Date: 05/05/2016

CS 562 - B : Team Guardians

SQL Query Processor

By Mehul Gupta, Sanjana Brid

Presentation Contents

- Problem Statement
- Our Solution
- Technical Significance
- Project Architecture
- Query Structure
- Technology Used
- Technical Limitations
- Demo

What is SQL Query Processor?

Problem Statement

Ad-hoc OLAP queries / Multi-dimensional queries when expressed in simple standard SQL, often lead to complex relational algebraic expressions with multiple joins, group-bys, and subqueries.

When faced with the challenges of processing such queries, traditional query optimizers do not consider the “big picture”. Rather, they try to optimize a series of local joins and group-bys, leading to poor performance.

Solution

Based on the published work:
[1] C. Damianos, *Evaluation of Ad Hoc OLAP: In-Place Computation*
[2] C. Damianos and R. Kenneth, *Querying Multiple Features of Groups in Relational Databases*

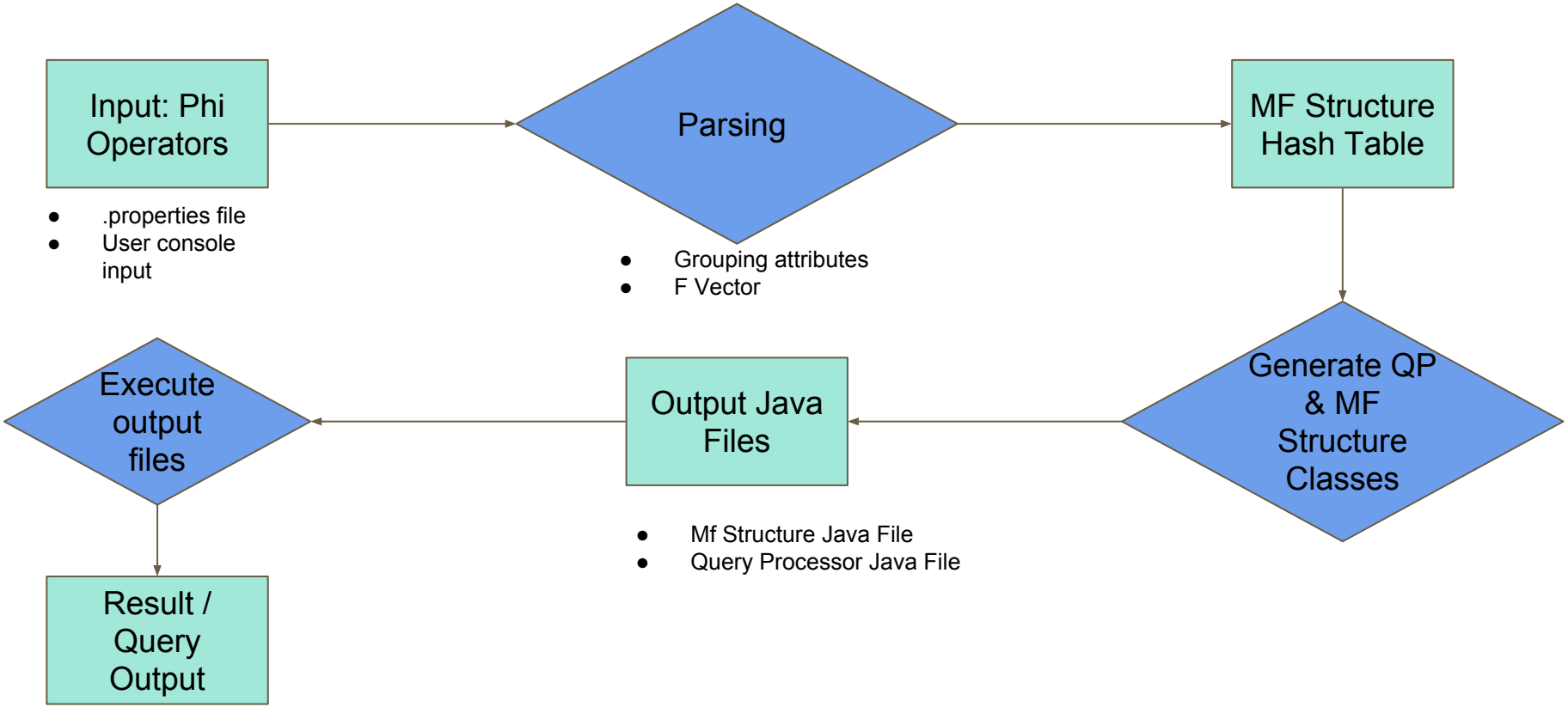
Our SQL Query Processor provides a *Syntactic Framework* that allows succinct expression of Ad Hoc OLAP Queries, by extending the *group-by* statement and introducing new clause - "*such that*".

Technical Significance

Based on the published work:
[1] C. Damianos, *Evaluation of Ad Hoc OLAP: In-Place Computation*
[2] C. Damianos and R. Kenneth, *Querying Multiple Features of Groups in Relational Databases*

Extending the standard SQL with the new syntax, provides us with a *simple*, *efficient*, and *scalable* algorithm for query processing.

Project Architecture



Query Structure

```
selection_attributes = cust,avg_quant_1,
avg_quant_2,avg_quant_3
where_clause = year\=1997
grouping_attributes = cust
no_of_gv = 3
f_vect = avg_quant_1,avg_quant_2,avg_quant_3
condition_vect = mftable[index].cust.equals(cust)
&& rs.getString("state").equals("NY"),mftable
[index].cust.equals(cust) && rs.getString("state").
equals("NJ"),mftable[index].cust.equals(cust) &&
rs.getString("state").equals("CT")
having_condition = avg_quant_1 > avg_quant_2
&& avg_quant_1 > avg_quant_3
```

- selection_attributes: List of projected attributes for query o/p
- where_clause: Predicate for the where clause
- grouping_attributes: List of grouping attributes
- no_of_gv: Number of grouping variables
- condition_vect: List of predicates to define the range for grouping variables
- having_condition: Predicate for the having clause

Technology Used

- JAVA : Programming Language
- PostgreSQL : Database Tool
- Eclipse : IDE
- External Libraries : postgres jdbc driver

Technical Limitations

- Input should be in Phi operator format.
- Keywords Supported : sum, avg, count, min, max
- Database Table : sales (default)

Demo - 0

selection_attributes = cust,prod
where_clause =
grouping_attributes = cust,prod
no_of_gv =
f_vect =
condition_vect =
having_condition =

- Display each and every customer that are associated with each product

Demo - 1

selection_attributes = cust,prod,count_year
where_clause = year\=2000
grouping_attributes = cust,prod
no_of_gv =
f_vect = count_year
condition_vect =
having_condition =

- For the year 2000 for each customer and product count number of records

Demo - 2

selection_attributes = prod,mftable[index].sum_quant_1,mftable[index].sum_quant_2, mftable[index].sum_quant_3
where_clause = year\=2008
grouping_attributes = prod
no_of_gv = 3
f_vect = sum_quant_1,sum_quant_2, sum_quant_3
condition_vect = mftable[index].prod.equals (prod) && mftable[index].month == 1, mftable [index].prod.equals(prod) && mftable[index]. month == 2, mftable[index].prod.equals(prod) && mftable[index].month == 3
having_condition =

- For 2008 , show for each product the total of January, the total of February and the total of March sales (in three columns).

Demo - 3

selection_attributes = prod,month,avg_quant_1, avg_quant_2
where_clause = year\=2008
grouping_attributes = prod,month
no_of_gv = 2
f_vect = avg_quant_1,avg_quant_2
condition_vect = mftable[index].prod.equals (prod) && month < mftable[index].month, mftable [index].prod.equals(prod) && month > mftable [index].month
having_condition =

- For each product and sales of 2008, show the product's average sale before and after each month of 2008.

Demo - 4

selection_attributes = prod,month,
count_quant_1,count_quant_2
where_clause = year\=2008
grouping_attributes = prod,month
no_of_gv = 2
f_vect = count_quant_1,count_quant_2
condition_vect = mftable[index].prod.equals
(prod) && month ==mftable[index].month-1,
mftable[index].prod.equals(prod) && month
==mftable[index].month+1
having_condition =

- For each product, count for each month of 2008, how many sales of the previous and how many sales of the following month had quantity greater than that month's average sale. (trends)

Demo - 5

selection_attributes = prod,month,mftable[index].sum_quant_1 * 100.0/mftable[index].sum_quant_2
where_clause = year\=2008
grouping_attributes = prod,month
no_of_gv = 2
f_vect = sum_quant_1,sum_quant_2
condition_vect = mftable[index].prod.equals(prod) && mftable[index].month == month, mftable[index].prod.equals(prod)
having_condition =

- For each product show each month's total sales as percentage of this product's year-long total sales. (hierarchies)

Demo - 6

selection_attributes = prod,month,mftable[index].sum_quant_1 * 100.0/mftable[index].sum_quant_2
where_clause = year\=2008
grouping_attributes = prod,month
no_of_gv = 2
f_vect = quant_1,sum_quant_1,sum_quant_2,avg_quant_2
condition_vect = mftable[index].prod.equals(prod) && mftable[index].month == month, mftable[index].prod.equals(prod)
having_condition = mftable[index].quant_1 > avg_quant_2

- For each product, find for each month the total of the sales with quantity greater than the all- product year-long average sale, as percentage of the product's year-long total sales.

Demo - 7

selection_attributes = cust,prod,avg_quant_1, avg_quant_2
where_clause =
grouping_attributes = cust,prod
no_of_gv = 2
f_vect = avg_quant_1,avg_quant_2
condition_vect = mftable[index].cust.equals(cust) && mftable[index].prod.equals(prod),!mftable [index].cust.equals(cust) && mftable[index].prod. equals(prod)
having_condition =

- “For each customer, show for each product the customer’s average sale,and the other customers’ average sale (for the same product).”

Conclusion & Future Scope

Based on the published work:
[1] C. Damianos, *Evaluation of Ad Hoc OLAP: In-Place Computation*
[2] C. Damianos and R. Kenneth, *Querying Multiple Features of Groups in Relational Databases*

As demonstrated we can extend the capabilities of standard SQL, by implementing the concepts provided in the research papers.
E-SQL provides us with simple, efficient and scalable algorithm for query processing

- Parsing input in SQL query format.
- Provide handling for more keywords

Questions????