The *IDUG Solutions Journal*: Letter from the Executive Editor
September 2003 - Volume 10 Number 2

The Need for Speed:

Speeding up DB2 UDB Using Sampling

by Peter J. Haas

Introduction

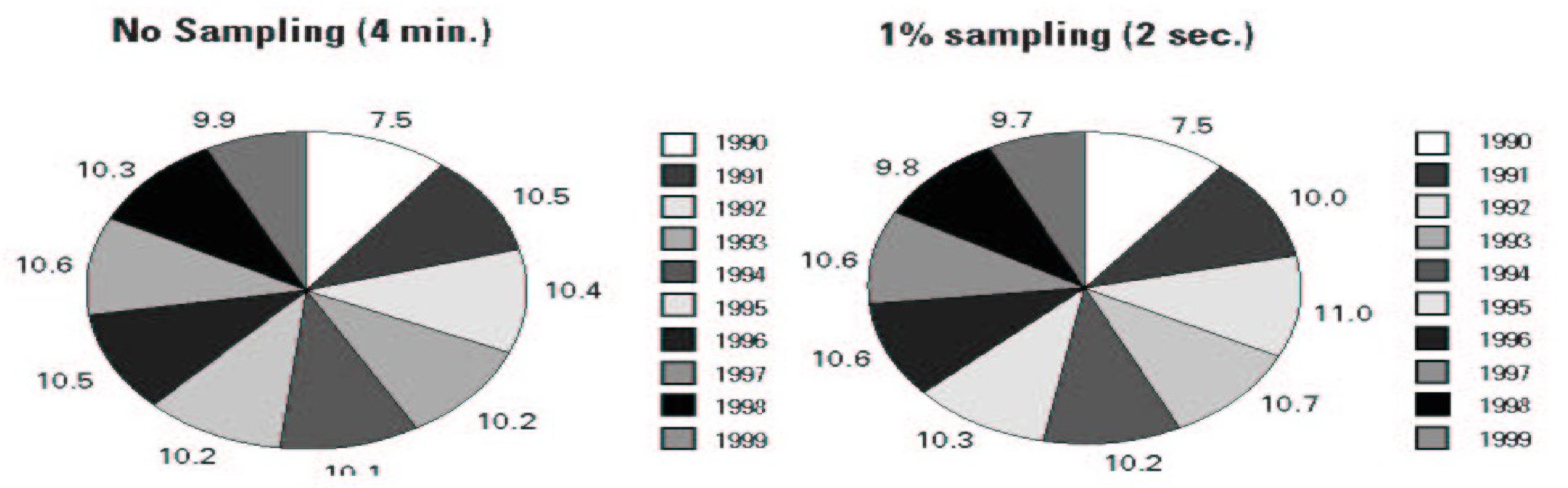
The vast amount of data in warehouses and on the web poses a major challenge for users who want to run BI or OLAP queries, execute advanced analytical, mining, and statistical algorithms or interactively explore their data. Most of these tasks simply do not scale to the hundreds of terabytes of data often found in modern repositories. At the same time, users are demanding that decision support systems be increasingly fast, flexible, and responsive. Although increases in CPU and disk speeds are helpful in dealing with massive data, hardware improvements alone do not suffice. Indeed, there is evidence that computer systems are becoming slower as the volume of online data is growing at a rate faster than Moore's law.

To help users address these increasingly difficult scalability problems, DB2 UDB will support random sampling in SQL queries, starting with Fixpack 2 of V8.1. Sampling techniques permit the computation of approximate query results-which often suffice in practice-in a fraction of the time required to compute an exact answer. For example, Figure 1 shows the result of executing the simple SQL aggregation query

```
SELECT SUM(sales) FROM transactions GROUP BY year
```

on the entire transactions table and on a 1% sample of the table. As can be seen, the results for the sampled table are almost indistinguishable from those for the entire table, while the required processing time is reduced by over two orders of magnitude.

Figure 1: Effect of sampling on an aggregation query



Besides application to aggregation queries as above, a sample can also be used for auditing purposes, interactive data exploration or as input to a mining or analysis application. In these latter settings, a sample can be viewed as a synopsis or compressed version of a set of data.

In the following sections, we describe how sampling will work in DB2 UDB, and we show how the power of sampling can be greatly enhanced by combining DB2's basic sampling functionality with the expressive power of SQL.

Sampling Syntax and Semantics

Any stored table appearing in the FROM clause of a SELECT query can have an appended sampling clause. The basic syntax of a sampling clause is as follows (illustrated here for a single table):

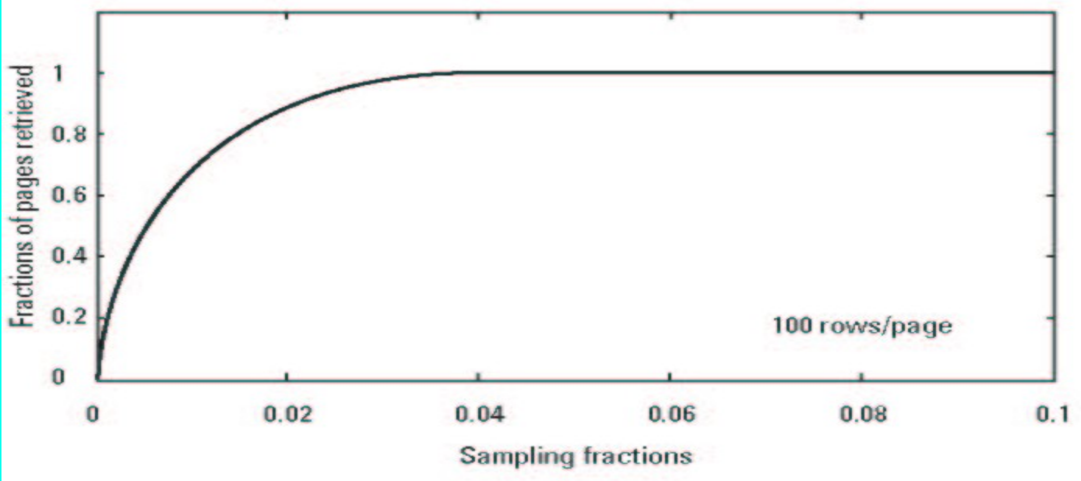
```
SELECT select_list
FROM table_name TABLESAMPLE sampling_method (P)
[ REPEATABLE (S) ]
```

Here TABLESAMPLE is a new SQL keyword that tells DB2 to process only a sample of rows from table_name rather than all of the rows, sampling_method specifies the method used to sample the rows, and P specifies the target percentage of rows to retrieve (between 0 and 100). The optional REPEATABLE clause, discussed below, is useful for debugging sampling queries. DB2's syntax is consistent with the ISO SQL-200n Change Proposal that has nominally been approved by ISO's Database Languages Working Group.

Although the proposed ISO syntax permits a sampling clause to be associated with any table reference, DB2 will initially restrict sampling to stored tables. Thus a materialized query table (MQT/AST) or global temporary table can be sampled, but sampling will not be allowed on logical views, nicknames (in federated databases), table expressions, table functions, or so forth.

DB2 supports the two sampling methods currently specified in the ISO proposal: BERNOULLI and SYSTEM. BERNOULLI sampling (with sampling percentage P) "flips a weighted coin" for each row individually, including that row in the sample with probability *P*/100 and excluding it with probability 1-*P*/100, independently of the other rows. For this reason the BERNOULLI sampling method is sometimes called *row-level* Bernoulli sampling. Although on average the sample contains P percent of the rows in the table, the actual sample size is random and hence may differ on subsequent executions of the query. Figure 2 illustrates this.

Figure 2: Minimum fraction of pages retrieved for BERNOULLI sampling as a function of fraction of rows sampled



When there is no index available, BERNOULLI sampling retrieves each row in the table, so that there is no I/O savings and sampling performance can be poor. When there is an index on one or more of the columns on the table, then DB2 performs the Bernoulli "coin flips" on the RIDs in the index leaf pages, thereby retrieving only those pages that contain at least one sampled row. However, even under the most efficient possible implementation of BERNOULLI sampling, a large fraction of the pages must be retrieved unless the sampling fraction is extremely low.

SYSTEM sampling permits the query optimizer to determine the most efficient manner in which to perform the sampling. In most cases, SYSTEM sampling applied to a table means that each page of the table is included in the sample with probability *P*/100 and excluded with probability 1-*P*/100. For each page that is included, all rows on that page qualify for the sample. This sampling method is called *page-level* Bernoulli sampling. SYSTEM sampling generally executes much faster than BERNOULLI sampling, since fewer data pages need to be retrieved. SYSTEM sampling can, however, yield less accurate estimates for aggregate functions, e.g., SUM(SALES), especially if there are many rows on each page or if the rows of the table are clustered on any columns referenced in the query. The optimizer may in certain circumstances decide that it is more efficient to perform SYSTEM sampling as if it were BERNOULLI sampling, for example, when a predicate can be applied by an index and is much more selective than the sampling rate *P*.

Semantically, sampling of a table occurs before any other query processing, such as applying predicates or performing joins. One can envision the original tables referenced in a query being initially replaced by temporary "reduced" tables containing sampled rows, and then normal query processing commencing on the reduced tables. (For performance reasons, actual processing may not occur in exactly this manner.) It follows, for example, that repeated accesses of a sampled table within the same query, such as in a nested-loop join or a correlated subquery, will see the same sample of that table within a single execution of that query.

Illustrative Examples

Suppose we wish to estimate the sum of sales over a set of transactions using a 2% SYSTEM sample. Then we can use the following query:

```
SELECT SUM(sales) / 0.02
FROM transactions TABLESAMPLE
SYSTEM(2);
```

Because we have computed the sum of sales over only about 2% of the rows in the table, we need to scale up our answer by a factor of 50 (i.e., 1/0.02) to estimate the sum over the entire table. (Such a scale-up would not have been needed had we used AVERAGE instead of SUM.) If we modifiy the query to look as follows:

```
SELECT SUM(sales) / 0.02
FROM transactions TABLESAMPLE
SYSTEM(2) REPEATABLE(3);
```

then we will take the exact same sample from transactions every time that we execute the query, and hence compute the same estimate every time (assuming that transactions is not modified between query executions). If we change the argument of REPEATABLE from 3 to 17, then we will also get the same estimate every time we run the modified query, but this latter estimate will differ from the estimate that we obtain using a REPEATABLE argument equal to 3.

We can also compute the join of two samples:

```
SELECT s.a, t.b
FROM s TABLESAMPLE SYSTEM(1), t
TABLESAMPLE SYSTEM(1)
WHERE s.key = t.fkey;
```

In general, the join of samples has very different statistical properties from the sample of a join, e.g., as computed by the following query:

```
CREATE USER TEMPORARY TABLESPACE;
DECLARE GLOBAL TEMPORARY TABLE r(a REAL, b REAL);
INSERT INTO SESSION.r
(SELECT s.a, t.b FROM s,t WHERE
s.key = t.fkey);
SELECT a, b FROM SESSION.r TABLESAMPLE
SYSTEM(0.01);
```

(We use a global temporary table for this query because such tables are inexpensive to create and modify; no entries are created in the system catalog and the table persists only for the duration of the database connection.)

Sampling can be used to obtain quick estimates of simple aggregates that are computed over joins, provided that the appropriate scaleup factor is used; e.g., a SUM must be scaled up by a factor of 1 / (product of sampling rates). Here is an example:

```
SELECT SUM(s.a * t.b) / (0.05 * 0.01)
FROM s TABLESAMPLE SYSTEM(5), t
TABLESAMPLE SYSTEM(1)
WHERE s.c > s.d;
```

The estimate computed by this query is *unbiased* for the true value in that if we execute the sample query over and over, then we obtain the correct answer on average.

The scope of DB2's basic sampling tools can be greatly enhanced using the expressive power of SQL, as shown by our final example. This query permits drilldown into a "*fuzzy*" data cube of sales data:

```
SELECT country, state, city, year, month,
  AVG(value) AS avg_sales,
FROM trans TABLESAMPLE
SYSTEM(:samp_rate), loc
WHERE trans.locid = loc.locid
GROUP BY ROLLUP(country, state, city),
ROLLUP(year, month)
```

```
HAVING COUNT(*) > 100
```

Note that the operations of sampling and application of predicates (including predicates implicit in a GROUP BY query) are interchangeable.

Conclusion

Sampling will be a powerful enhancement to DB2's capabilities, potentially speeding up query processing by orders of magnitude. DB2 will support almost the entire proposed ISO standard; we expect that additional sampling methods and other enhancements will be added both to the standard and to DB2 over time. By judiciously combining DB2's native sampling support with the SQL language, the DB2 user can effectively apply BI, OLAP, mining, and analysis techniques even in the face of massive data volumes.

About the author

Peter Haas received a Ph.D. from Stanford University in 1986 and has been a Research Staff Member at IBM Almaden Research Center since 1987. In addition, he is currently a Consulting Associate Professor in the Management Science and Engineering Department at Stanford and an Associate Editor for the journal Operations Research. He has played a key role in both incorporating and exploiting sampling functionality in DB2 UDB, and has recently received an Outstanding Technical Achievement Award from IBM for this work. He has received awards from both ACM SIGMOD and the IBM Research Division for his research on online query processing, as well as a number of IBM Invention Achievement Awards. He is the author of over 50 conference publications, journal articles, and books.

International DB2 Users Group
401 N. Michigan Ave.
Chicago, IL 60611
(312) 321-6881