Derby/S: A DBMS for Sample-Based Query Answering

Anja Klein Rainer Gemulla Philipp Rösch Wolfgang Lehner

TU Dresden
Database Technology Group
dbgroup@mail.inf.tu-dresden.de

ABSTRACT

Although approximate query processing is a prominent way to cope with the requirements of data analysis applications, current database systems do not provide integrated and comprehensive support for these techniques. To improve this situation, we propose an SQL extension—called SQL/S—for approximate query answering using random samples, and present a prototypical implementation within the engine of the open-source database system Derby—called Derby/S. Our approach significantly reduces the required expert knowledge by enabling the definition of samples in a declarative way; the choice of the specific sampling scheme and its parametrization is left to the system. SQL/S introduces new DDL commands to easily define and administrate random samples subject to a given set of optimization criteria. Derby/S automatically takes care of sample maintenance if the underlying dataset changes. Finally, samples are transparently used during query processing, and error bounds are provided. Our extensions do not affect traditional queries and provide the means to integrate sampling as a first-class citizen into a DBMS.

1. INTRODUCTION

The sheer amount of data in today's ever-growing databases overstrains the processing resources currently available. For example, the usability of explorative data analysis and decision support applications grows significantly when answers are provided almost instantly—instead of letting users wait for hours. A solution to the problem is provided by approximate query answering techniques, with sampling being among the most popular and versatile strategies.

Database sampling has made immense progress during the past years, but to best of our knowledge, there is no DBMS which natively supports these techniques on a broader scale. The Derby/S system (an open-source deviate of the IBM Cloudscape DBMS) is the first to integrate approximate query processing into an existing DBMS. We extended the SQL language, thereby relieving the user of the need to learn a new query language. Moreover, there is a huge variety of database-related sampling schemes, but the decision on the optimal technology for a given setting is challenging. To overcome these limitations, the contributions of Derby/S are as follows:

- Derby/S provides an integrated language to administrate materialized samples of arbitrary database views.
- Derby/S keeps the samples up-to-date when base data evolves.
- Our query language seamlessly extends SQL. Derby/S transparently selects the best sample for a query.

SIGMOD 2006, June 27–29, 2006, Chicago, Illinois, USA.

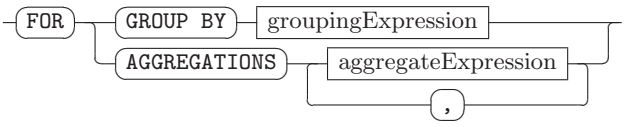

Figure 2: The optimization clause

2. SAMPLES AS DATABASE OBJECTS

Using state-of-the-art technology, a user has to query the data source, choose a sampling scheme, parametrize it, implement it, run it, propagate results to the DBMS and implement triggers to maintain the sample. Clearly, these steps require extensive knowledge of database technology and sampling theory, which is accessible to experts only. In contrast, our Derby/S prototype provides an easy and intuitive way to relieve the user of the need to perform any of the above steps. Given an arbitrary SQL statement, Derby/S automatically chooses the optimum sampling strategy, and takes the responsibility to compute and maintain the sample. The syntax of the `CREATE SAMPLE` command is shown in Figure 1. As every other database object, each sample is identified by a unique name. It is either bounded in size (`TUPLES`) or grows and shrinks with the size of the underlying data (`PERCENT`). Source data is defined using an SQL query, which basically underlies the same restrictions as materialized views [6] do.

Derby/S is able to optimize a sample for a given set of group-by operations and/or aggregate functions, thereby increasing precision for a specific class of queries (see Fig. 2). For example, the following statement creates a 10%-sample of the `lineitem` table of a TPC-H database, which computes monthly turnovers with improved precision:

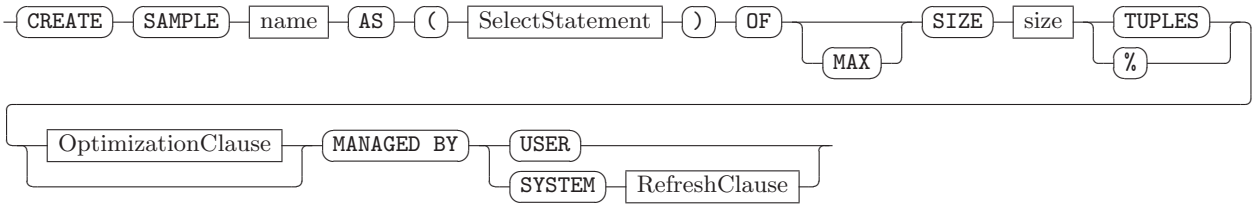

Figure 1: The `CREATE SAMPLE` statement

```
CREATE SAMPLE s_lineitem
AS (SELECT * FROM lineitem)
OF SIZE 10 %
FOR GROUP BY month
FOR AGGREGATIONS sum(turnover) ...;
```

During the execution of the `CREATE SAMPLE` command, empty sample tables are created and the metadata catalog of the DBMS is updated. Sampling is not conducted until a `REFRESH` statement is executed (see Sec. 3). If no optimization information is given, Derby/S computes a uniform sample of the base data. Otherwise, the basic challenge is to produce samples which are tailored, but not restricted, to the optimization criteria. Several studies, e.g. [8], have shown that knowledge of the application significantly enhances the precision of the estimates derived from the sample. For example, [3, 2] proposed sampling schemes for group-by operations and [4, 8] introduced techniques for dealing with high-variance datasets.

Derby/S is able to store and exploit additional synopses for each of its samples. The `MAX` keyword (see Fig. 1) ensures that the total memory of the sample and the synopses never exceeds the given space constraint. In this case, Derby/S carefully weighs the amount of space allocated to the sample (for arbitrary queries) and additional synopses (for "monthly-turnover" queries). Otherwise, Derby/S produces additional synopses as needed and without any space limitations. The motivation behind this procedure is that the query response time for arbitrary queries depends on the sample size, while "monthly-turnover" queries use additional data. This approach is related to dynamic sample selection [3].

3. SAMPLE MAINTENANCE

Each sample has to be maintained if the underlying data changes. However, maintenance incurs additional cost, thereby reducing the performance of transactional query processing. Derby/S supports a variety of strategies to balance sample maintenance cost and up-to-dateness: `IMMEDIATE` refresh updates the sample immediately after a modification of the base data, so that the sample is always up-to-date. Since this approach may be expensive, Derby/S additionally supports deferred maintenance [5], i.e., the sample is refreshed periodically or on demand. The latter is initiated by issuing a `REFRESH SAMPLE` command. This command is applicable (though not necessary) for other types of deferred refresh as well. The syntax of the refresh clause is shown in Fig. 3.

There are two options for the refresh procedure: `COMPLETE` refresh recomputes the sample from scratch, whereas `FAST` refresh tries to use as much of the existing synopsis as possible (thereby reducing maintenance cost). Note that complete recomputation in combination with immediate refresh is neither supported nor desired:

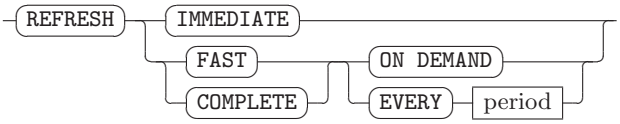

Figure 3: The `REFRESH` clause

	Incremental	Complete
Immediate	`IMMEDIATE`	—
Deferred	`FAST`	`COMPLETE`

4. APPROXIMATE QUERY ANSWERING

We extended the SQL DML in order to support approximate query processing. Our extension is independent from the actual synopses used for estimation, and applies to other approximation techniques as well. Traditionally, approximate query processing involves at least the following steps:

1. Decide on the optimal synopsis for the given query.
2. Implement an appropriate estimator.
3. Rewrite the query to use the chosen synopsis instead of base data.
4. Compute the query result.
5. Try to quantify the estimation error.

Again, Derby/S automatically executes the above steps. SQL/S is a seamless extension of the SQL query language, with a minimum amount of modifications. It is up to the user to decide whether an exact answer is required or an approximate result suffices.

4.1 Completeness and Exactness

A major contribution of SQL/S is the careful treatment of the type of approximation conducted by the underlying algorithm. Consider a dataset $R = \{(A, 10), (A, 20), (B, 20), (B, 30)\}$ and the following SQL query:

```
SELECT col1, AVG(col2) FROM r GROUP BY col1
```

Now, suppose that we want to compute an approximate answer to the given query. There are two different classes of error: On the one hand, the approximate answer may be incomplete, i.e., some of the required results are simply missing. One the other hand—even if the answer is complete—the aggregate of COL2 may be an estimate, thereby inducing an estimation error. Thus, we classify approximate results as shown in the following table:

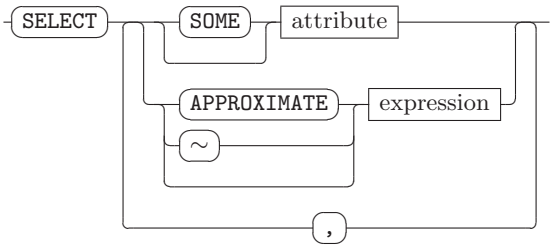

Figure 4: Extension of the SELECT statement

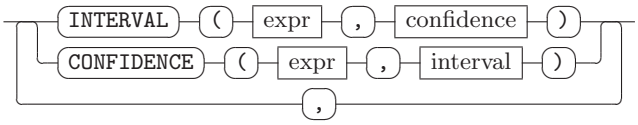

Figure 5: Providing error bounds

	Exact	Estimate
Complete	$(A, 15), (B, 25)$	$(A, 17), (B, 23)$
Incomplete	$(A, 15)$	$(B, 23)$

SQL/S allows the user to decide which types of error are acceptable. We introduced two additional keywords into the SQL query language (see Fig. 4): `SOME` and `APPROXIMATE` (abbr. $\sim$). The absence of the `SOME` keyword ensures completeness of the query result. Since the amount of completeness is difficult to quantify, Derby/S selects only synopses which are known to return the complete result in this case. The `APPROXIMATE` keyword allows Derby/S to estimate the value of an arbitrary aggregate. Thus, in SQL/S, the above query may be written as follows:

```
SELECT col1, ~AVG(col2) FROM r GROUP BY col1
```

Note that the tilde is the only difference between the exact and the approximate query. Both queries reference the base data only, that is, approximate queries do not refer to the underlying synopses. In fact, from a user's point of view, it is of no interest which estimation method is used by the processing engine.

4.2 Error Bounds

Derby/S provides customized error bounds. Such error bounds are a key requirement for any approximate answer. Typically, most estimators provide error guarantees in the form of confidence intervals, i.e., the estimate lies within $\pm I$ of the exact answer with probability p. To compute the error bound, either I or p have to be specified (p defaults to 95%). We introduce two additional functions—one which computes I given p and one which computes p given I (see Fig. 5). Our current implementation makes use of large-sample confidence intervals [7].

In analogy, Derby/S allows to bound the error in advance using the very same techniques (see Fig. 6). Thus, if we want the approximate result to be within 5% of the exact result with high probability, the example query changes to:

```
SELECT col1, ~AVG(col2) INTERVAL=5%
FROM r GROUP BY col1
```

5. RELATED WORK

The AQUA project [1] pioneered the use of synopsis data

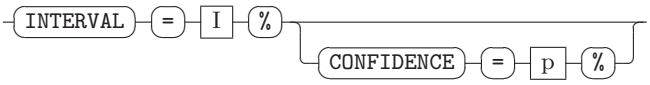

Figure 6: Bounding the error

structures for approximate query answering. Like Derby/S, AQUA transparently rewrites incoming SQL queries, but the system provides no means to distinguish approximate and exact queries. Furthermore, the error bounds produced by AQUA are not customizable, i.e., users have to compute them by themselves in most cases. An additional drawback is AQUA's lack of support for synopsis creation, maintenance and management.

The SQL standard proposes a `TABLESAMPLE` clause, which unfortunately only covers the creation of samples of a single table. Neither estimation, nor error bounds, nor any advanced sampling schemes are supported therein.

6. DETAILS OF THE DEMO

We plan to interactively build up a well-designed sample structure for the TPC-H benchmark by using the `CREATE SAMPLE` command. We will show how maintenance of these synopses is conducted on-the-fly, and quantify the performance impact of the different strategies on transactional query processing. We will provide insight into the internal data structures used by Derby/S, e.g., its metadata management. We will demonstrate how samples are selected dynamically, and report on the quality of the produced estimates as well as on the increase in performance. By providing some examples, we will explain how Derby/S decides on the best sample to use and how estimates are computed.

Acknowledgment. The authors like to thank F. Beyer, M. Dumat and B. Jäcksch for implementing large parts of the prototype.

7. REFERENCES

[1] S. Acharya, P. B. Gibbons, and V. Poosala. Aqua: A fast decision support system using approximate query answers. In *VLDB*, pages 754–757, 1999.

[2] S. Acharya, P. B. Gibbons, and V. Poosala. Congressional sampling for approximate answering of group-by-queries. In *SIGMOD*, pages 487–498, 2000.

[3] B. Babcock, S. Chaudhuri, and G. Das. Dynamic sample selection for approximate query processing. In *SIGMOD*, pages 539–550, 2003.

[4] S. Chaudhuri, G. Das, and M. Datar. Overcoming limitations of sampling for aggregation queries. In *ICDE*, pages 534–544, 2001.

[5] R. Gemulla and W. Lehner. Deferred maintenance of disk-based random samples. In *EDBT*, pages 423–441, 2006.

[6] A. Gupta and I. S. Mumick. Maintenance of materialized views: Problems, techniques, and applications. *IEEE Data Eng. Bull.*, 18(2):3–18, 1995.

[7] P. J. Haas. Large-sample and deterministic confidence intervals for online aggregation. In *SSDBM*, pages 51–63, 1997.

[8] C. Jermaine. Robust estimation with sampling and approximate pre-aggregation. In *VLDB*, pages 886–897, 2003.