Join Synopses for Approximate Query Answering

Swarup Acharya Phillip B. Gibbons Viswanath Poosala Sridhar Ramaswamy*

{swarup,gibbons,poosala}@research.bell-labs.com, sridhar@epiphany.com
Information Sciences Research Center
Bell Laboratories
600 Mountain Avenue
Murray Hill NJ 07974

Abstract

In large data warehousing environments, it is often advantageous to provide fast, approximate answers to complex aggregate queries based on statistical summaries of the full data. In this paper, we demonstrate the difficulty of providing good approximate answers for join-queries using only statistics (in particular, samples) from the base relations. We propose *join synopses* as an effective solution for this problem and show how precomputing just *one join synopsis* for each relation suffices to significantly improve the quality of approximate answers for arbitrary queries with foreign key joins. We present optimal strategies for allocating the available space among the various join synopses when the query work load is known and identify heuristics for the common case when the work load is not known. We also present efficient algorithms for incrementally maintaining join synopses in the presence of updates to the base relations. Our extensive set of experiments on the TPC-D benchmark database show the effectiveness of join synopses and various other techniques proposed in this paper.

1 Introduction

Traditional query processing has focused solely on providing exact answers to queries, in a manner that seeks to minimize response time and maximize throughput. However, in large data recording and warehousing environments, providing an exact answer to a complex query can take minutes, or even hours, due to the amount of computation and disk I/O required.

There are a number of scenarios in which an exact answer may not be required, and a user may prefer a fast, approximate answer. For example, during some drill-down query sequences in ad-hoc data mining, initial queries in the sequence are used solely to determine what the interesting queries are [HHW97]. An approximate answer can also provide feedback on how well-posed a query is. Moreover, it can provide a tentative answer to a query when the base data is unavailable. Another example is when the query requests numerical answers, and the full precision of the exact answer is not needed, e.g., a total, average, or percentage for which only the first few digits of precision are of interest (such as the leading few digits of a total in the millions, or the nearest percentile of a percentage). Finally, techniques for fast approximate answers can also be used in a more traditional role within the query optimizer to estimate plan costs; such an application demands very fast response times but not exact answers.

Motivated by the above reasons, we study the issue of providing approximate answers to queries in this paper. Our goal is to provide an estimated response in orders of magnitude less time than the time to compute an exact answer, by avoiding or minimizing the number of accesses to the base data. Our work is tailored to the typical data warehousing environments, which have a few "central" fact tables connected via foreign-key relationships to multiple dimension tables. In such a scenario, it is very common to pose aggregate queries that join the fact table with the dimension tables on their respective foreign-keys. For example, 13 of the 17 queries in the TPC-D benchmark involve foreign-key joins. In this paper, we present novel techniques for providing approximate answers to such queries [1].

We show, both theoretically and empirically, that schemes for providing approximate join aggregates that rely on using random samples of base relations alone suffer from serious disadvantages (Section 3). Instead, we propose the use of *precomputed samples of a small set of distinguished joins* —referred to as *join synopses*—in order to compute approximate join aggregates (Section 4). Our key contribution is to show that for queries with foreign-key joins, *it is possible to provide good quality approximate join aggregates using a very small number of join synopses.* An important issue arising out of the use of several sets of statistics is the careful allocation of a limited amount of space among

*This work was done while the author was at Bell Labs. Current affiliation is Epiphany Inc., 2300 Geng Road, Suite 200, Palo Alto CA 94303.

[1] We use the term "approximate join aggregates" to refer to such answers.

them. When a query workload characterization is available, we show how to design an optimal allocation for join synopses that minimizes the overall error in the approximate answers computed. We discuss heuristic allocation strategies that work well when the workload is not known (Section 5). A critical issue in approximate query answering is that of providing confidence bounds for the answers. Such bounds give the user valuable feedback on how reliable an answer is. In addition to discussing how traditional methods for providing confidence bounds (for example, based on Hoeffding bounds or the Central Limit Theorem [Haa97]) apply to join synopses, we propose a novel empirical technique for computing confidence bounds based on extracting subsamples from samples (Section 6). We also show how join synopses can be incrementally maintained in the presence of updates (Section 7). Finally, we present the results of a detailed experimental study on the performance of the techniques we propose. Using the TPC-D benchmark, we show the advantages of join synopses over samples of base relations in computing approximate join aggregates with good confidence bounds. We also show that join synopses can be maintained efficiently and with minimal overheads (Section 8).

Previous work related to approximate query answering is presented in Section 9. Due to limited space, we omit the proofs of all theoretical results from this paper and refer the reader to a full version of this paper for all the details [AGPR99b].

The research in this paper was conducted as part of our efforts to develop an efficient decision support system based on approximate query answering, called Aqua [GMP97a]. A brief introduction of Aqua is presented in the next section.

2 The Aqua System

The goal of Aqua is to improve response times for queries by avoiding accesses to the original data altogether. Instead, Aqua maintains smaller-sized statistical summaries, called *synopses*, on the warehouse and uses them to answer queries. Currently, these statistics take the form of various types of samples and histograms on the data in the data warehouse. A key feature of Aqua is that the system provides probabilistic error/confidence bounds on the answer, based on the Hoeffding and Chebychev formulas [AGPR99b]. Currently, the system handles arbitrarily complex SQL queries applying aggregate operations (`avg, sum, count, etc.`) over the data in the warehouse.

Aqua has three key components:

- **Statistics Collection:** This component of Aqua is responsible for collecting all the synopses which Aqua uses to answer queries posed by the user. In this paper, we propose new techniques to augment this component to accurately answer multi-way foreign key join queries (Section 4).

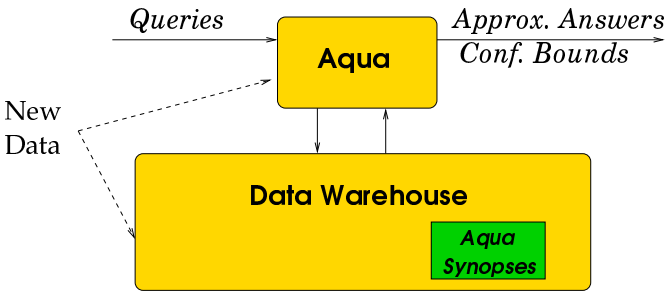


Figure 1: The Aqua architecture.

- **Query Rewriting:** Aqua achieves response time speed ups by rewriting queries posed by the user to instead use the synopses. This module is responsible for parsing the input SQL query and generating an appropriately translated query. Additionally, the rewriting involves appropriate scaling of certain operators to take into account the size of the synopses vis a vis the original data.

- **Maintenance:** This component is responsible for keeping the synopses up to date in the presence of updates to the underlying data. In Section 7, we extend our prior work and propose novel techniques for incrementally maintaining join synopses.

The high-level architecture of the Aqua system is shown in Figure 1. It is designed as a software tool that can sit atop any commercial DBMS (currently, Oracle) managing a data warehouse. Initially, Aqua takes as an input from the warehouse administrator the space available for synopses and if available, hints on important query and data characteristics.[2] This information is then used by the statistics collector to precompute a suitable set of synopses on the data, which are stored as regular relations in the DBMS.

Figure 2 shows a screen shot of the current web user interface for Aqua. It shows the actual and approximate answers along with error bounds for a 4-way join query. The good quality of the approximate answers is in part due to the use of join synopses to answer foreign key join queries. The figure also shows the times taken to generate the two answers. Further details on Aqua are available in [GMP97a, AGPR99b, AGPR99a].

In the rest of the paper, we motivate the need for join synopses and present optimal allocation schemes and maintenance techniques for them.

3 The Problem with Joins

A natural set of synopses for an approximate query engine would include uniform random samples of each base relation in the database. We refer to these as *base samples*. The use of base samples to estimate the output of a join of

[2] Work is also in progress to automatically extract this information from a query workload and adapt the statistics dynamically.

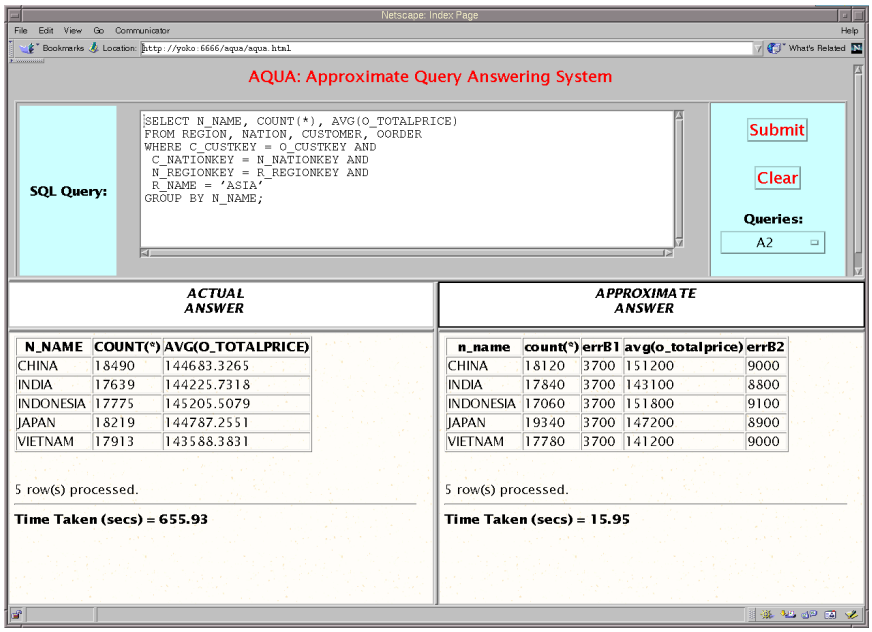


Figure 2: Aqua User Interface

two or more relations, however, can produce a poor quality approximation. This is for the following two reasons:

1. **Non-Uniform Result Sample:** In general, the join of two uniform random base samples is *not a uniform random sample of the output of the join*. In most cases, this non-uniformity significantly degrades the accuracy of the answer and the confidence bounds.

2. **Small Join Result Sizes:** The join of two random samples typically has very few tuples, even when the actual join selectivity is fairly high. This can lead to both inaccurate answers and very poor confidence bounds since they critically depend on the query result size.

Consider the first problem. In order for the join of the base samples to be a uniform random sample of the actual join, the probability of any two joined tuples to be in the former should be the same as their probability in the latter. (This is a necessary, but not a sufficient condition.) We will use a simple counter example to show that this is not always the case.

Consider the (equality) join of two relations R and S on an attribute X. The distribution of X values in the two relations are given in Figure 3. The edges connect joining tuples. Consider joining base samples from R and S. Assume that each tuple is selected for a base sample with probability $1/r$. From Figure 3, we see that $a1$ and $a2$ are in the join if and only if both a tuples are selected from R and the one a tuple is selected from S. This occurs with probability $1/r^3$, since there are three tuples that must be selected. On the other hand, $a1$ and $b1$ are in the join if and only if the *four* tuples incident to these edges are selected. This occurs with

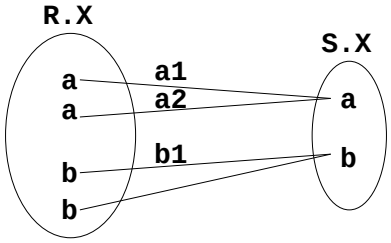


Figure 3: Join of samples is *not* a sample of joins

probability only $1/r^4$. This contrasts with the fact that in a uniform random sample of the actual join, the probability that both $a1$ and $a2$ are selected *equals* the probability that both $a1$ and $b1$ are selected.

We now highlight the second problem of small output sizes. Consider two relations, A and B, and base samples comprising of 1% of each relation. The size of the foreign key join between A and B is equal to the size of A. However the expected size of the join of the base samples is .01% of the size of A, since for each tuple in A, there is only one tuple in B that joins with it, and that tuple is in the sample for B with only a 1% probability. In general, consider a k-way foreign key join and k base samples each comprising $1/r$ of the tuples in their respective base relations. Then the expected size of the join of the base samples is $1/r^k$ of the size of the actual join. In fact the best known confidence interval bounds for approximate join aggregates based on base samples are quite pessimistic [Haa97].

Thus, it is in general impossible to produce good quality approximate answers using samples on the base relations alone, a fact that we further demonstrate in our experiments. Since nearly all queries in the warehousing context involve

complex queries with large number of (foreign-key) joins, it is critical to solve this problem. In the next section we provide a solution for this problem.

4 Join Synopses

In this section we present a practical and effective solution for producing approximate join aggregates of good quality. At a high level, we propose to precompute samples of join results, making quality answers possible even on complex joins. A naive way to precompute such samples is to execute all possible join queries of interest and collect samples of their results. However, this is not feasible since it is too expensive to compute and maintain. Our main contribution is to show that by computing samples of the results of a *small set of distinguished joins*, we can obtain random samples of all possible joins in the schema. We refer to samples of these distinguished joins as *join synopses*. Our technique works for the star and snowflake schemas typically found in data warehousing [Sch97]. More precisely, we propose a solution for queries with only *foreign key joins*, which are defined as follows.

Definition 4.1 Foreign Key Join: *A 2-way join $r_1 \bowtie r_2$, $r_1 \neq r_2$, is a* foreign key join *if the join attribute is a foreign key in r_1 (i.e., a key in r_2). For $k \geq 3$, a k-way join is a k-*way foreign key join *if there is an ordering $r_1, r_2, \ldots, r_k$ of the relations being joined that satisfies the following property: for $i = 2, 3, \ldots, k$, $s_{i-1} \bowtie r_i$ is a 2-way foreign key join, where s_{i-1} is the relation obtained by joining $r_1, r_2, \ldots, r_{i-1}$.*

In order to develop this solution, we model the database schema by a graph with a vertex for each base relation and a directed edge from a vertex u to a vertex $v \neq u$ if there are one or more attributes in u's relation that constitute a foreign key for v's relation. The edge is labeled with the foreign key. Figure 4 shows the corresponding graph for the TPC-D schema. We restrict our attention in this work to acyclic (schema-)graphs, which are common in warehousing environments.

From the figure, it can be seen that $C \bowtie N$ and $L \bowtie O \bowtie C$ are 2-way and 3-way foreign key joins respectively. Note that two 2-way foreign key joins involving a common relation does not imply that a 3-way join among them would also be a foreign key join. For example, though $C \bowtie N$ and $S \bowtie N$ are foreign key joins, $C \bowtie N \bowtie S$ is not a foreign key join, by Definition 4.1.

The key result we prove is that there is a one-one correspondence between a tuple in a relation r and a tuple in the output of *any* foreign key join involving r and the relations corresponding to one or more of its descendants in the graph. This provides us with the technical tool for join synopses: a sample S_r of a relation r can be used to produce another relation $\mathcal{J}(S_r)$—called a join synopsis of r—that can be used to provide random samples of *any join involving r and one or more of its descendants*.

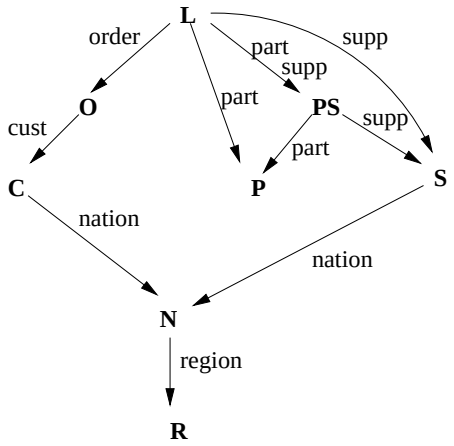


Figure 4: Directed graph for the TPC-D schema.

We now move to the technical development of the results. Consider the directed acyclic graph G corresponding to the schema of a database. We show two key lemmas about the properties of such graphs.

Lemma 4.1 *The subgraph of G on the k nodes in any k-way foreign key join must be a connected subgraph with a single root node.*

We denote the relation corresponding to the root node as the *source* relation for the k-way foreign key join.

Lemma 4.2 *There is a 1-1 correspondence between tuples in a relation r_1 and tuples in any k-way foreign key join with source relation r_1.*

From Lemma 4.1, we have that each node can be the source relation only for k-way foreign key joins involving its descendants in G. For each relation r, there is some *maximum foreign key join* (i.e., having the largest number of relations) with r as the source relation. For example, in Figure 4, $C \bowtie N \bowtie R$ is the maximum foreign key join with source relation C.

Definition 4.2 Join synopses: *For each node u in G, corresponding to a relation r_1, define $\mathcal{J}(u)$ to be the output of the maximum foreign key join $r_1 \bowtie r_2 \bowtie \cdots \bowtie r_\kappa$ with source r_1. (If u has no descendants in G, then $\kappa = 1$ and $\mathcal{J}(u) = r_1$.) Let S_u be a uniform random sample of r_1. Define a* join synopsis, *$\mathcal{J}(S_u)$, to be the output of $S_u \bowtie r_2 \bowtie \cdots \bowtie r_\kappa$. The* join synopses *of a schema consists of $\mathcal{J}(S_u)$ for all u in G.* ∎

To emphasize the sampling nature of join synopses, we will sometimes refer to them as *join samples*.

For example, in the TPC-D schema, the join synopsis for R is simply a sample of R whereas for C it is the join of N, R, and a sample of C. Next, we show that the join synopsis of a relation can be used to obtain a uniform random sample for a large set of queries.

Theorem 4.3 *Let $r_1 \bowtie \cdots \bowtie r_k$, $k \geq 2$, be an arbitrary k-way foreign key join, with source relation r_1. Let u be the node in G corresponding to r_1, and let S_u be a uniform random sample of r_1. Let A be the set of attributes in $r_1, \ldots, r_k$. Then, the following are true:*

- *$\mathcal{J}(S_u)$ is a uniform random sample of $\mathcal{J}(u)$, with $|S_u|$ tuples. (From Lemma 4.2.)*
- *$r_1 \bowtie \cdots \bowtie r_k = \pi_A \mathcal{J}(u)$, i.e., the projection of $\mathcal{J}(u)$ on the attributes in $r_1, \ldots, r_k$. (Trivially true from the definition of $\mathcal{J}(u)$ given in the above definition.)*
- *$\pi_A \mathcal{J}(S_u)$ is a uniform random sample of $r_1 \bowtie \cdots \bowtie r_k$ $(= \pi_A \mathcal{J}(u))$, with $|S_u|$ tuples. (Follows from the above two statements.)*

Thus we can extract from our synopsis a uniform random sample of the output of any k-way foreign key join, $k \geq 2$. For example, the join synopsis on L in the TPC-D schema can be used to obtain a sample of any join involving L (which is true for most queries in the benchmark). The next lemma shows that a single join synopsis can be used for a large number of *distinct joins*, especially for the star-like schemas common in data warehouses. Here, two joins are *distinct* if they do not join the same set of relations.

Lemma 4.4 *From a single join synopsis for a node whose maximum foreign key join has κ relations, we can extract a uniform random sample of the output of between $\kappa - 1$ and $2^{\kappa-1} - 1$ distinct foreign key joins.*

Note that since Lemma 4.2 fails to apply in general for any relation other than the source relation, the joining tuples in any relation r other than the source relation will not in general be a uniform random sample of r. Thus distinct join synopses are needed for each node/relation.

Since tuples in join synopses are the results of multi-way joins, a possible concern is that they will be too large because they have many columns. To reduce the columns stored for tuples in join synopses, we can eliminate redundant columns (for example, join columns) and only store columns of interest. Small relations can be stored in their entirety, rather than as part of join synopses. To further reduce the space required for join synopses, we can renormalize the tuples in a join synopsis into its constituent relations and remove duplicates. To the extent that foreign keys are many-to-one, this will reduce the space, although the key will then be replicated. Of course, with renormalization, when a tuple in S_u is deleted, one has to delete any joining tuples in the constituent relations as well. This can be done either immediately or in a lazy fashion in a batch. The following lemma places a bound on the size of a renormalized join synopsis.

Lemma 4.5 *For any node u whose maximum foreign key join is a κ-way join, the number of tuples in its renormalized join synopsis $\mathcal{J}(S_u)$ is at most $\kappa|S_u|$.*

Example 4.1 Consider the TPC-D schema in Figure 4. In the TPC-D benchmark database, the relations N and R, corresponding to Nation and Region, have 25 and 5 tuples in them, respectively. Therefore, we can store them in their entirety without considering any samples for them. We can therefore remove them from the graph. We are left with the nodes L, PS, O, C, P, and S. For each of these relations, the system needs to store a join synopsis corresponding to the join for which the relation is a source.

We now briefly highlight the space overhead for join synopses in TPC-D. The number of relations in the *maximum foreign key join* corresponding to each of these nodes (denoted by the letter κ above) is $6, 3, 2, 1, 1$, and 1 for L, PS, O, C, P and S respectively. Let us now make two simplifying assumptions: (1) the size of the tuples in each base relation is the same; and (2) the number of tuples, n, allocated to each of the join synopses is the same. By Lemma 4.5, the total number of tuples in the synopsis is at most $|N| + |R| + \sum_u \kappa_u |S_u| = 30 + 14n$. Thus we can obtain, *for every possible join* in the TPC-D schema, a uniform random sample of 1% of *each join result*, from a collection of join synopses that in total use less than 15% of the space needed for the original database! Note also that we can further reduce the size of the join synopses by taking advantage of the fact that many foreign keys are many-to-one. ∎

To summarize, we have shown that it is possible to create compact join synopses of a schema with foreign key joins such that we can obtain a random samples of any join in the schema. In the next section, we present a detailed analysis of deciding the size of the join synopses taking into account tuple size, query frequency, etc.

5 Allocation

In this section, we present optimal strategies for allocating the available space among the various join synopses when certain properties of the query work load are known and identify heuristics for the common case when such properties are not known.

5.1 Optimal strategies

We consider the following high-level characterization of a set, S, of queries with selects, aggregates, group bys and foreign key joins. For each relation, R_i, we determine the fraction, f_i, of the queries in S for which R_i is either the source relation in a foreign key join or the sole relation in a query without joins. For example, for the 17 queries in the TPC-D benchmark, L is the source or sole relation for 14 queries and PS is the source or sole relation for 3 queries, and hence the fraction f_i equals $14/17$ for L, equals $3/17$ for PS, and equals zero for all other relations.

We seek to select join synopsis (join sample) sizes so as to minimize the average relative error over a collection of aggregate queries, based on this characterization of the set of queries. This can be done analytically by minimizing the

average relative error bounds (i.e., confidence intervals) over the collection. Although this seems to imply that the optimal sample size allocation is specific to the type of error bounds used, we will show that a large class of error bounds share a common property that we will exploit for this purpose. Namely, we observe that the error bounds for COUNT, SUM, and AVG based on the commonly-used Hoeffding bounds and/or Chebychev bounds, including the new approaches discussed in Section 6, *all* share the property that the error bounds are inversely proportional to $\sqrt{n}$, where n is the number of tuples in the (join) sample. (Details on these bounds are discussed in Section 6.)

Thus the average relative error bound over the queries is proportional to

$$\sum_i \frac{f_i}{\sqrt{n_i}}, \tag{1}$$

where n_i is the number of tuples allocated to the join sample for source relation R_i.

Our goal is to select the n_i so as to minimize Equation 1 for a given bound, N, on the total memory allotted for join synopses. For each source relation R_i, let s_i be the size of a single join synopsis tuple for i. Then we require $\sum_i n_i s_i \leq N$. We show that the optimal allocation selects n_i to be proportional to $(f_i/s_i)^{2/3}$:

Theorem 5.1 *Given N, and f_i and s_i for all relations R_i, taking*

$$n_i = N' \cdot \left(\frac{f_i}{s_i}\right)^{2/3}$$

where $N' = N/(\sum_j {f_j}^{2/3} {s_j}^{1/3})$, minimizes Equation 1 subject to $\sum_i n_i s_i \leq N$.

Note that the above analysis has ignored predicate selectivities. We observe that the relative error bounds for COUNT, SUM, and AVG based on the commonly-used Hoeffding bounds and/or Chebychev bounds, including our new approaches, are either proportional to $1/\sqrt{qn}$ or proportional to $1/q\sqrt{n}$, where q is the selectivity. In the absence of a characterization of the query work load in terms of predicate selectivities, we assume that the selectivities are independent of the relations. (Incorporating a selectivity characterization can readily be done, although the analysis is more detailed.) Under this assumption, our analysis above holds good for any mix of selectivities.

Finally, note that the sample sizes can be adapted to a changing query load by maintaining the frequencies f_i, and reallocating among the join samples as the frequencies change.

5.2 Heuristic strategies

We next consider three strategies for allocating join synopses that can be used in the absence of query work load information. These can be used as starting points for the adaptive procedure proposed above.

- *EqJoin* divides up the space allotted, N, equally amongst the relations. Each relation devotes all its allocated space to join synopses. (For relations with no descendants in the schema, this equates to a sample of the base relation.)
- *CubeJoin* divides up the space amongst the relations in proportion to the cube root of their join synopsis tuple sizes. Each relation devotes all its allocated space to join synopses.
- *PropJoin* divides up the space amongst the relations in proportion to their join synopsis tuple sizes. Each relation devotes all its allocated space to join synopses, and hence each join synopsis has the same number of tuples.

Thus for *EqJoin*, *CubeJoin*, and *PropJoin*, the number of tuples for a join synopsis with tuple size s_i is inversely proportional to s_i, $s_i^{2/3}$, and 1, respectively. When the error bounds are inversely proportional to $\sqrt{n}$, *CubeJoin* minimizes the average relative error bounds when all frequencies f_i are assumed to be equal (Theorem 5.1), and *PropJoin* minimizes the *maximum* error bound when all frequencies f_i are nonzero.

These allocation strategies using join samples can be compared against similar strategies that use only base samples: (a) *EqBase* is like *EqJoin* on base samples, i.e., it devotes all its allocated space to samples of the base relations; (b) *CubeBase* is like *CubeJoin* on base samples; and (c) *PropBase* is like *PropJoin* on base samples.

The experimental results in Section 8 quantify the advantage of the join samples strategies over the base samples strategies for representative queries.

6 Improved Accuracy Measures

A critical issue in approximate query answering of aggregation queries is that of providing confidence bounds for the answers. There are several popular methods for deriving confidence bounds for approximate answers obtained from samples; these are based on Central Limit Theorem (CLT) bounds, Chebychev bounds, and/or Hoeffding bounds. An important advantage of using join synopses is that queries with foreign key joins can be treated as queries without joins (i.e., single-table queries). Known confidence bounds for single-table queries are much faster to compute and much more accurate than the confidence bounds for multi-table queries (see, e.g., [Haa96]).

In the full paper [AGPR99b][3], we summarize methods for single-table queries and then present a detailed analysis that demonstrates the *precise trade-offs* among these methods, as well as a method based on subsampling, which we describe next.

Consider the following estimation approach, which we call *chunking*:

[3] See also [GM99a] for extensions and further analysis.

1. Partition the sampled tuples in a join synopsis into k subsets (subsamples), which we call "chunks", and for each chunk j, compute an estimator, e_j, based on the sample points in the chunk.

2. Report an estimate and a bound based on the e_j.

Previous work (see, e.g., [AMS96]) has shown that confidence bounds for an estimator can be improved by repeating an estimation procedure many times, and then applying a chunking-like approach. We extend this previous work as follows.

- Within the general chunking framework, we propose and explore (analytically and experimentally) a number of alternative procedures for reporting an estimate and an error bound based on the chunks, including varying the number of chunks. We consider two possible choices for reporting an overall estimate e: taking the average of the e_j and taking the median of the e_j.

- Whereas previous work on taking the median has been asymptotic in nature, we show the precise (i.e., non-asymptotic) trade-offs for when the guaranteed bounds for the median improve upon the bounds with no chunking, and for the optimal number of chunks to use for confidence probabilities of practical interest.

- We propose and explore the use of the chunk estimators in generating *empirical* error bounds, as described next.

Using chunking for empirical error bounds: Often, confidence bounds derived analytically are overly pessimistic: the estimated answer is closer to the exact answer more often than indicated by the analytical bound. A common approach taken to verify this is to conduct multiple trials of an experiment on various data sets. However, this is not entirely satisfactory, as the data sets of interest in some applications may not exhibit the good behavior of the data sets used in the study.

We propose chunking as a means to report on multiple experiments run on the *actual query and data*. Each subsample is its own experiment on the actual query and data, and there are various possibilities on how to report these results to the user. In the full paper [AGPR99b], we study the effectiveness of reporting a CLT bound using the sample variance of the chunk estimators, or alternatively, reporting the minimum and maximum of the chunk estimators. Other alternatives include reporting various quantiles of the chunk estimators. The feedback to the user is intuitively of the form: k independent experiments were run for your query, all (or say, 90%) of which fell within the range $[x, y]$, with the average (or median) being e. Our experiments confirm that these empirical bounds are a good compliment to traditional guaranteed bounds.

7 Maintenance of Join Synopses

In this section, we focus on the maintenance of join synopses when the underlying base relations are being updated (we consider both insertions and deletions. The techniques we propose are simple to implement and require only infrequent access to the base relations.

Our algorithm for maintaining a join synopsis $\mathcal{J}(S_u)$ for each u is as follows. Let p_u be the current probability for including a newly arriving tuple for relation u in the random sample S_u. (This probability is typically the ratio of the number of tuples in S_u to the number of tuples in u.) On an insert of a new tuple τ into a base relation corresponding to a node u in G, we do the following. Let $u \bowtie r_2 \bowtie \cdots \bowtie r_\kappa$ be the maximum foreign key join with source u. (1) We add τ to S_u with probability p_u. (2) If τ is added to S_u, we add to $\mathcal{J}(S_u)$ the tuple $\{\tau\} \bowtie r_2 \bowtie \cdots \bowtie r_\kappa$. This can be computed by performing at most $\kappa - 1$ look-ups to the base data, one each in $r_2, \ldots, r_\kappa$. (For any key already in $\mathcal{J}(S_u)$, the look-ups for it or any of its "descendants" are not needed.) (3) If τ is added to S_u and S_u exceeds its target size, then select uniformly at random a tuple τ' to evict from S_u. Remove the tuple in $\mathcal{J}(S_u)$ corresponding to τ'.

On a delete of a tuple τ from u, we first determine if τ is in S_u. If τ is in S_u, we delete it from S_u, and remove the tuple in $\mathcal{J}(S_u)$ corresponding to τ. As in [GMP97b], if the sample becomes too small due to many deletions to the sample, we repopulate the sample by rescanning relation u.

Note that this algorithm only performs look-ups to the base data with (small) probability p_u. Also, when a tuple is inserted into a base relation u, we never update join synopses for any ancestors of u. Such updates would be costly, since these operations would be performed for every insert and for each ancestor of u. Instead, we rely on the integrity constraints to avoid these costly updates.

Theorem 7.1 *The above algorithm properly maintains all S_u as uniform random samples of u and properly maintains all join synopses $\mathcal{J}(S_u)$.*

We assume that updates may be applied in a "batch" mode. In such environments, join synopses can be kept effectively up-to-date at all times without any concurrency bottleneck. In an online environment in which updates and queries intermix, an approximate answering system can not afford to maintain up-to-date synopses that require examining every tuple (e.g., to find the minimum and maximum value of an attribute), without creating a concurrency bottleneck. In such environments, maintenance is performed only periodically. Approximate answers depending on synopses that require examining every tuple would not take into account the most recent trends in the data (i.e., those occurring since maintenance was last performed), and hence the accuracy guarantees would be weakened. Note that the techniques described in this section can also be used to compute a join synopsis from scratch in limited storage, in one scan

Table 1: Features of relations in the TPC-D benchmark.

Table Name	# of Columns	Cardinality
Customer	8	$45K$
Lineitem	16	$1800K$
Nation	4	25
Order	9	$450K$
Part	9	$60K$
Partsupplier	5	$240K$
Region	3	5
Supplier	7	$3K$

of the base data followed by indexed look-ups on a small fraction of the keys.

8 Experimental Evaluation

In this section, we present the results of an experimental evaluation of the techniques proposed in this paper. Using data from the TPC-D benchmark, we show the effectiveness of our approach in providing highly accurate answers for approximate join aggregates.

We begin this section by describing the experimental testbed. We then present results from two classes of experiments—*accuracy* experiments and *maintenance* experiments. In the accuracy experiments, we compare the accuracy of techniques based on join synopses to that of techniques based on base samples. The two key parameters in this study are query selectivity and total space allocated to precomputed summaries (summary size). We first compare the techniques for a fixed selectivity and varying summary size and then compare the techniques for a fixed summary size and varying selectivities. In the maintenance experiments, we study the cost of keeping the join synopses up to date in the presence of insertions/deletions to the underlying data. We show that join synopses can be maintained with very little overhead even when updates significantly change the characteristics of the underlying data.

8.1 Experimental testbed

We ran the tests on the TPC-D decision support benchmark. We used a scale factor of 0.3 for generating our test data. This results in a database that is approximately 300 megabytes. Table 1 summarizes the important features of the eight relations in the TPC-D database.

Our experiments were run on a lightly loaded 296MHz UltraSPARC-II machine having 256 megabytes of memory and running Solaris 5.6. All data was kept on a local disk with a streaming throughput of about 5MB/second.

Query model: The query used for the accuracy experiments is based on query $Q5$ in the TPC-D benchmark and is an aggregate that is computed on the join of `Lineitem`, `Customer`, `Order`, `Supplier`, `Nation` and `Region`. Of the six relations involved in the join, the `Nation` and `Region` relations are sampled in their entirety by Aqua because of their low cardinality. This effectively reduces the query to a (still complex) four-way join query.

The SQL statement for the query is given in Figure 5. It computes the average price of products delivered by suppliers in a nation to customers who are in the same nation. The select conditions take three input parameters — `region`, `startdate` and `enddate`. These restrict suppliers and customers to be within a specific region and focus on business conducted within a specific time interval. In the following experiments, we will vary one or more of these parameters to study the performance for various query selectivities.

In this study, we have focused only on the hard problem of computing approximate aggregates on multi-way joins. Of course, our sampling results extend to the simple case of single table aggregates. Thus, due to space constraints, we do not show any results for the single table case. Besides, those results qualitatively mirror the ones presented in the context of *online aggregation* for single table aggregates [HHW97].

Space allocation schemes: Recall from Section 5 that we proposed a number of schemes for allocating a given amount of summary space to enable approximate query answering. For the case where certain characterizations of the query mix were known, we presented optimal allocation strategies to minimize overall error. However, for this experimental study, we assume the more realistic scenario where this information is unavailable. Thus, we study the six space allocation schemes proposed in Section 5.2, namely, *EquiBase*, *CubeBase*, *PropBase*, *EquiJoin*, *CubeJoin* and *PropJoin*. For the purposes of this experiment, we focus on the four major relations used in $\mathcal{Q}_a$, and allocate base samples and join synopses only on those relations. Therefore, the base sampling schemes divide up the summary space among samples of `Lineitem`, `Customer`, `Order`, and `Supplier`, whereas the join synopses schemes distribute the summary space to join synopses for `Lineitem` (which includes columns from `Customer`, `Order`, and `Supplier`), for `Customer` (which includes columns from `Order`), for `Order` (whose join synopsis is just a base sample), and for `Supplier` (whose join synopsis is also a base sample).

Recall that *PropJoin* gives an equal number of tuples to the various samples whereas *EquiJoin* divides the space equally. Thus, among the various schemes, the source relation in the 4-way join in $\mathcal{Q}_a$, `Lineitem`, is allocated the most space by *PropJoin* since it has the largest tuple and the least space by *EquiJoin*, while *CubeJoin* allocates space in between these two extremes. Likewise, among the base sample schemes, *PropBase* allocates the most space to the base sample of `Lineitem`, followed in order by *CubeBase* and *EquiBase*. To avoid clutter in the graphs that follow, we do not plot *CubeJoin* and *CubeBase* and only show numbers for the other four schemes. They cover the entire range of performance for the different schemes.

```
select avg(l_extendedprice) from customer, order, lineitem, supplier, nation, region
    where c_custkey = o_custkey and o_orderkey = l_orderkey and l_suppkey = s_suppkey
    and c_nationkey = s_nationkey and s_nationkey = n_nationkey and n_regionkey = r_regionkey
    and r_name = [region] and o_orderdate >= DATE [startdate] and o_orderdate < DATE [enddate]
```

Figure 5: Query $\mathcal{Q}_a$ used for accuracy experiments. Based on Query $Q5$ from the TPC-D benchmark.

The experiments also study the sensitivity of the various schemes to the total summary size allocated (parameter *SummarySize* in the figures). *SummarySize* is varied from 0.1% to 3% of the total database size, varying the actual summary size in bytes from 420 KBytes to 12.5 MBytes.

8.2 Experimental results

In this section, we present the results of the experimental study. The first two experiments cover the accuracy studies and the final experiment addresses the problem of maintaining join synopses during updates to the underlying data. It should be noted that the graphs presented in this section are a small subset of the results that we obtained. These results have been chosen because they demonstrate the different aspects of approximate query answering using join synopses.

8.2.1 Experiment 1: Join synopsis accuracy

In this experiment, we study the accuracy of the four space allocation schemes for different values of summary size (parameter *SummarySize*) and for different query selectivities. We compare the actual answer of running query $\mathcal{Q}_a$ (Figure 5) on the full TPC-D database against the approximate answers obtained from the different schemes.

Consider Figure 6(a). It plots the average extended price computed by the different schemes for varying summary sizes. The actual answer is shown as a straight line parallel to the x-axis. Following the specification for query $Q5$ in the TPC-D benchmark, the `region` parameter is set to `ASIA` and the selection predicate on the `o_orderdate` column to the range [`1/1/94,1/1/95`].

Consider the two schemes that use only samples of the base relations, *EquiBase* and *PropBase*. Figure 6(a) shows that these schemes produce answers consistently only when the summary size exceeds 1.5% of the database. (For lower sample sizes, the join of the base samples is completely empty!) In fact, it is not until 2% summary size that the approximate answer produced by them comes close to the actual answer. In fact, on the left end of the graph (for smaller summary sizes), these scheme either produce no output at all (e.g., *PropBase* for 1.25% synopsis size), or produce answers that are significantly different from the real answer (with errors close to 100% in some cases).

The schemes based on join synopses, *EquiJoin* and *PropJoin*, on the other hand, not only produce output over the entire range of summary size studied but are also fairly accurate in estimating the correct answer. Even for a summary size of 0.1% (420 Kbytes) shared among all the four join synopses, the results from both the schemes are within 14% of the actual answer! Moreover, the variation in the answers is lower than the variation in the answers from base sampling schemes. The difference between the two types of allocation schemes is further highlighted in Table 2, which shows the number of tuples in the join output for the four schemes. In most cases, the schemes based on join synopses produce at least an order of magnitude more number of tuples than the base sampling schemes do. As expected, *PropJoin* is the most accurate since it assigns the most space to `Lineitem`, the root of the 4-way join.

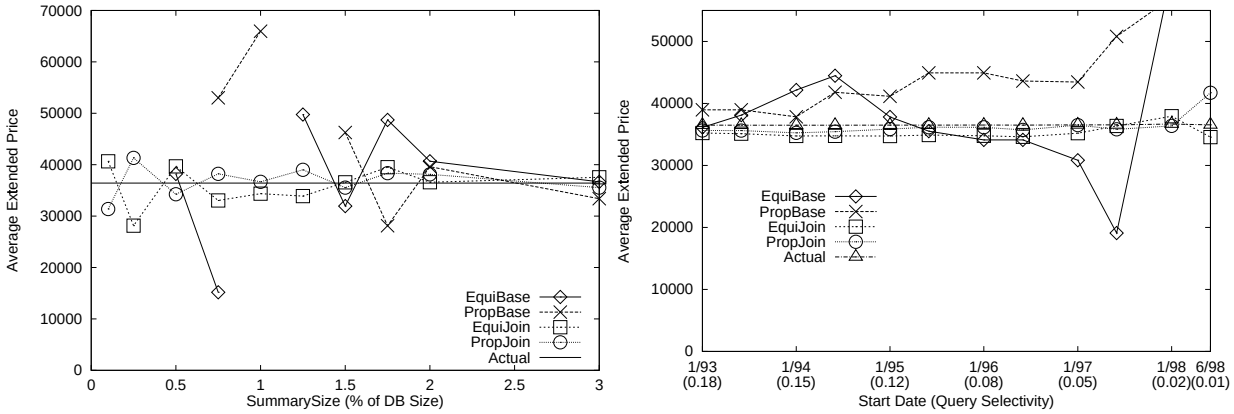
Figure 6(b) studies the sensitivity of the four allocation schemes for varying selectivities, with the summary size set to 1.5% of the database size. We change the selectivity of query $\mathcal{Q}_a$ by changing the date range in the selection condition on the `o_orderdate` attribute. To control the selectivity, we fixed the parameter `enddate` to `1/1/99`, the tail end of the date range in the TPC-D specification. We varied the `startdate` parameter from `1/1/93` to `6/1/98` in steps of six months. The `startdate`s are shown on the x-axis with the corresponding query selectivity given in brackets below.

Selectivity and summary size have a similar effect on the performance of the base sampling schemes. While the answers returned by the *EquiBase* and *PropBase* techniques are reasonably close to the actual answer when the selectivity is high (left end of the x-axis), the answers fluctuate dramatically as the selectivity decreases. As expected, the join synopsis schemes, *EquiJoin* and *PropJoin*, stay close to the actual answer over the entire range deviating only slightly when the selectivity is down to 1% on the right end of the graph.

These graphs demonstrate the advantages of schemes based on join synopses over base sampling schemes for approximate join aggregates. Even with a summary size of only 0.1%, join synopses are able to provide fairly accurate aggregate answers.

8.2.2 Experiment 2: Query execution timing

Figure 7 plots the time taken by the various strategies to execute the query (the y-axis is in logscale). The time to execute the actual query is 122 seconds and is shown as a straight line near the top of the figure. As expected, the response times increase with increasing summary size. However, for all the sizes studied, the execution time for the query using join synopses is two orders of magnitude smaller! (The times using base samples are more than an



(a) Date Interval = [1/1/94,1/1/95] (b) *SummarySize* = 1.5%

Figure 6: Behavior of join synopsis and base sample allocation strategies for different (a) summary size values and (b) for different query selectivities.

Table 2: Output Size for the various allocation schemes.

SummarySize	Base Samples		Join Synopses	
	EquiBase	*PropBase*	*EquiJoin*	*PropJoin*
0.1%	0	0	6	25
1%	0	2	56	142
1.5%	12	4	104	228
2%	38	44	131	300
3%	38	108	195	453

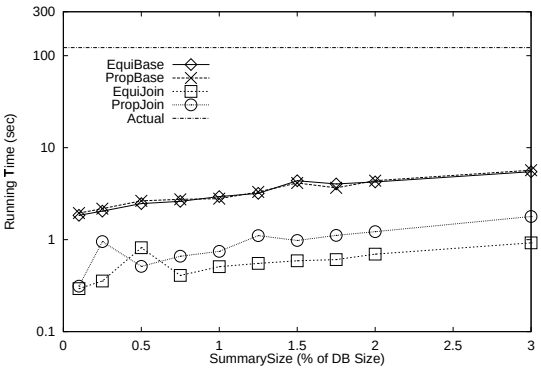


Figure 7: Query execution time for various schemes.

order of magnitude smaller than those computing the actual answer.)

This experiment demonstrates that it is possible to use join synopses to obtain extremely fast approximate answers with minimal loss in accuracy. This is good evidence that applications such as decision support and data warehousing, which can often tolerate marginal loss in result accuracy, can benefit tremendously from the faster responses of approximate query answering systems.

8.2.3 Experiment 3: Join synopsis maintenance

In this section, we show experimental results demonstrating that join synopses can be maintained with very minimal overhead. Such join synopses can give very good approximate answers even when updates significantly change the nature of the underlying data. We base this section on a join between the `Lineitem` and `Order` tables. The query used retrieves the average quantity of tuples from the `Lineitem` table that have a particular value for the `o_orderstatus` column. The SQL statement for the query is given in Figure 8.

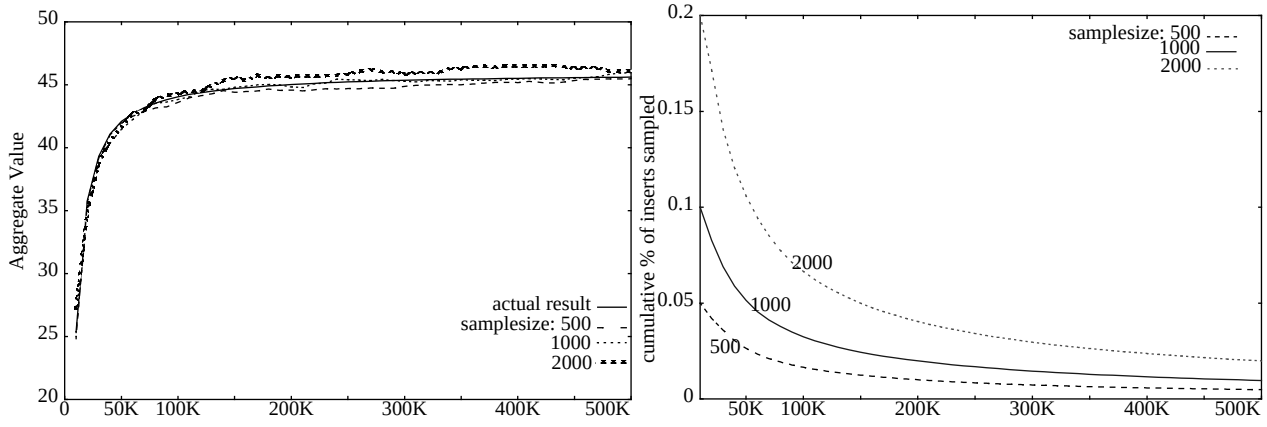
We consider the maintenance of a join synopsis for `Lineitem` as tuples are inserted into the `Lineitem` table, using the algorithm of Section 7. Note that insertions into other tables in the schema can safely be ignored in maintaining the `Lineitem` join synopsis. Figure 9(a) plots the aggregate values computed from join synopses of different sizes. Even for extremely small sizes, the join synopsis is able to track the actual aggregate value quite closely despite significant changes in the data distribution. Figure 9(b) shows that maintenance of join synopses is very inexpensive, by plotting the average fraction of the new `Lineitem` tuples that are actually inserted into the join synopsis. In accordance with the algorithm of Section 7, we go to the base data only when a tuple is inserted into the join synopsis. It is clear from the figure that this number is a small fraction of the total number of tuples inserted. (For example, when maintaining a sample of 1000 tuples and processing $500,000$ inserts, we go to the base data only 4822 times.)

8.2.4 Summary of experiments

The experimental results in this section empirically demonstrate the validity of the techniques proposed in this paper. The results show that join synopses can be used to compute approximate join aggregates extremely quickly, and that the performance of join synopses is superior to that of base sam-

```
select avg(l_quantity) from lineitem, order
    where l_orderkey = o_orderkey and o_orderstatus = F
```

Figure 8: Join synopsis maintenance query $\mathcal{Q}_m$.



(a) Result accuracy (b) Maintenance cost

Figure 9: Maintenance of join synopses for $500,000$ updates to the `Lineitem` table. (a) Accuracy of aggregate values computed from join synopses of various sizes (b) Cost of online maintenance.

pling schemes. Moreover, the results also show that join synopses can be maintained inexpensively during updates.

In the full paper, we also present experimental results demonstrating that our empirical error bounds are a good complement to traditional guaranteed bounds on approximate answers.

9 Related Work

Statistical techniques have been applied in databases for more than two decades now, primarily inside a query optimizer for selectivity estimation [SAC+79]. However, the application of statistical techniques to approximate query answering has started receiving attention only very recently. Below, we describe the work on approximate query answering and the work on general statistical techniques applied in databases.

Approximate query answering: Hellerstein *et al* proposed a framework for approximate answers of aggregation queries called *online aggregation* [HHW97], in which the base data is scanned in random order at query time and the approximate answer is continuously updated as the scan proceeds. Unlike Aqua, this work involves accessing original data at query time, thus being more costly, but at the same time, this approach provides an option to get the fully accurate answer gradually and it is not affected by database updates. However, the problems with join queries discussed in this paper also apply to online aggregation – basically, a large fraction of the data needs to be processed before the errors become tolerable. Other systems support limited online aggregation features; e.g., the Red Brick system supports running COUNT, AVG, and SUM (see [HHW97]). Since the scan order used to produce these aggregations is not random, the accuracy can be quite poor. In the APPROXIMATE query processor, developed by Vrbsky and Liu [VL93], an approximate answer to a set-valued query is any superset of the exact answer that is a subset of the cartesian product. The query processor uses various class hierarchies to iteratively fetch blocks relevant to the answer, producing tuples certain to be in the answer while narrowing the possible classes that contain the answer. Clearly, this work is quite different from the statistical approach taken by us and by Hellerstein *et al.*

Statistical techniques: The three major classes of techniques used are *sampling* (e.g., [HÖT88, LNS90, HNS94, LN95, HNSS95, GGMS96]), *histograms* (e.g., [Koo80, PIHS96, Poo97, APR99]), and parametric modeling (e.g., [CR94]). A survey of various statistical techniques is given in the paper by Barbará *et al* [BDF+97]. Gibbons and Matias present a framework for studying synopsis data structures for massive data sets [GM99b] and introduced two sampling-based synopses, *concise samples and counting samples*, that can be used to obtain larger samples for the same space and to improve approximate query answers for hot list queries [GM98]. Maintenance algorithms exist for samples [OR92, GMP97b, GM98] and histograms [GMP97b]. However, these maintenance techniques are applicable only to "base" statistics and not to the join synopses presented in this paper.

10 Conclusions

In this paper, we have focused on the important problem of computing approximate answers to aggregates computed on multi-way joins. For data warehousing environments

with schemas that involve only foreign-key joins, we have proposed join synopses as a solution to this problem. We have shown that schemes based on join synopses provide better performance than schemes based on base samples for computing approximate join aggregates. Further, we have also shown that join synopses can be maintained efficiently during updates to the underlying data. Finally, we have explored the use of empirical confidence bounds for approximate answers and have shown that they are a good complement to traditional guaranteed bounds.

Approximate query answering is becoming increasingly essential in data warehousing and other applications. Hence, it is important to eliminate any fundamental problems that limit its applicability to complex queries. This paper identifies one such problem and presents a complete solution to it. However, many other problems remain. These include accurately approximating answers to group-by, rank and set-valued queries. We are currently addressing these issues as part of the Aqua project.

Acknowledgements

Yossi Matias was one of the co-initiators of the Aqua project. Additional contributors to Aqua include Torsten Suel and S. Muthukrishnan. We thank Yossi for his contributions to the results in Section 6, and Yair Bartal for his work on the proof of Theorem 5.1. We also thank the anonymous referees for their comments.

References

[AGPR99a] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy. The Aqua approximate query answering system. In *Proc. ACM SIGMOD International Conf. on Management of Data*, June 1999. Demonstration paper.

[AGPR99b] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy. Join synopses for approximate query answering. Technical report, Bell Laboratories, Murray Hill, New Jersey, 1999. Full version of the paper appearing in SIGMOD'99.

[AMS96] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. In *Proc. 28th ACM Symp. on the Theory of Computing*, pages 20–29, May 1996. Full version to appear in JCSS special issue for STOC'96.

[APR99] S. Acharya, V. Poosala, and S. Ramaswamy. Selectivity estimation in spatial databases. In *Proc. ACM SIGMOD International Conf. on Management of Data*, June 1999.

[BDF+97] D. Barbará, W. DuMouchel, C. Faloutsos, P. J. Haas, J. M. Hellerstein, Y. Ioannidis, H. V. Jagadish, T. Johnson, R. Ng, V. Poosala, K. A. Ross, and K. C. Sevcik. The New Jersey data reduction report. *Bulletin of the Technical Committee on Data Engineering*, 20(4):3–45, 1997.

[CR94] C. M. Chen and N. Roussopoulos. Adaptive selectivity estimation using query feedback. In *Proc. ACM SIGMOD International Conf. on Management of Data*, pages 161–172, May 1994.

[GGMS96] S. Ganguly, P. B. Gibbons, Y. Matias, and A. Silberschatz. Bifocal sampling for skew-resistant join size estimation. In *Proc. ACM SIGMOD International Conf. on Management of Data*, pages 271–281, June 1996.

[GM98] P. B. Gibbons and Y. Matias. New sampling-based summary statistics for improving approximate query answers. In *Proc. ACM SIGMOD International Conf. on Management of Data*, pages 331–342, June 1998.

[GM99a] P. B. Gibbons and Y. Matias. Selecting estimation procedures and bounds for approximate answering of aggregation queries. Technical report, Bell Laboratories, Murray Hill, New Jersey, 1999.

[GM99b] P. B. Gibbons and Y. Matias. Synopsis data structures for massive data sets. *DIMACS: Series in Discrete Mathematics and Theoretical Computer Science*, 1999. To appear. Available as Bell Labs tech. rep., Sept. 1998, and at http://www.bell-labs.com/~pbgibbons/.

[GMP97a] P. B. Gibbons, Y. Matias, and V. Poosala. Aqua project white paper. Technical report, Bell Laboratories, Murray Hill, New Jersey, December 1997.

[GMP97b] P. B. Gibbons, Y. Matias, and V. Poosala. Fast incremental maintenance of approximate histograms. In *Proc. 23rd International Conf. on Very Large Data Bases*, pages 466–475, August 1997.

[Haa96] P. J. Haas. Hoeffding inequalities for join-selectivity estimation and online aggregation. Technical Report RJ 10040, IBM Almaden Research Center, San Jose, CA, 1996.

[Haa97] P. J. Haas. Large-sample and deterministic confidence intervals for online aggregation. In *Proc. 9th International Conf. on Scientific and Statistical Database Management*, August 1997.

[HHW97] J. M. Hellerstein, P. J. Haas, and H. J. Wang. Online aggregation. In *Proc. ACM SIGMOD International Conf. on Management of Data*, pages 171–182, May 1997.

[HNS94] P. J. Haas, J. F. Naughton, and A. N. Swami. On the relative cost of sampling for join selectivity estimation. In *Proc. 13th ACM Symp. on Principles of Database Systems*, pages 14–24, May 1994.

[HNSS95] P. J. Haas, J. F. Naughton, S. Seshadri, and L. Stokes. Sampling-based estimation of the number of distinct values of an attribute. In *Proc. 21st International Conf. on Very Large Data Bases*, pages 311–322, September 1995.

[HÖT88] W.-C. Hou, G. Özsoyoğlu, and B. K. Taneja. Statistical estimators for relational algebra expressions. In *Proc. 7th ACM Symp. on Principles of Database Systems*, pages 276–287, March 1988.

[Koo80] R. P. Kooi. *The Optimization of Queries in Relational Databases.* PhD thesis, Case Western Reserve University, September 1980.

[LN95] R. J. Lipton and J. F. Naughton. Query size estimation by adaptive sampling. *J. Computer and System Sciences*, 51(1):18–25, 1995.

[LNS90] R. J. Lipton, J. F. Naughton, and D. A. Schneider. Practical selectivity estimation through adaptive sampling. In *Proc. ACM SIGMOD International Conf. on Management of Data*, pages 1–12, May 1990.

[OR92] F. Olken and D. Rotem. Maintenance of materialized views of sampling queries. In *Proc. 8th IEEE International Conf. on Data Engineering*, pages 632–641, February 1992.

[PIHS96] V. Poosala, Y. E. Ioannidis, P. J. Haas, and E. J. Shekita. Improved histograms for selectivity estimation of range predicates. In *Proc. ACM SIGMOD International Conf. on Management of Data*, pages 294–305, June 1996.

[Poo97] V. Poosala. *Histogram-based estimation techniques in databases.* PhD thesis, Univ. of Wisconsin-Madison, 1997.

[SAC+79] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. T. Price. Access path selection in a relational database management system. In *Proc. ACM SIGMOD International Conf. on Management of Data*, pages 23–34, June 1979.

[Sch97] D. Schneider. The ins & outs (and everything in between) of data warehousing. Tutorial in the *23rd International Conf. on Very Large Data Bases*, August 1997.

[VL93] S. V. Vrbsky and J. W. S. Liu. Approximate—a query processor that produces monotonically improving approximate answers. *IEEE Trans. on Knowledge and Data Engineering*, 5(6):1056–1068, 1993.