, data):
    - Parse the terms of '<=', '>=', '!=', and '='
    - Select rows according to relop
      - if relop is '=', first check whether the current column is the index of the table. If yes, find the index and use index to select rows
  - select(item, dict):
    - Parse the operation of select
    - Use regular expression to distinguish the requirement of the select and divide it to three conditions: normal condition, 'or' condition and 'and' condition
      - for 'or' condition, parse each terms in order and append each selected items to result table, then delete duplicated items
      - for 'and' condition, parse each terms in order and append each time select items from the result table
- project(items, dict)
  - Parse the operation of project
  - Select columns
- avg(item,dict):
  - Parse the operation of avg
  - Find columns and change column type to int
  - Calculate the avg
- sum(item,dict):
  - Parse the operation of sum
  - Find columns and change column type to int
  - Calculate the sum
- count(item,dict):
  - Parse the operation of count
  - return the length of certain column
- sumgroup(item,dict):

    - Divide the requirement (how many colunmns we need to do) of groupby using if statement
    - Append all the data from an array to a list
    - Sort the columns which need to be grouped first use the function: sorted(datalist[1:], key = (lambda x: column)
    - Change the value that we need to add from string to int
    - And use i to compare the group one line by one line
    - If one line is not equal to the line before, add all the value in the column before in this group
    - Append the column name to the list and then append the data(Sum, Columns of groupby) to a list
    - Use np.asarray function to change the format from list to array
- avggroup(item,dict):
    - Operations like Sumgroup Function
    - Avg = Sum / Count
    - Keep two decimal places by round(avg, 2)
    - Append the data(Avg, Columns of groupby) to a list and to an array
- countgroup(item,dict):
    - Operations like Sumgroup Function
    - Count = Numbers in a group
    - Append the data(Count, Columns of groupby) to a list and to an array
- join:
    - join_equal(all_table_dict, table1, table2, term)
        - join two tables on the first time '=' appears
        - check whether the attibute of each table is index.
            - if table1 has index, iterate table2 and find the rows accordingly to the value of table2 with index, join rows
            - if table2 has index, iterate table1 and find the rows accordingly to the value of table1 with index, join rows
            - else, iterate table1, record a dict checked{} to find the rows of table2 which have same value as the current value in table1 to reduce the times of iteration on table 2
    - parse_join_terms(var1, var2):parse attribute[arithhop constant] part
    - join_terms(op, terms, mat): calculate attribute[arithhop constant] part
    - join(items,all_table_dict)
        - Parse the operation of join
        - Use regular expression to distinguish the requirement of the join and divide it to two conditions: normal condition and 'and' condition
            - for 'and' condition
                - first find the tems with '=', and join two tables according to the term to be the first result table
                - parse other terms in order and select items accordingly from the result table
- sort:
    - Divide the requirement of sort
    - Change the format of the value that we need to sort from string to int
    - Use the function: sorted(datalist[1:], key = (lambda x: column)
- movavg:
    - running_mean(l,n): calculate n moving average on column l

    - movavg(items, dict): parse the operation of movavg
- movsum:
    - moving_sum(l,n): calculate l moving sum on column l
    - movsum(items, dict): parse the operation of movsum
- concat(item,dict): concat two array
- Btree(item,dict,btree_dict,col_dict):
    - Parse the operation of outputtofile
    - use a list to record all row numbers of a certain value
    - put the value and the list in a Btree
- hash(item,dict,hash_dict,col_dict):
    - Parse the operation of outputtofile
    - use a list to record all row numbers of a certain value
    - hash the list to the column
- outputtofile:
    - Parse the operation of outputtofile
    - use np.savetxt() to output array

Running Time of Each Operation

R:=inputfromfile(sales1)
inputfromfile running time:0.00152802467346s

R1:=select(R,(time>50)or(qty<30))
select running time:0.00789904594421s

R2:=project(R1,saleid,qty,pricerange)
project running time:4.31537628174e-05s

R3:=avg(R1,qty)
avg running time:0.00269412994385s

R4:=sumgroup(R1,time,qty)
sumgroup running time:0.00335097312927s

R5:=sumgroup(R1,qty,time,pricerange)
sumgroup running time:0.00331282615662s

R6:=avggroup(R1,qty,pricerange)
avggroup running time:0.0113599300385s

S:=inputfromfile(sales2)
inputfromfile running time:0.381053924561s

T:=join(R,S,R.customerid=S.C)
join running time:1.52790904045s

T1:=join(R1,S,(R1.qty>S.Q)and(R1.saleid=S.saleid))
join running time:7.29016590118s

T2:=sort(T1,S_C)
sort running time:0.00353908538818s

T2prime:=sort(T1,R1_time,S_C)
sort running time:0.0030369758606s

T3:=movavg(T2prime,R1_qty,3)
movavg running time:0.00102615356445s

T4:=movsum(T2prime,R1_qty,5)
movsum running time:0.00034499168396s

Q1:=select(R,qty=5)
select running time:0.00322794914246s

Btree(R,qty)
Btree running time:0.00217008590698s

Q2:=select(R,qty=5)
select running time:0.00218200683594s

Q3:=select(R,itemid=7)
select running time:0.00304007530212s

Hash(R,itemid)
Hash running time:0.001620054245s

Q4:=select(R,itemid=7)
select running time:0.000173807144144165s

Q5:=concat(Q4,Q2)
concat running time:3.91006469727e-05s

outputtofile(Q5,Q5)
outputtofile running time:0.00566911697388s

outputtofile(T,T)
outputtofile running time:0.0135281085968s