

## CS562 - The Project

## The Idea

```
# EMF query in ESQL
ESQL Query:
select cust, count(ny.quant), sum(nj.quant), max(ct.quant)
  from sales
group by cust; ny, nj, ct
such that ny.cust = cust and ny.state = 'NY',
          nj.cust = cust and nj.state = 'NJ' and nj.quant > avg(ny.quant),
               → . . . example of a predicate for a "dependent aggregate" . . .
               \Rightarrow 2.cust = cust and 2.state = 'NJ' and 2.quant > avg 1 quant
          ct.cust = cust and ct.state = 'CT'
# EMF query in Phi operation
Phi Expression:
# 1. S - projected columns / expressions
cust, count 1 quant, sum 2 quant, max 3 quant
# 2. n - number of grouping variables
# 3. V - grouping attributes
cust
# 4. F-VECT - vector of aggregate functions
count 1 quant, sum 2 quant, avg 2 quant, max 3 quant
# 5. PRED-LIST - list of predicates for grouping var's
1.state = 'NY'; 2.state = 'NJ'; 3.state = 'CT'
# 6. HAVING
NONE
# PART 1 of 2: define the data structure (mf struct)
struct {
       char
             cust[50];
                                     #-- 3. V (grouping attrib)
                                    #-- 4. F-VECT (list/vector of agg. Func's)
#-- 4. F-VECT (list/vector of agg. Func's)
       int
              count_1_quant;
               sum_2_quant;
       int
       int
               max_3_quant;
                                     #-- 4. F-VECT (list/vector of agg. Func's)
} mf struct [500];
       Example of your project code generating the code to define the 'mf_struct'
       for the ESQL above:
       To get the details of the schema of the 'sales' table, you have 2 options:
       1. Have hard-coded schema data for 'sales' table in your project code,
               e.g., schema[("cust", "varchar(50)"), ("prod", "varchar(50)"), ...]
       2. Use 'information schema.columns' table to get the schema information...
        # sample code of how you to generate the code for mf_struct
        # the following code assumes that you're storing the 6 operands of Phi
        # in separate variables: e.g., you can have a variable V (of array or list type)
```



```
# to store the list of group-by attributes and their schema data and F_VECT
       # (of array or list type) to store the list of aggregate functions and the corresponding
       # types, etc.
       printf ("struct {\n");
       printf (" %s %s;\n", F_VECT[0].type, F_VECT[0].agg);
printf (" %s %s;\n", F_VECT[1].type, F_VECT[1].agg);
printf (" %s %s;\n", F_VECT[2].type, F_VECT[2].agg);
       printf (") mf struct[500];\n");
       NUM OF ENTRIES = 0;
int
# PART 2 of 2: processing logic
lookup (cur_row)
#-- search for a given "group by" attrib. value(s) in mf struct
       for (i = 0; i < NUM_OF_ENTRIES; i++)</pre>
              if (mf struct [i].cust == cur row.cust)
                    return i;
       return -1:
}
add (cur row)
#-----
#-- adds a new entry in mf struct, corresponding to a newly found "group by" attrib. value
{
       mf_struct[NUM_OF_ENTRIES].cust = cur_row.cust;
       mf struct[NUM OF ENTRIES].count 1 quant = 0;
       mf struct[NUM_OF_ENTRIES].sum_2_quant = 0;
       mf_struct[NUM_OF_ENTRIES].max_3_quant = -1;
       NUM OF ENTRIES++;
}
output ()
#-- adds a new entry in mf_struct, corresponding to a newly found "group by" attrib. value
       printf (". . . . .\n");
                                    # header of the output (from operand S)
       for (int i=0; i<NUM_OF_ENTRIES; i++)</pre>
              printf ("%s
                            કત
                                           %d\n",
              mf struct[i].cust,
              mf_struct[i].count_1_quant,
              mf struct[i].sum 2 quant,
              mf_struct[i].max_3_quant);
}
```



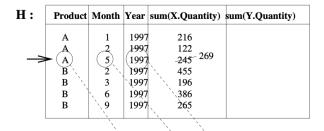
The following is the diagram from the research paper #2 ("Ad-hoc OLAP Query Processing.PDF") which outlines the processing logic for ESQL queries - the pseudo code below the diagram is based on the logic described in the diagram.

Product	Month	Year	sum(X.Quantity)	sum(Y.Quantity)

## (a) mf-structure of Query Q4

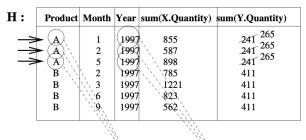
Product	Month	Year	sum(X.Quantity)	sum(Y.Quantity)
A A	1 2	1997 1997		
A	5	1997		
В	2	1997		
В	3	1997		
В	6	1997		
В	9	1997		

(b) end of first scan



	Customer	Product	Day	Month	Year	Quantity
ι.	12443	A	11	5	1997	24

## (c) during second scan



t:

Customer Product Day Month Year Quantity

12443 A 11 5 1997 24

(d) during third scan



```
main()
{
       current_row = connect_to_dbms();
                                                     # setting cursor to 1st row
       # TABLE SCAN 1: populate mf-struct with distinct values of grouping attribute (V)
        {
               if (end of table)
                      break;
               # look up current_row.cust in mf_struct
               pos = lookup (current row);
               if (pos = -1)
                                                     # current_row.cust not found in mf_struct
                       add (current_row);
               current_row.next();
                                                     # to the next row
       }
        # grouping variable 1 (1.state = 'NY')
       while()
               if (end of table)
                      break:
               if (current_row.state == 'NY')
                       # look up current_row.cust in mf_struct
                       pos = lookup (current_row, mf_struct);
                       # current_row.cust found in mf_struct
                       mf_struct[pos].count_1_quant++;
               current row.next();
                                                     # to the next row
       }
        # grouping variable 2 (2.state = 'NJ')
       while()
        {
               if (end of table)
                      break:
               if (current_row.state == 'NJ')
                       # look up current row.cust in mf struct
                       pos = lookup (current_row, mf_struct);
                       # current_row.cust found in mf_struct
                       mf_struct[pos].sum_2_quant += current_row.quant;
               current_row.next();
                                                     # to the next row
        # grouping variable 3 (3.state = 'CT')
       while()
               if (end of table)
                      break;
               if (current_row.state == 'CT')
                       # look up current_row.cust in mf_struct
                       pos = lookup (current_row, mf_struct);
                       # current_row.cust found in mf_struct
                       current_row.quant > mf_struct[pos].max_1_quant &&
```



```
mf_struct[pos].max_1_quant = current_row.quant
}
current_row.next();  # to the next row
}
# output the result
Output();
}
```



If you're using Python, the built-in data structure, "dictionary" can be useful in maintaining the internal data structures of your project code (e.g., for the Phi operands, contents of mf\_struct, etc.) - one advantage would be the ability to access an element using more meaning names as indexes vs. numeric indexes.

The following is a simple code illustrating the use of dictionary data structre.

```
# Create a dictionary of customers and their corresponding quantities
cust quant = {
   "Sam": 100,
   "Joe": 90,
   "Mia": 80,
   "Dan": 70,
   "Sue": 60
# Accessing and printing the quantities of individual customers
print("INITIAL LIST OF CUSTOMERS and QUANTITIES....")
print("Sam's quant:", cust_quant["Sam"])
print("Joe's quant:", cust_quant["Joe"])
print("Mia's quant:", cust_quant["Mia"])
print("Dan's quant:", cust_quant["Dan"])
print("Sue's quant:", cust_quant["Sue"])
# Modifying a customer's quantity
cust quant["Dan"] = 90
print("CHANGING a customer's quantity (Dan)....")
print("Dan's new quant:", cust_quant["Dan"])
# Adding a new customer and their quantity
cust quant["Claire"] = 80
print("ADDING a new customer (Claire)....")
print("Claire's quant:", cust_quant["Claire"])
# Removing a customer from the dictionary
print("REMOVING an existing customer (Joe)....")
del cust quant["Joe"]
# Printing all the customers and their quantities using a loop
print("LIST OF CUSTOMERS and QUANTITIES (after the
changes)....")
for cust, quant in cust_quant.items():
   print(cust, ":", quant)
```