Evaluation of Ad Hoc OLAP : In-Place Computation

Outline

- Motivation
- Syntax and Examples
- Evaluation Algorithm and Optimizations

Motivation

Running Example

`Sales(cust,prod,day,month,year,state,sale)`

Table Sales stores the purchases of a product by a customer on a date and state for a sale amount.

Problematic Syntax

Difficult to express ad hoc OLAP in standard SQL

Example : Find for each customer and for each month of 1997 the customer's average sale *before*, *during* and *after* this month.

Main idea : For each customer and month (i.e. for each group), define the sets of sales of the customer before, during, and after this month and compute the average sale of each set.

Challenge : Extend simply and intuitively SQL to express in a declarative way this iteration over the groups of a query.

Problematic Evaluation

Expensive to evaluate ad hoc OLAP in standard optimizers

Main idea : Complex SQL expressions lead to complex relational algebra expressions, with multiple joins, group-bys, subqueries. Traditional optimizers do not consider the “big picture”. Rather, they try to optimize a series of joins and group-bys.

Challenge : Have a simple, efficient and scalable implementation, easy to optimize.

Syntax and Examples

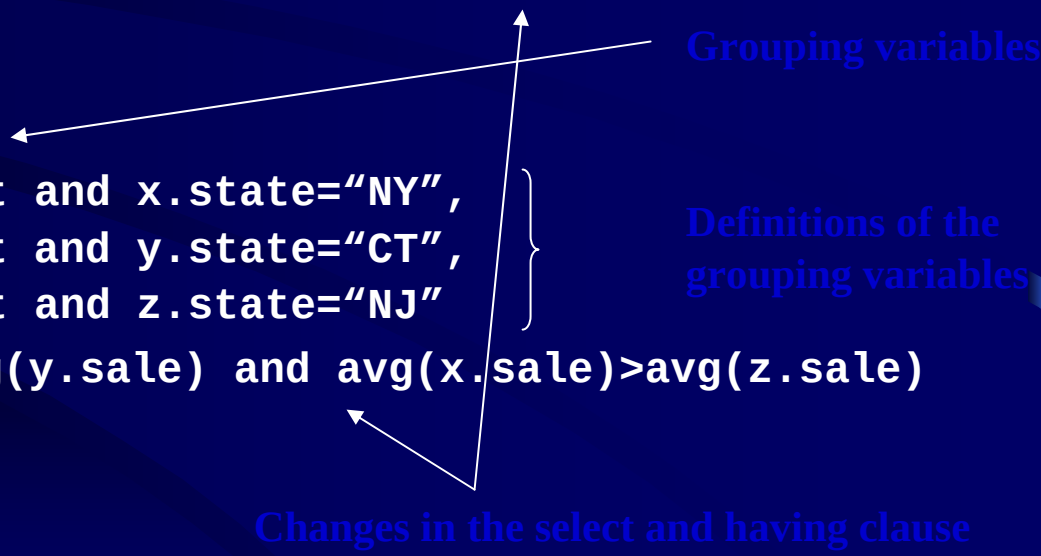
Extended SQL (EMF SQL)

- Extend the group by statement.
- Users can define *for each* group one or more sets of tuples (called grouping variables) and compute aggregates of these sets. A new clause, the such that clause, is used to define these grouping variables.
- Effectively we have an iteration over the groups.

Example 1 - Pivoting

Find for each customer the average sale in “NY”, the average sale in “CT” and the average sale in “NJ”, if New York's average is greater than the other two.

```
select cust,avg(x.sale),avg(y.sale),avg(z.sale)
from Sales
where year=1997
group by cust ; x,y,z
such that  x.cust=cust and x.state=“NY”,
           y.cust=cust and y.state=“CT”,
           z.cust=cust and z.state=“NJ”
having avg(x.sale)>avg(y.sale) and avg(x.sale)>avg(z.sale)
```



Example 2 - Hierarchies

For each product and for sales of 1997, show each month's total sales as percentage of the year-long total sales.

```
select prod, month, sum(x.sale)/sum(y.sale)
from Sales
where year=1997
group by prod,month ; x,y
such that x.prod = prod and x.month = month,
          y.prod = prod
```

Example 3 - Trends, Moving aggregates

Find for each customer and for each month of 1997 the customer's average sale *before*, *during* and *after* this month.

```
select cust,month,avg(x.sale),avg(sale),avg(y.sale)
from Sales
where year=1997
group by  cust, month ;  x, y
such that x.cust=cust and x.month < month,
          y.cust=cust and y.month > month
```

Example 4 - Dependent Aggregation

For each product count for each month of 1997 the sales that were between the previous and the following month's average sale.

```
select prod, month, count(z.*)
from Sales
where year=1997
group by prod,month ; x,y,z
such that x.prod = prod and x.month = x.month-1,
          y.prod = prod and y.month = y.month+1,
          z.prod = prod and z.month = month and
          z.sale>avg(x.sale) and z.sale<avg(y.sale)
```

Example 5 - Complex Comparisons

Compare for each customer and each product, the customer's average sale of this product versus the average sale of the product to the other customers.

```
select cust, prod, avg(x.sale), avg(y.sale)
from Sales
group by cust, prod ; x, y
such that x.cust=cust and x.prod=prod,
          y.cust<>cust and y.prod=prod
```

Example 6 - Medians

Find the for each product the median sale (we assume odd number of sales for simplicity of presentation).

```
select prod, sale
from Sales
group by prod, sale ; x, y
such that x.prod=prod
          y.prod=prod and y.sale < sale
having count(y.prod) = count(x.prod) / 2
```

Evaluation and Optimizations

Evaluation Algorithm

- The output of an EMF query can be represented by one table, called the *mf-structure* (or *mf-table*) of the query. This table has as columns the grouping attributes and all the aggregates mentioned in the query (of the group or the grouping variables).
- Perform one or more scans (one for each grouping variable); for each scanned tuple t, if the such that clause of the currently computed grouping variable *Xi* is satisfied w.r.t. some row(s) of the mf-structure, update *Xi*'s aggregates appropriately.

Example

For each product and for sales of 1997, show each month's total sales as percentage of the year-long total sales.

```
select prod, month, sum(x.sale)/sum(y.sale)
from Sales
where year=1997
group by prod,month ; x,y
such that x.prod = prod and x.month = month,
          y.prod = prod
```

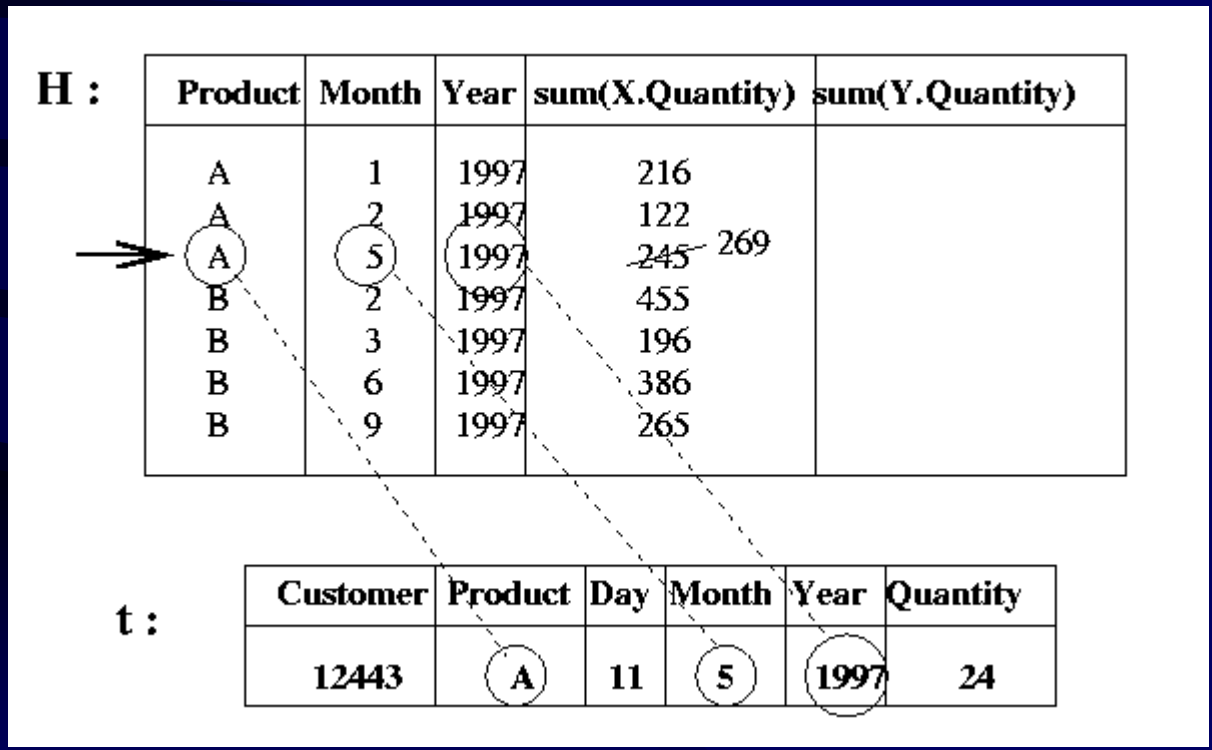
Example (cont.)

- Initial scan of Sales :

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

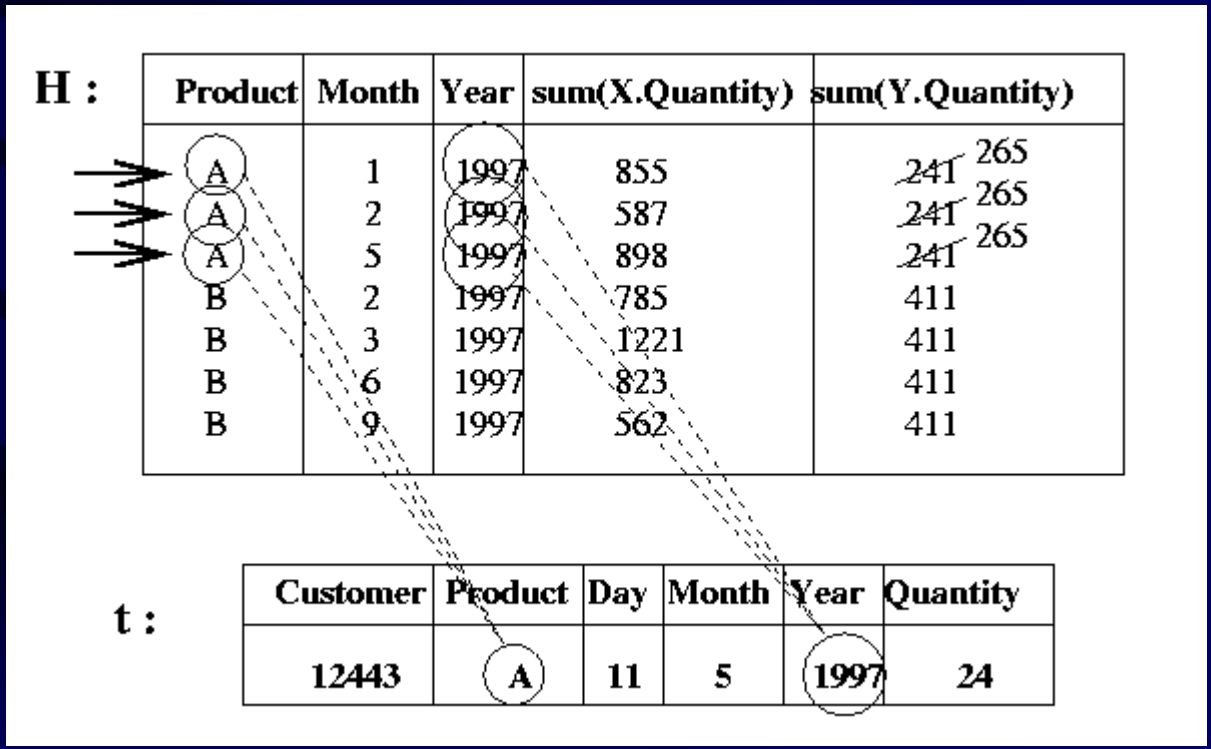
Example (cont.)

- First scan of Sales to compute g.v. x :



Example (cont.)

- Second scan of Sales to compute g.v. y :



Optimization : Indexing of the mf-structure

- **Main idea** : Reduce searching of the mf-structure

– Searching the entire mf-structure is expensive.
– Given a tuple t during evaluation of grouping var X, try to identify the relative entries of the mf-structure.
– Analyze the defining conditions of each grouping variable and keep the mf-structure as a hash table, a B+-tree, or build additional indices on the fly.

Optimization : Dependency Analysis

- **Main idea** : Reduce the number of scans.

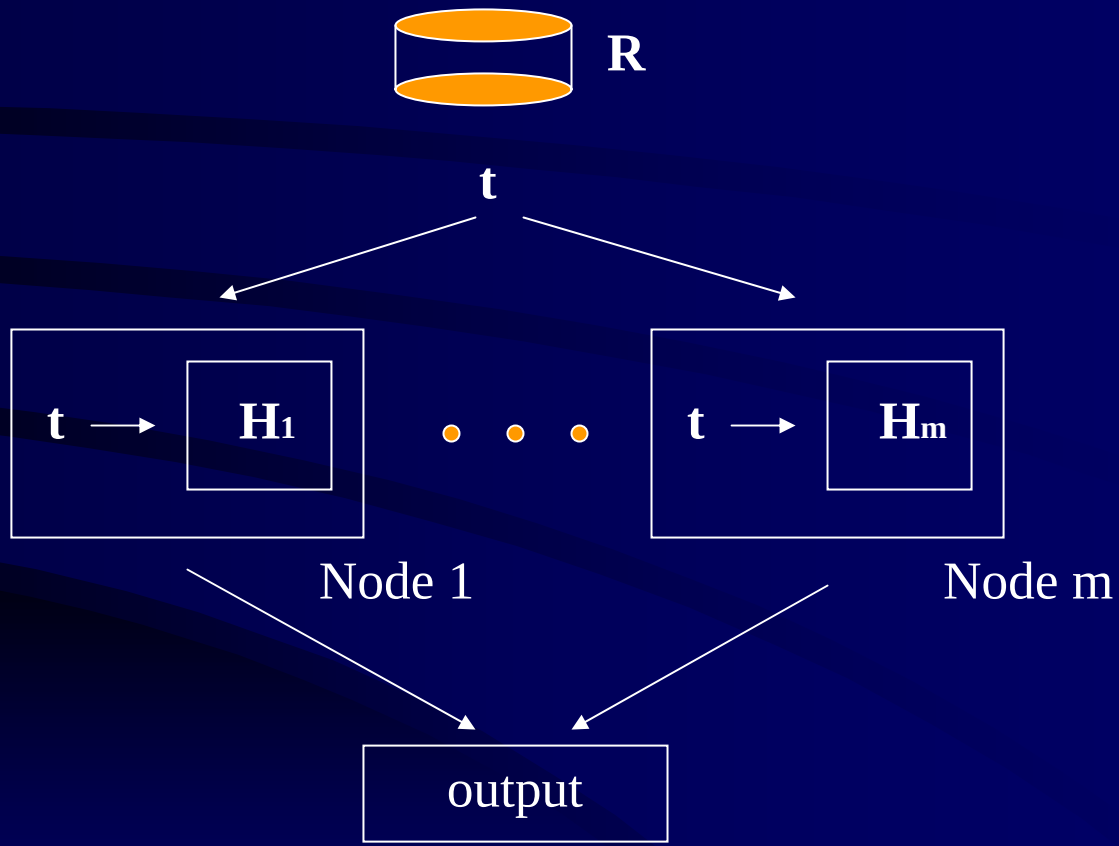
If there is no dependency between two grouping variables (aggregate of one mentioned in the defining condition of the other), then compute both in the same scan. Sometimes we can do so, even in the presence of dependencies (temporal sequences).

Optimization : More Scans, Less Memory

- **Main idea :** keep always the mf-structure memory resident.

Compute the answer in *m* "rounds" of *n* scans each, instead of just in one round. During the initial scan of each round (in which the mf-structure is created), built into memory only part of the mf-structure. The remaining scans of the round proceed as before, considering only the current part of the mf-structure.

Optimization : Parallelism



Punch Line - Key ideas

- Succinct syntax => the output of a complex query can be represented by *one* table (no intermediate tables).
- The only DB operation to compute an EMF query is scanning (or indexed scanning) => flexibility, multiple query processing.
- Different model : index the result and try to understand how a just scanned tuple affects this result (result-oriented computation).
- Scalability and parallelism are easy and *intuitive*.