Improving Performance of Data Analysis in Data Warehouses: A Methodology and Case Study

Janet Siebert
Data Pantheon, Inc.
391 Clarkson Street
Denver, CO 80218
303-698-2593

jsiebert@acm.org

ABSTRACT

Data analysis in very large databases requires innovative techniques. In an exploration and discovery scenario, the performance of standard SQL techniques can be unacceptable. This paper provides a methodology and case study for an alternative technique.

Keywords

Data warehouse, VLDB, data analysis, performance, synthetic join.

1. INTRODUCTION

Growing data warehouses quickly become very large databases (VLDBs), with multiple multi-million row tables. When data analysts and developers are faced with problems whose solutions involve multiple multi-million row tables, standard SQL query and join techniques are woefully inadequate. This paper presents a synthetic join technique offering satisfactory performance and numerous advantages. The advantages include:

- the ability to limit the number of rows included in the analysis;
- support for progression through the stages of exploration, hypothesis testing, and formal reports;
- the ability to record and report both summary and detail data; and
- embedded performance benchmarks.

The following discussion includes a design overview, a case study, and performance results from that case study.

2. DESIGN OVERVIEW

The fundamental design of a synthetic join is to use nested loops and key values to connect the tables. Pseudo-code is as follows:

```
select N rows from table1
foreach row selected
   output columns from table1
   select corresponding row(s) from
    table2, based on key value(s)
   foreach row selected
      output columns from table2
```

This basic algorithm is extended as the analysis evolves. In early stages of exploration, the analyst is seeking patterns. N is small, and most of the columns from both tables are displayed. Once patterns are identified, some data becomes irrelevant. The columns which are displayed as output are limited, and N is increased. At this stage, hypotheses are formulated.

Hypothesis testing is added to the algorithm through computations. A sample computation is to sum the dollar values of column x for rows in table2, and compare the sum to the dollar value of column y in table1. This introduces a metric into the reporting. The following pseudo-code would be inserted after the end of the inner foreach loop:

```
   if sum(table2.dollarX) = table1.dollarY
      success=success + 1
   else
      failure=failure + 1
```

The final step in the program is to output the number of successes and failures. At this stage, N is increased, and the percentage of successes and the percentage of failures is noted.

The algorithm evolves to support reporting by logging the detail behind the success or failure. The code becomes:

```
   if sum(table2.dollarX) = table1.dollarY
      success=success + 1
      print "SUCCESS:
      key_value, table2.dollarX,
       table1.dollarY"
   else
      failure=failure + 1
      print "FAILURE:
      key_value, table2.dollarX,
       table1.dollarY"
```

The detail of the failure can be extracted by searching the output file for the key word "FAILURE." Additionally, at this stage of the analysis, N is increased to include all rows.

3. CASE STUDY

The case study is drawn from the health insurance industry. The problem at hand is reconciling transaction detail with detail that interfaces to an accounting system. Essentially, customers file insurance claims, which contain one or many services. Some portion of the service dollar amount will be paid by the insurer. This amount must then have an account number attached to it, and be transmitted to the accounting system. From a data quality perspective, applicable amounts per service reconcile from service to accounting data.

Selected columns in the first table, services, are:

service_num

amt_allowed

discount_amt

copay_amt

paid_amt

Selected columns in the second table, accounting_data, are:

serv_num

entry_type

amount

During the initial exploration of the data, the challenge is to understand which amounts from the services table correspond to which amounts in the accounting_data table. Thus, the pseudo-code will look like:

```
select * from services where rownum < N
foreach row
   foreach column
      print “SERVICE.this_column: data_val”
      select * from accounting_data where
       serv_num=this_service.service_num
      foreach row
         foreach column
            print “ACCOUNTING.this_column:
             data_val”
```

The results of this exploration suggest that services.paid_amt corresponds to accounting_data.amount, entry_types 1 and 2. Thus, our code becomes:

```
select service_num, paid_amt from services
 where rownum < N
foreach row
   select * from accounting_data where
    serv_num=this_service.service_num
   foreach row
      if (this_accounting_data.entry_type
       = 1 or 2)
         sum=sum+this_accounting_data.amount
      if (sum = this_service.paid_amt)
         success=success + 1
      else
         failure=failure + 1
         print “SERVICE:
          this_service.service_num,
          this_service.paid_amt,
          ACCOUNTING:  sum”
```

4. PERFORMANCE RESULTS

In this case study, the services table had 10.5 million rows, and the accounting_data table, 2.4 million rows.

Timings for retrieval of data from the synthetic join follow:

N	Time
10	2 seconds
100	3 seconds
1,000	26 seconds
10,000	2.3 minutes
100,000	8.3 minutes
548,000	61 minutes

A SQL join required 8 hours.

5. SUMMARY

The synthetic join technique yields more information more quickly than standard SQL techniques. It readily supports the evolution of analysis from exploration to hypothesis testing to formal reporting. It facilitates expansion and contraction of the data set as the algorithm and reporting formats evolve. It collects summary and detail reporting in a single pass of the data set. It integrates performance benchmarks into the analysis, supporting high performance data analysis in VLDB environments.