

View-Size Estimation in Self-Organizing Databases

Kyoung-Hwa Kim and Rada Chirkova
Computer Science Department
North Carolina State University
Raleigh, NC 27695, USA
{kkim3,rychirko}@ncsu.edu

Abstract

Data-intensive systems routinely use derived data, such as indexes or materialized views, to improve query-evaluation performance. In this context, the problem of designing derived data is as follows: Given a database and a set of queries, return definitions of derived data that, when precomputed and stored in the database, would reduce the evaluation costs of the queries. Designing materialized views and indexes is an important part of automated query-performance tuning in data-management systems that experience changes over time, where a system addresses the performance requirements of current frequent and important queries by periodically reconsidering and rematerializing the stored derived data. In this paper we focus on the accuracy of estimating the sizes of views that are considered for materialization. We analyze and experimentally evaluate several size-estimation techniques and suggest guidelines for using the techniques depending on system parameters. We describe experimental results obtained in our prototype self-organizing database system QPET. Our guidelines are also applicable to estimating the sizes of intermediate results in query optimization.

1. INTRODUCTION

Derived data, such as indexes or materialized views, are routinely used in data-intensive systems to improve query-evaluation performance. In this context, the problem of *designing derived data* can be stated as follows: Given a database, a set of queries on the data stored in the database, and a set of constraints on derived data (e.g., a limit on the amount of storage available on disk), return definitions of derived data that, when precomputed and stored (i.e., *materialized*) in the database, would satisfy the constraints and reduce the

evaluation costs of the queries. Automated design of materialized views and indexes to answer queries is an important component of *automated query-performance tuning*, where a system addresses the performance requirements of *current* important queries by periodically redesigning the stored derived data. For this reason, developing techniques for designing derived data to improve query-answering performance is a recognized research direction in self-administering data-intensive systems [1, 14, 27].

We are maintaining and extending a prototype of an extensible self-organizing relational data-management system for Query-Performance Enhancement by Tuning (QPET) [9]. QPET monitors incoming user queries to determine the current workload of frequent and important queries, and periodically invokes its view- and index-design component to respond to the changes in the prevalent query workload.¹ In the process of designing useful materialized views for a given workload of frequent and important queries, see Figure 1, QPET first generates *definitions* of candidate views; all view definitions are expressed as queries on the stored data. It then determines whether the candidate-view definitions are *competitive*, that is, whether the answers to those view definitions could contribute significantly to improving the performance of the workload queries. If the answer is yes, then the answers to the competitive views should be precomputed and stored (that is, materialized) in the database. Those materialized views can then be used, possibly in combination with the original stored data, to answer in real time user queries posed on the database, see Figure 2.

Determining in QPET whether candidate views are competitive and thus should be materialized for the given query workload is the task of an extended query optimizer based on [7]. The decisions of the optimizer are based on estimates of the sizes of the materialized

¹For simplicity, in this paper we consider processing in QPET of only views, rather than indexes.

answers to the candidate-view definitions. (In most databases, the typically large volume of stored data would render prohibitive the time costs of obtaining the exact sizes of the materialized views by computing the answers to the definitions of candidate views.) Using view-size estimates gives the optimizer a way to efficiently estimate the costs of answering the workload queries using the candidate views, as well as the I/O costs and the amount of disk space required to store the materialized answers to the views. The optimizer can obtain the estimates using query-size-estimation techniques, which are traditionally used in query optimization [16] to give the costs of intermediate results. (We can apply these techniques to view definitions because views are defined using queries.)

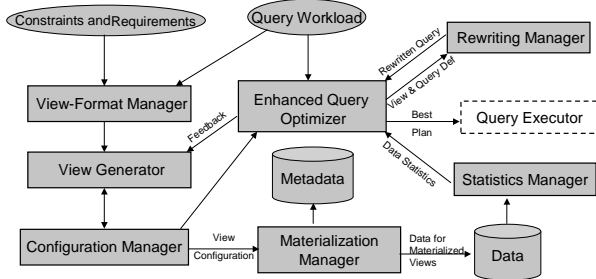


Figure 1. Designing derived data in QPET.

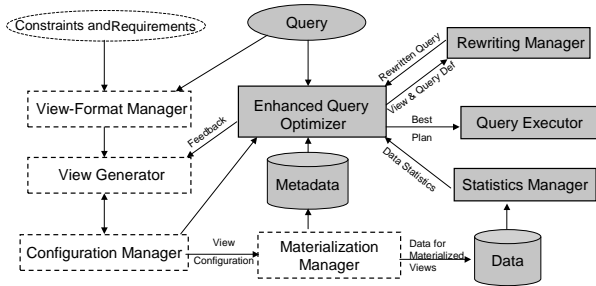


Figure 2. Using derived data in QPET.

To give reliable feedback to the view-design module, the optimizer in QPET should apply accurate query-size estimation techniques to estimate the sizes of materialized views. At the same time, it is well known [17] that the effect of even small errors can in some circumstances render most estimation schemes no better than random guesses. In this paper we focus on experimentally evaluating existing methods for estimating sizes of the materialized answers to given view definitions; we seek to develop guidelines for preferring some of those

methods to others depending on system parameters, and to use the guidelines in our QPET prototype for automated query-performance tuning, to improve the optimizer recommendations to the view-design module. The results we report in this paper are as follows.

Our contributions. Based on our analysis of the query-size estimation methods reported in the literature, we have selected for implementation those methods which have shown the most promise for size estimation of materialized views. We have experimentally evaluated the implemented methods using the TPC-H benchmark data [29] and a variety of TPC-H-style queries. Based on the experimental results, we report our recommendations on the most accurate and optimal methods, depending on the values of parameters we have determined for query definitions. Our recommendations can be used for view-size estimation in various view-design frameworks in relational databases, as well as for estimating the sizes of intermediate results in query optimization.

1.1. Related Work

A number of studies of query- or view-size estimation methods in database-management systems have been reported in the literature. The methods in the studies can be classified into three categories: parametric [6, 22, 26, 28], histogram based [15, 18, 19, 24], and sampling based [11, 12, 13, 20, 21, 31]. We now briefly describe each category.

Parametric methods are developed for a particular distribution assumption on the stored data, such as uniform, normal, Poisson, Zipf, and so on. Selinger et al. [6, 26], Makinouchi et al. [22] and Swami and Schiefer [28] have introduced parametric methods for the uniform-distribution assumption; Christodoulakis [10] describes methods that work for the normal and Pearson distributions for estimating the sizes of answers to select-join queries. The advantage of parametric methods is their time efficiency, as they typically require as inputs (in addition to the query) just the number of distinct values of each attribute and the number of tuples in each relation. The drawback of the methods in this category is that it is typically time consuming to verify whether the stored data satisfy the assumed data distribution.

A histogram is built by partitioning the stored data into mutually disjoint subsets called buckets, and by approximating the data frequency in each bucket. Several types of histograms have been proposed and evaluated, see, for instance, [18, 19, 24, 25]. Histogram-based methods can be subdivided into two categories. The first category contains methods based on equi-

width and equi-height histograms; the second category uses maxdiff, compressed, end-biased, or v-optimal histograms. Histogram-based methods for query-size estimation are easy to implement and do not require the knowledge of the data-distribution assumptions for the stored data. As a result, it is possible to obtain reliable estimates for queries on skewed data. On the other hand, some histogram-based methods rely on sorted data, and sorting data could be time consuming. Moreover, many histogram-based methods are more suited for numeric data values. In addition, it is complex to estimate the size of the result of a join of two relations using histograms, as the method involves merging the histograms for the participating attributes. Finally, multidimensional histograms for correlated attributes are difficult to construct.

Sampling is the basis of relatively more recent approaches to obtaining estimates for the sizes of answers to queries. Sampling-based methods estimate query sizes by collecting and processing random samples of the stored data. Lipton and Naughton [21] and Hass and Swami [11] have proposed adaptive sampling; Harangri et al [23] have introduced systematic sampling. Sampling methods are relatively simple to implement, do not require statistics on the stored data, and do not depend on underlying data types (in particular, they can be used on non-numeric data). At the same time, sampling can be time consuming, especially in case of systematic sampling, which relies on sorted data.

To the best of our knowledge, we are the first to propose guidelines for preferring some of query-size-estimation methods to others depending on system parameters, to improve the optimizer recommendations on view materialization in the context of automated query-performance tuning.

2. The Problem and Our Approach

In general, a SQL [5] query on relational data can be executed using operators such as full table scans, nested-loop or hash joins. A query optimizer estimates the costs of executing the query using various strategies, and then determines the lowest-cost strategy for the actual evaluation of the query. Research and commercial database-management systems (DBMS) (e.g., [2, 3, 4]) use the concept of a *materialized view* to improve query-evaluation performance on large databases. In a relational DBMS, a materialized view is a stored set of derived data, computed as an answer to a SQL query (which is the definition of the view) on the original stored data. Materialized views are precomputed and physically stored as relations and are thus available for query processing alongside the

original stored data.

The cost of using a materialized view M in evaluating a query Q on a database D is determined by the number of tuples in the answer to the view-definition query V on D , in combination with the I/O and CPU costs of accessing M when executing the query Q . In our setting, we assume that each materialized view is retrieved from disk via a sequential scan.² Therefore, the I/O and CPU costs of accessing a materialized view in query processing depend on the number of tuples in the relation for the view. (Note that given the schema of a relation and the number of tuples in it, it is straightforward to compute the number of disk blocks required to store the relation.) Hence, our focus is obtaining maximal possible accuracy in estimating the number of tuples in materialized views; the estimation can be done directly using query-size estimation methods. Thus, the goals of our project are to (1) determine query parameters whose values would influence the accuracy of query-size estimation methods, and to (2) formulate guidelines for choosing the most accurate view-size estimation methods for query evaluation in various environments determined by the values of the parameters.

In our experiments we consider select-project-join queries with equality and range (i.e., inequality) selection conditions, on both numeric and string data. The sizes of the answers to such queries depend on the selectivities of the selections and joins in the query definition. We say that a relation is of *size* τ if the relation has τ tuples. The *selection selectivity* of a selection operator σ on a relation R is the size of the portion of R that satisfies σ , divided by the total size of R . The *join selectivity* of a join operator \bowtie_C with join condition C on relations R and S is the size of the relation $R \bowtie_C S$, divided by the size of the cross product of R and S .

3. Method Analysis and Implementation

In this paper, we review the concepts used in some of the query-size estimation methods described in the literature, describe the promising methods for view-size estimation that we have selected for implementation, and report the guidelines that we have obtained using our experimental results with these methods. Our criteria for choosing promising methods for view-size estimation include the following:

- whether the method has high accuracy;
- whether the method can work without using underlying data-distribution assumptions;

²Evaluating queries using *indexed* materialized views is a direction of our current work.

- whether the method is straightforward to implement³;
- whether the method can be used to handle a broad spectrum of data types.

We have analyzed the advantages and disadvantages of the methods described in Section 1.1. Using our criteria, we have decided not to implement the parametric method, as its accuracy cannot be guaranteed when the data do not satisfy a fixed data-distribution assumption. We have implemented a method that uses maxdiff histograms, as the method does not require as inputs the underlying data distribution and types. It has been shown [25] that v-optimal and maxdiff histograms perform better than compressed histograms, and that the maxdiff-histogram based method is more accurate than the method based on v-optimal histograms. (In the maxdiff-histogram based method, we use sampling to avoid looking through the entire relation.) Finally, sampling methods look promising for our purposes: In addition to matching our criteria, these methods require no extra storage and are straightforward to implement. We have implemented both simple random sampling and systematic sampling.

We now describe our implementation of the methods we consider promising for view-size estimation.

3.1. Simple Random Sampling

Simple random sampling, also called adaptive sampling, has been proposed by Lipton and Naughton [21]. Based on the values sampled so far, the method estimates the mean and variance for the sample values and stops when stopping conditions are met. We use the stopping conditions suggested by Haas and Swami in Algorithm S2 in [11]. There are three subconditions to stop sampling, based on the variance and mean for the sample values, which are updated whenever a sample is acquired. Given some mean and variance values, we apply the stopping conditions as follows.

Stopping Conditions

1. $n > 0$, where n is the number of sampled values;
2. $S_n^2 > 0$, where S_n^2 is the sample variance;
3. $\epsilon \max(s, n\psi) \geq t(nS_n^2)^{1/2}$: This condition finds the number of tuples that have to be sampled within relative error $\max(s, n\psi)$. For large values of n and assuming the normal sample distribution, this formula follows

³Our goal is to build an extensible system where view-size estimators can be easily added as needed.

from the Standard Central Limit Theorem and prevents the new sample value from exceeding the given confidence interval. A sanity bound ψ is proposed to prevent over-sampling, that is, to limit the sample size n . Over-sampling is avoided by applying the sanity bound ψ in $\max(s, n\psi)$.

3.2. Method Based on MaxDiff Histograms

This method uses value-frequency pairs for attribute values. The use of maxdiff histograms prevents values with vastly different frequencies from being represented in the same bucket, by maintaining bucket boundaries between adjacent frequencies. The method first sorts values in the input relation and calculates the frequency of each value. Then bucket boundaries are inserted, starting from the highest frequency differences. Each histogram bucket stores the number of distinct values, the frequency and the average frequency. By increasing the number of buckets we can estimate query sizes more accurately. Because using this histogram method as described can be time consuming on large relations, our implementation uses sampling to construct the histograms.

3.3. Random Sampling Algorithm

We have implemented a sampling algorithm that is commonly used in selection and join selectivity algorithms. We first determine the number of tuples to sample in a relation. This algorithm is based on the method of [8]. As explained in the paper, the minimum size of a random sample is $r = 4k \frac{\ln(2 \cdot n / \gamma)}{f^2}$, where k is histogram size, n is relation size, f is the maximum relative error in bin size, and γ is the error probability; both k and n are provided by the system catalog. We then start sampling using Algorithm Z described in [30]; the algorithm works by repeatedly computing the size of the next batch of tuples to fetch, which will replace a randomly chosen element of the current tuple reservoir. At all times, the reservoir is a true random sample of the tuples we have passed over so far; the stopping condition is to exhaust the input relation. We use the results of the sampling procedures in our selection and join selectivity estimation algorithms.

3.4. Systematic Sampling

We have implemented the method proposed by Harangri et al [23]. Systematic sampling takes tuples within a k interval from a sorted relation R of size N .

Suppose that the tuples of \mathbf{R} are assigned numbers between 1 and N , in the sorted order. To select a systematic sample of n samples, if $k = \lceil \frac{N}{n} \rceil$ then every k th tuple is selected, starting with a randomly chosen number between 1 and k . For instance, for $N = 2000$ and $n = 200$, $k = 2000/200 = 10$. Therefore, we would select every 20th tuple, starting with a randomly chosen number 5.

4. Experimental Setup

We have implemented the algorithms described in Section 3 in our QPET prototype [9]; the prototype is based on the code of an open-source relational data-management system PostgreSQL version 7.3.4 [4]. We performed the experiments on a single-processor 2.8 GHz Pentium-IV machine with 512MB memory and 80GB hard space running RedHat 9 Linux. We used the TPC-H [29] database of total size 2.89MB (scale factor 1); please see Appendix A for the relation layout in the database. The query workload for the experiments is loosely based on TPC-H queries; please see Appendix B for a full description of the queries we used in the experiments. We have conducted experiments with queries defined using the following parameters. (In the experiments, we used at least 30 queries for each parameter distribution.)

- **Relations sizes:** We divided stored TPC-H data into relations of small (1 to 10,000 tuples), medium (10,000 to 800,000 tuples), and large (over 800,000 tuples) size. In the experiments, we used queries defined on relations of either uniform or mixed sizes. Queries on relations of small size were defined using TPC-H relations `region`, `nation`, and `supplier`; relations `customer`, `part`, and `partsupp` were used to define queries on relations of medium size; finally, we used `orders` and `lineitem` for queries on large-size relations. The small/medium mix used relations `region`, `nation`, `supplier`, `customer`, `part`, and `partsupp`, and so on.
- **Data distribution :** We conducted the experiments using several degrees of skew for the attributes, in the range between the low value of 0 and the high value of 6. We used the following formula for skew:

$$\frac{n}{(n-1)(n-2)} \sum \left(\frac{x_i - \bar{x}}{s} \right)^3,$$

here n is relation size, x_i is attribute value in tuple i , \bar{x} is the average value, and s is standard deviation. We consider the following ranges of degrees

of skew: 0 (which corresponds to the uniform distribution of data), $(0 - 1]$, $(1 - 2]$, and $(2 - 6]$. In case of uniform distribution (i.e., in case of degree of skew 0), we ran separate experiments on non-key and key attributes.

- **Selectivities :** We conducted the experiments using two different types of selectivities. If the (join or selection) selectivity in a query is within the interval $[0, 0.2]$, we say that the query has low selectivity; queries with selectivity within $[0.8, 1]$ are high-selectivity queries. (By design of TPC-H, join selectivities on TPC-H relations do not exceed 1.) In the experiments, we ignored selectivities between 0.2 and 0.8, because the probabilities of the tuples to be selected are almost the same in those cases.
- **Query conditions:** In the experiments, we used select-project-join queries with equality and range (i.e., inequality) selection and join conditions.
- **Sample sizes:** In our experiments, we investigated the accuracy of query-size estimates using various sample fractions. We use 5%, 10% and 15% sample fractions, which are the number of tuples sampled relative to the size of the relation to be sampled.
- **Elapsed time:** We show elapsed time on relations of varying sizes using various sampling fractions. To estimate the time elapsed for a range of conditions on the relations, we used the 10% sampling fraction on all TPC-H stored tables. (Because the stopping conditions for simple random sampling do not depend on relation sizes, the time elapsed in this case does not depend on relation size either. Thus, this approach is ignored for that method. Moreover, we do not use relations `nation` and `region` in the approach, as their size is too small.)
- **Mean relative error:** All the experimental results are presented in terms of the mean relative error, which is defined as

$$\sum_{i=1}^S 100 * \frac{abs(\hat{\mu}_i N - \mu N)}{S \mu N};$$

here, S is the number of samplings, μ is the actual selectivity, $\hat{\mu}$ is the estimated selectivity which results from the i th sampling, $\hat{\mu}_i N$ is the query-size estimate with the i th sampling, and μN is the actual size of the query answer. The range of acceptable values of the relative error is application dependent or is determined by user requirements.

5. Experimental Results

5.1. Impact of Relation Sizes

Figure 3 shows the average relative error between actual query sizes and the outputs of each method, depending on the sizes of relations used in the query definitions. In Figure 3, we use the letter **S** to denote small-size relations, **M** to denote medium-size relations, **L** for large-size relations; the Figure shows results for queries on same-size relations and on combinations. For instance, the value of the average relative error for random sampling is 30 on queries defined using medium- and large-size relations, **M-L**. We can see from Figure 3 that overall, systematic sampling performs better in our experiments than maxdiff and simple random sampling. In fact, systematic sampling method shows almost stable results for all cases except queries on small-size relations. For small-size queries, the maxdiff-histogram based method shows very accurate results — approximately 0% average relative error. In other cases, systematic sampling is the best-performing method. In particular, for queries on large-size relations, the accuracy of systematic sampling is significantly higher than the accuracy of simple random sampling and of the maxdiff method. Based on these results, we suggest using maxdiff histograms when queries are defined on small-size relations; in other cases, systematic sampling gives higher accuracy than the other methods.

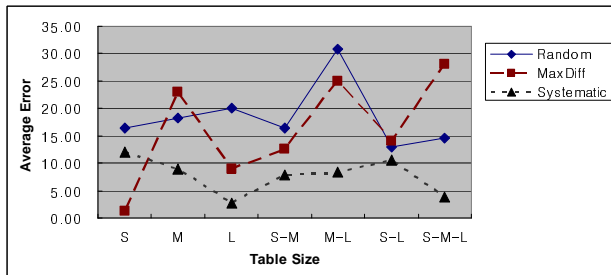


Figure 3. Impact of relation sizes

5.2. Impact of Frequency Skew

Figure 4 shows the average relative error depending on the degree of skew in the frequency set for attribute values. In the Figure, we use **Non** to denote nonskew frequency, **Low** for low skew, **Med** for medium skew, and **High** for high skew, see Section 4 for the skew ranges used in the experiments. (In the nonskew case, we ran the experiments using *primary-key* attributes.)

Overall, our results show that as the degree of skew increases, the accuracy of the outputs of all methods increases with the increase in the degree of skew. In our experiments, maxdiff and systematic sampling exhibited higher accuracy than simple random sampling — the two methods show less than 5% average relative error in most cases. When value frequencies are highly skewed, all three methods perform with high accuracy (i.e., output estimates within 3% of the actual result sizes). In summary, when data frequencies are highly skewed or not skewed at all, we can choose any of three methods because the average errors are very similar. On the other hand, simple random sampling does not seem to be accurate when data frequencies are low-skewed or medium-skewed.

We did another set of experiments for uniformly distributed data, this time using *nonkey* attributes. As can be seen in Figure 5, the results are different from those for the nonskew (**Non**) case in Figure 4. The reason is, the values of primary keys are unique by definition. Therefore, for queries with conditions on a mix of primary keys and nonkey attributes, the estimated relative error is worse than that for queries with conditions on only nonkey attributes. In the experiments with nonkey attributes (Figure 5), systematic sampling shows more accurate results than those of the other two methods; the relative error is low for all three methods.

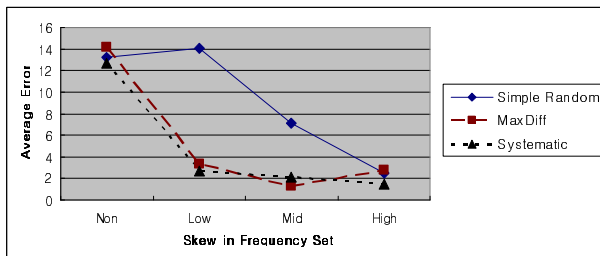


Figure 4. Impact of frequency skew

5.3. Impact of Selectivities

Figure 6 shows the average relative error depending on the values of selectivities. In a low-selectivity environment, the method based on maxdiff histograms and the systematic-sampling method have relatively low errors. In contrast, we see the opposite results in a high-selectivity environment: Simple random sampling performs slightly better than the two other methods. Intuitively, for queries whose conditions have low selectivities it is better to choose systematic sampling.

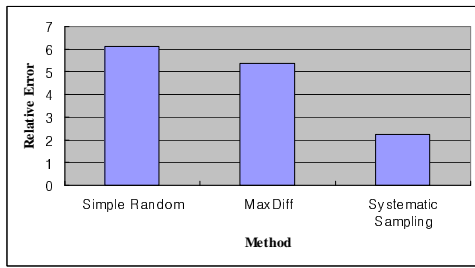


Figure 5. Nonskew data without primary keys

In other cases, simple random sampling is preferred, although all three methods have low error.

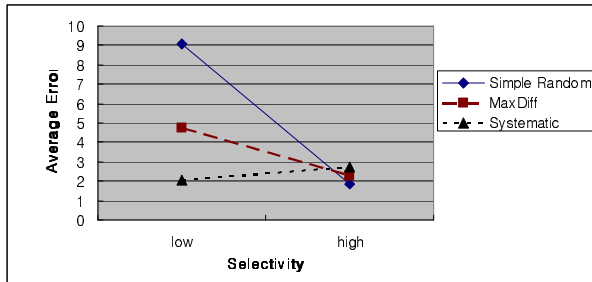


Figure 6. Impact of selectivities

5.4. Impact of Query Conditions

The average relative errors for various query conditions are plotted in Figure 7. For equality conditions, the relative error given by systematic sampling is close to zero, and the maxdiff method shows better results than simple random sampling. We conclude that the best method for queries with equality conditions is systematic sampling. For queries with range selectivities, we see excellent accuracies in all three methods. Overall, systematic sampling has a lower relative error than the other two methods. Intuitively, more accurate results in case of range queries stem from a larger number of candidate samples. Therefore, if a stored relation has repeated values, the probability of selecting a value repeatedly is relatively high. In conclusion, any of the three methods can be used for range queries. It is recommended to use systematic sampling for join operations.

5.5. Impact of Sampling Fractions on Accuracy

We ran experiments with 5, 10 and 15% sampling fractions in the maxdiff-histogram based method and

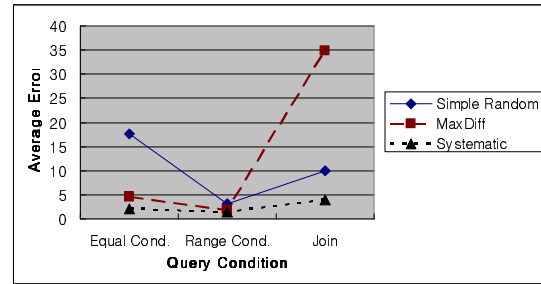


Figure 7. Impact of query conditions

in systematic sampling. For the experiments, instead of using Chaudhuri's algorithm [8] for maxdiff, we set the number of samples in maxdiff to a fixed value of sampling fraction. For systematic sampling, Figure 8 shows that, not surprisingly, by sampling more values we can get more accurate results. Similarly, for maxdiff, we can see in Figure 8 that we can get more accurate result when we apply 10% sampling fraction than in case of 5%. However, we get worse results when we use 15% sampling fraction. Thus, it is not always good to use too many sample values. Therefore, we recommend calibration of sampling sizes in methods for view-size estimation.

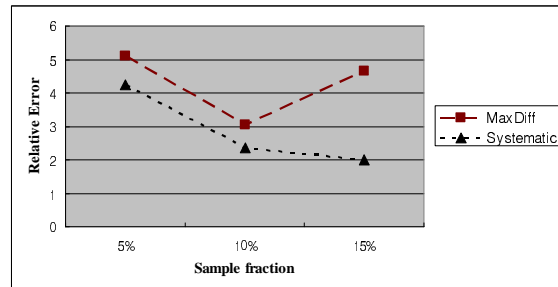


Figure 8. Sampling fractions and accuracy

5.6. Measuring Elapsed Times

We ran experiments to measure the time required to execute each of the three methods using relations and samples of varying sizes. We ignored simple random sampling for the case of relation sizes, because the method stops when many sample values are matched early. Thus, Figure 9 shows the measurements for maxdiff and systematic sampling only. Systematic sampling may take a long time to find the specific location of a sampled tuple. In contrast, because sample values are chosen randomly for maxdiff histograms, histogram construction is not too time consuming unless

the tables are very large. Figure 10 shows the results of another experiment that we did to measure the elapsed time with respect to sample sizes. We can see in the Figure that simple random sampling takes more time than either other method. Intuitively, when many sample values are taken in simple random sampling, it is time consuming to compare and recalculate the mean and variance for each new sample value. For systematic sampling and maxdiff, Figure 10 shows patterns that are similar to the results for relation sizes, see Figure 9.

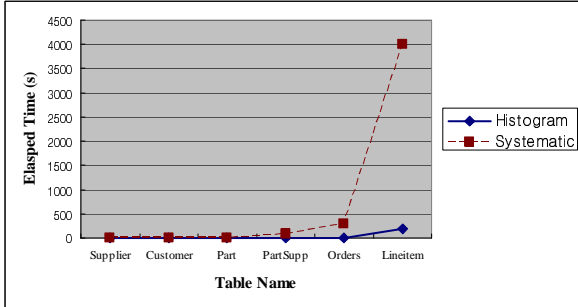


Figure 9. Relation sizes and elapsed times

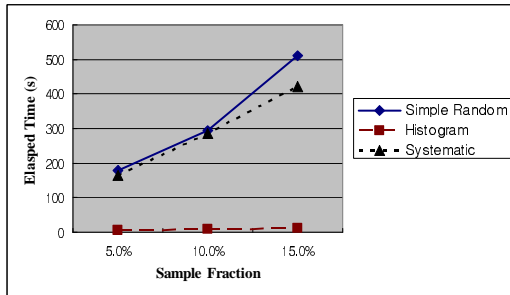


Figure 10. Sample sizes and elapsed times

5.7. Summary of the Experimental Results

In our experiments, we analyzed the accuracy of query-size estimation for select-project-join queries using simple random sampling, maxdiff histograms, and systematic sampling. We measured the accuracy based on the difference between the actual sizes of the query answers on the TPC-H database and the size estimates returned by each method. The goal of the experiments was to determine query parameters whose values would influence the accuracy of the methods, and to use the parameters to formulate guidelines in various query environments. We have elicited the following parameters:

relation size, degree of skew, selectivity, query predicates and sample size fractions, see Section 4 for details. In addition to doing experiments which measured the accuracy of each method based on the values of each parameter, we estimated the runtime of each method with respect to relation and sample sizes.

Overall, systematic sampling is the best method in almost all cases if the stored data is sorted and the runtime of the method is ignored. Maxdiff histograms are a preferred method when relation sizes are small and the degree of frequency skew is between 1 and 2. Finally, simple random sampling is preferred when query selectivities are within the range of 0.8 to 1. When the stored data are not sorted, systematic sampling is not a preferred method because to sort a relation, it scans the entire relation. (In this case, it does not make sense to sample further, because all values in the relation have already been touched by the sorting procedure.) Moreover, systematic sampling is not preferred if elapsed time is a consideration, because it may take too long to find specific tuple positions.

6. Conclusions and Future Work

Deciding whether a materialized view can improve the efficiency of evaluating a query is based on the number of tuples in the materialized view — in particular, if the relation for a materialized view is too large then using the view in query execution could be more expensive than other available strategies for the same query. Given a view definition, one can compute exactly the size of the materialized view by precomputing and storing in the database the answer to the definition; at the same time, using this strategy can be prohibitive in terms of the time and system resources consumed in the precomputation of large view relations. Our QPET system architecture for automated query-performance tuning uses view definitions and feedback from an extended query optimizer to determine whether a view is competitive in answering the given frequent and important queries and thus should be materialized; the optimizer bases its feedback on an estimate of the size of each materialized view based on the view definition.

In this paper we studied three approaches for estimating the sizes of materialized views based on the view definitions; the approaches are based on methods for estimating query-result sizes. We have explored the accuracy of view-size estimates given by the promising methods of simple random sampling, maxdiff histograms, and systematic sampling. We did an implementation and an experimental evaluation of the accuracy of the methods in several environments; in the experimental setup, we defined several parameters in

query definitions and studied how the values of each parameter affect each view-size estimation method. The experimental results can be used as guidelines to choose the best method in each condition when estimating the sizes of materialized views in automated query-performance tuning.

Our experimental results show that systematic sampling is the most accurate and stable method for our purposes. However, when we consider the runtimes of the methods (in particular, sorting in case of systematic sampling), it may take systematic sampling too long to choose specific tuple positions, depending on the relation and sample sizes. We can summarize the recommended methods in each category when sorting procedure and taken time are ignored as follows:

- Size : We can recommend maxdiff histograms with sampling if relation sizes do not exceed 10,000 tuples. Otherwise, systematic sampling is preferred with approximately 10% average relative error.
- Degree of skew: Overall, we can choose any of the three methods in the nonskew case (for primary keys) or if the degree of skew is high. In the other cases, maxdiff histograms and systematic sampling are recommended.
- Selectivity: We recommend systematic sampling if query selectivities are between 0 and 0.2, and recommend simple random sampling if selectivities are within the range 0.8 to 1.
- Query conditions: We recommend systematic sampling for all types of query conditions. In particular, each of the three methods works well for range queries.
- Sampling fraction: We recommend systematic sampling in all 5, 10 and 15% of sampling fractions. For maxdiff histograms, we show that selecting an appropriate sampling fraction may be necessary, as the accuracy of the result is not directly proportional to sample sizes.

We now discuss some directions of our ongoing and of possible future work. We plan to extend our experiments to more parameters and other datasets. Further, more work is needed on examining the tradeoffs between accuracy and resource consumption in view-size estimators. In particular, relatively more expensive methods may be still acceptable when views are materialized offline for queries that are expected to remain frequent and important in the system for a long time (e.g., weeks or months). Finally, different methods seem to be needed for complex queries whose definitions use numerous join and selection conditions. In

this case, the methods we have explored so far are expected to become less accurate as the error accumulates from the leaves to the root of the query plan. The answer in this case could be to evaluate the query on the results of sampling the original stored data. Studying algorithms for accurate sampling of stored data for this purposes is a topic of our future work.

References

- [1] Microsoft Research AutoAdmin Project: Self-Tuning and Self-Administering Databases. <http://research.microsoft.com/dmx/autoadmin/default.asp>.
- [2] Microsoft SQL. <http://www.microsoft.com/sql/>.
- [3] Oracle. <http://www.oracle.com>.
- [4] PostgreSQL. <http://www.postgresql.org>.
- [5] ISO/IEC SQL3. (ISO-ANSI Working Draft) Database Language SQL (SQL3). Document ISO/IEC JTC1/SC21 N6931, American National Standards Institute, 1992. (Later versions available from Working Group or Rapporteur Group documents.).
- [6] Donald D. Chamberlin, Morton M. Astrahan, Michael W. Blasgen, James N. Gray, W. Frank King, Bruce G. Lindsay, Raymond Lorie, James W. Mehl, Thomas G. Price, Franco Putzolu, Patricia Griffiths Selinger, Mario Schkolnick, Donald R. Slutz, Irving L. Traiger, Bradford W. Wade, and Robert A. Yost. A history and evaluation of System R. *Communications of the ACM*, 24(10):632–646, 1981.
- [7] S. Chaudhuri, R. Krishnamurthy, S. Potamianos, and K. Shim. Optimizing queries with materialized views. In *Proceedings of the Eleventh International Conference on Data Engineering (ICDE)*, pages 190–200, 1995.
- [8] Surajit Chaudhuri, Rajeev Motwani, and Vivek Narasayya. Random sampling for histogram construction: how much is enough? In *Proceedings of ACM SIGMOD*, pages 436–447, 1998.
- [9] R. Chirkova, S. Gupta, K.-H. Kim, and S. Sandhu. Extensible framework for query-performance enhancement by tuning. Code downloads and documentation are available from <http://research.csc.ncsu.edu/selftune/>, 2004.

- [10] S. Christodoulakis. Implications of certain assumptions in database performance evaluation. *ACM Trans. Database Syst.*, 9(2):163–186, 1984.
- [11] Peter J. Haas and Arun N. Swami. Sequential sampling procedures for query size estimation. In *Proceedings of ACM SIGMOD*, pages 341–350, 1992.
- [12] Wen-Chi Hou and Gultekin Ozsoyoglu. Statistical estimators for aggregate relational algebra queries. *ACM Trans. Database Syst.*, 16(4):600–654, 1991.
- [13] Wen-Chi Hou, Gultekin Ozsoyoglu, and Baldeo K. Taneja. Processing aggregate relational queries with hard time constraints. *SIGMOD Record*, 18(2):68–77, 1989.
- [14] IBM. Autonomic Computing. <http://www.research.ibm.com/autonomic/>.
- [15] Yannis E. Ioannidis. Universality of serial histograms. In *Proceedings of VLDB*, pages 256–267, 1993.
- [16] Yannis E. Ioannidis. Query optimization. In *The Computer Science and Engineering Handbook*, pages 1038–1057. 1997.
- [17] Yannis E. Ioannidis and Stavros Christodoulakis. On the propagation of errors in the size of join results. In *SIGMOD*, pages 268–277, 1991.
- [18] Yannis E. Ioannidis and Viswanath Poosala. Balancing histogram optimality and practicality for query result size estimation. In *Proceedings of ACM SIGMOD*, pages 233–244, 1995.
- [19] Robert Philip Kooi. *Optimization of queries in relational databases*. PhD thesis, 1980.
- [20] Yibei Ling and Wei Sun. An evaluation of sampling-based size estimation methods for selections in database systems. In *Proceedings of ICDT*, pages 532–539, 1995.
- [21] Richard J. Lipton, Jeffrey F. Naughton, and Donovan A. Schneider. Practical selectivity estimation through adaptive sampling. In *Proceedings of ACM SIGMOD*, pages 1–11, 1990.
- [22] Akifumi Makinouchi, Masayoshi Tezuka, Hajime Kitakami, and S. Adachi. The optimization strategy for query evaluation in RDB/V1. In *VLDB*, pages 518–529, 1981.
- [23] A. H. H. Ngu, B. Harangsri, and J. Shepherd. Query size estimation for joins using systematic sampling. *Distributed and Parallel Databases*, 15(3):237–275, 2004.
- [24] Gregory Piatetsky-Shapiro and Charles Connell. Accurate estimation of the number of tuples satisfying a condition. In *Proceedings of ACM SIGMOD*, pages 256–276, 1984.
- [25] Viswanath Poosala, Peter J. Haas, Yannis E. Ioannidis, and Eugene J. Shekita. Improved histograms for selectivity estimation of range predicates. In *Proceedings of ACM SIGMOD*, pages 294–305, 1996.
- [26] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price. Access path selection in a relational database management system. In *Proceedings of ACM SIGMOD*, pages 23–34, 1979.
- [27] Dennis Shasha and Philippe Bonnet. *Database Tuning: Principles, Experiments, and Troubleshooting Techniques*. Morgan Kaufmann, 2002. <http://www.distlab.dk/dbtune/>.
- [28] Arun Swami and K. Bernhard Schiefer. On the estimation of join result sizes. In *Proceedings of EDBT*, pages 287–300, 1994.
- [29] TPC-H. TPC Benchmark H (Decision Support). Available from <http://www.tpc.org/tpch/spec/tpch2.1.0.pdf>.
- [30] Jeffrey S. Vitter. Random sampling with a reservoir. *ACM Trans. Math. Softw.*, 11(1):37–57, 1985.
- [31] Qiang Zhu. An integrated method for estimating selectivities in a multidatabase system. In *Proceedings of the 1993 Conference of the Centre for Advanced Studies on Collaborative research*, pages 832–847. IBM Press, 1993.

A. TPC-H Table Layout

In this section, we explain layouts for TPC-H tables. We indicate column names, data types, sizes and primary keys.

PART Table

<u>Column Name</u>	<u>Data Type</u>
PARTKEY	identifier
NAME	variable text, size 55
MFGR	fixed text, size 25
BRAND	fixed text, size 10
TYPE	variable text, size 25
SIZE	integer
CONTAINER	fixed text, size 10
RETAILPRICE	decimal
COMMENT	variable text, size 23
Primary Key	PARTKEY

SUPPLIER Table

<u>Column Name</u>	<u>Data Type</u>
SUPPKEY	identifier
NAME	fixed text, size 25
ADDRESS	variable text, size 40
NATIONKEY	identifier
PHONE	fixed text, size 15
ACCTBAL	decimal
COMMENT	variable text, size 101
Primary Key	SUPPKEY

PARTSUPP Table

<u>Column Name</u>	<u>Data Type</u>
PARTKEY	identifier
SUPPKEY	identifier
AVAILQTY	integer
SUPPLYCOST	decimal
COMMENT	variable text, size 199
Primary Key	PARTKEY, SUPPKEY

CUSTOMER Table

<u>Column Name</u>	<u>Data Type</u>
CUSTKEY	identifier
NAME	variable text, size 25
ADDRESS	variable text, size 40
NATIONKEY	identifier
PHONE	fixed text, size 15
ACCTBAL	decimal
MKTSEGMENT	fixed text, size 10
COMMENT	variable text, size 117
Primary Key	CUSTKEY

ORDERS Table

<u>Column Name</u>	<u>Data Type</u>
ORDERKEY	identifier
CUSTKEY	identifier
ORDERSTATUS	fixed text size, size 1
TOTALPRICE	decimal
ORDERDATE	date
ORDERPRIORITY	fixed text, size 15
CLERK	fixed text, size 15
SHIPPRIORITY	integer
COMMENT	variable text, size 79
Primary Key	ORDERKEY

NATION Table

<u>Column Name</u>	<u>Data Type</u>
NATIONKEY	identifier
NAME	identifier
REGIONKEY	identifier
COMMENT	variable text, size 152
Primary Key	NATIONKEY

REGION Table

<u>Column Name</u>	<u>Data Type</u>
REGIONKEY	identifier
NAME	identifier
COMMENT	variable text, size 152
Primary Key	REGIONKEY

LINEITEM Table

<u>Column Name</u>	<u>Data Type</u>
ORDERKEY	identifier
PARTKEY	identifier
SUPPKEY	identifier
LINENUMBER	integer
EXTENDEDPRICE	decimal
DISCOUNT	decimal
TAX	decimal
RETURNFLAG	fixed text, size 1
LINESTATUS	fixed text, size 1
SHIPDATE	date
COMMITDATE	date
RECEIPTDATE	date
SHIPINSTRUCT	fixed text, size 25
SHIPMODE	fixed text, size 10
COMMENT	variable text, size 44
Primary Key	ORDERKEY, LINENUMBER

B. Queries for Experiments

In this section, we explain representative queries for each category to use experimental results.

B.1. Size Category

This category has small, medium, large, small-medium, small-large, medium-large, small-medium-large set. For each set of table sizes, we illustrate queries for experiments.

1. SMALL TABLE : Table sizes are less than or equal to 10,000.

```
select regionkey from nation
where regionkey = '3';
```

```
select nationkey from nation
where nationkey = '12';
```

```
select regionkey from region
where regionkey = '4';
```

```
select nationkey from supplier
where nationkey = '21';
```

```
select acctbal from supplier
where acctbal = 8091.65;
```

```
select suppkey from supplier
where suppkey = '3942';
```

```
select r.regionkey, n.nationkey
from nation n,region r
where n.nationkey = '17'
and r.regionkey = '4'
and n.regionkey = r.regionkey;
```

```
select * from nation n,region r
where r.regionkey = n.regionkey;
```

```
select r.regionkey from nation n,region r
where n.regionkey = '1'
and n.regionkey = r.regionkey;
```

```
select * from supplier s,nation n
where s.nationkey = '8'
and s.nationkey = n.nationkey;
```

2. Medium Table : Table sizes are greater than 10,000 and less than or equal to 800,000.

```
select custkey from customer
where custkey = '7204';
```

```
select acctbal from customer
where acctbal = 1228.24;
```

```
select mktsegment from customer
where mktsegment = 'HOUSEHOLD';
```

```
select partkey from part
where partkey = '4941';
```

```
select mfgr from part
where mfgr = 'Manufacturer#3';
```

```
select brand from part
where brand = 'Brand#43';
```

```
select type from part
where type = 'PROMO ANODIZED STEEL';
```

```
select * from part p,partsupp ps
where p.partkey = ps.partkey;
```

```
select * from part p,partsupp ps
where p.partkey = '32'
and p.partkey = ps.partkey;
```

```
select brand from part
where brand >= 'Brand#20';
```

```
select availqty from partsupp
where availqty > 50;
```

3. Large Table : Table sizes are greater than 800,000 and less than or equal to 6,001,215.

```
select orderstatus from orders
where orderstatus = 'F';
```

```
select orderdate from orders
where orderdate = date '1992-01-23';
```

```
select orderpriority from orders
where orderpriority = '5-LOW';
```

```
select partkey from lineitem
where partkey = '32012';
```

```
select linenum from lineitem
where linenum = 4;
```

```
select orderstatus
from orders o,lineitem l
where o.orderkey = l.orderkey
and o.orderkey = '1000097';
```

```
select orderstatus from orders
where orderstatus <= '0';
```

```
select orderdate from orders
where orderdate <= date '1992-10-25';
```

```
select * from lineitem
where linenumber <= 5;
```

```
select * from lineitem
where commitdate > date '1992-06-21';
```

4. S-M Tables : Mixed type of small and medium sizes of tables.

```
select * from partsupp ps,supplier s
where ps.suppkey = s.suppkey;
```

```
select * from partsupp ps,supplier s
where ps.availqty >= 670
and s.suppkey = '7310'
and ps.suppkey = s.suppkey;
```

```
select * from partsupp ps,supplier s
where ps.supplycost < 1000.00
and s.acctbal < 3000.00
and ps.suppkey = s.suppkey;
```

```
select * from customer c,supplier s
where c.nationkey = s.nationkey;
```

```
select * from customer c,supplier s
where c.acctbal <= 3000.00
and s.acctbal <= 0.00
and c.nationkey = s.nationkey;
```

```
select * from nation n,region r,part p
where p.retailprice >= 935.00
and r.regionkey = n.regionkey;
```

```
select *
from part p,partsupp ps,
customer c,nation n
where p.container = 'JUMBO BOX'
and ps.supplycost = 1.00
and p.partkey = ps.partkey
and n.nationkey = c.nationkey;
```

```
select *
from part p,partsupp ps,
customer c,nation n
where p.partkey = ps.partkey
and c.acctbal <= 4000.00
and c.nationkey = n.nationkey;
```

```
select s.nationkey,p.size,ps.supplycost
from supplier s,customer c,
```

```
part p,partsupp ps
where s.nationkey = '10'
and s.nationkey = c.nationkey
and p.size > 50
and ps.supplycost <= 12.00
and p.partkey = ps.partkey;
```

```
select r.name,p.size,ps.availqty
from region r,nation n,supplier s,
part p,partsupp ps,customer c
where r.regionkey = n.regionkey
and n.regionkey = '4'
and s.nationkey = n.nationkey
and p.size < 150
and ps.availqty >=1000
and p.partkey = ps.partkey
and c.acctbal = 6853.37
and c.nationkey = n.nationkey;
```

5. S-L Tables : Mixed type of small and large sizes of tables

```
select n.name, o.orderstatus
from nation n,orders o
where n.name = 'JAPAN'
and o.orderstatus = 'F';
```

```
select n.nationkey, o.orderpriority
from nation n,orders o
where n.nationkey = '12'
and o.orderpriority > '3-MEDIUM';
```

```
select clerk
from nation n,orders o
where clerk = 'Clerk#000000039'
and n.nationkey = '22';
```

```
select n.nationkey, l.returnflag
from nation n,lineitem l
where n.nationkey = '4'
and l.returnflag = 'R';
```

```
select o.orderpriority
from orders o,region r
where o.orderpriority <= '4-NOT SPECIFIED'
and r.regionkey < '3';
```

```
select n.nationkey, o.orderpriority
from nation n,orders o,supplier s
where n.nationkey = s.nationkey
and o.orderpriority = '5-LOW';
```

```
select o.orderdate,n.nationkey
from orders o,nation n,supplier s
```

```

where o.orderdate >= date '1992-09-10'
and n.nationkey = s.nationkey;

select o.clerk,n.nationkey
from orders o,nation n,supplier s
where o.clerk = 'Clerk#000000028'
and n.nationkey = '4'
and s.acctbal > 10200.00
and s.nationkey = n.nationkey;

select *
from lineitem l,nation n,region r
where l.quantity <= 1000.00
and n.regionkey = r.regionkey;

select n.regionkey, l.quantity
from nation n,region r,lineitem l
where n.regionkey = r.regionkey
and l.quantity > 500.00;

```

6. M-L Tables : Mixed type of medium and large sizes of tables.

```

select c.custkey, c.mktsegment
from customer c,orders o
where c.custkey = o.custkey;

select o.custkey, c.acctbal
from customer c,orders o
where o.orderkey = '79999'
and c.acctbal >= 8000.00
and c.custkey = o.custkey;

select c.custkey, c.mktsegment
from orders o,customer c
where c.mktsegment = 'MACHINERY'
and o.clerk = 'Clerk#000000010'
and c.custkey = o.custkey;

select p.brand from part p,orders o
where p.brand = 'Brand#55'
and o.totalprice <= 300000.00;

select ps.supplycost
from partsupp ps,orders o
where ps.supplycost >= 10000.00
and o.custkey = '108290';

select l.linenumbr
from lineitem l,customer c
where l.linenumbr >= 7
and c.phone = '32-363-455-4837';

select l.partkey, p.type

```

```

from lineitem l,part p
where l.quantity < 100
and p.type = 'ECONOMY ANODIZED STEEL'
and l.partkey = p.partkey;

select p.partkey, l.returnflag
from lineitem l,part p
where size >= 50
and l.returnflag = 'A'
and l.partkey = p.partkey;

select l.supkey,l.returnflag
from partsupp ps,lineitem l
where ps.supkey = l.supkey;

select *
from partsupp ps,lineitem l
where ps.availqty > 1000000
and l.linenumbr < 4
and ps.supkey = l.supkey;

```

7. S-M-L Tables : Mixed type of small, medium and large sizes of tables.

```

select n.nationkey,c.custkey
from nation n,customer c,orders o
where n.nationkey = '3'
and c.custkey = o.custkey
and n.nationkey = c.nationkey;

select c.custkey, c.mktsegment
from customer c,orders o, nation n
where c.mktsegment = 'HOUSEHOLD'
and n.nationkey = '13'
and o.orderpriority = '3-MEDIUM'
and c.custkey = o.custkey
and n.nationkey = c.nationkey;

select o.custkey
from customer c,orders o,nation n
where c.custkey = o.custkey
and n.nationkey = '11'
and c.nationkey = n.nationkey;

select o.custkey, o.clerk
from nation n,customer c,orders o
where o.clerk = 'Clerk#000000002'
and n.nationkey = c.nationkey
and o.custkey = c.custkey;

select r.regionkey,p.size
from orders o,part p,region r
where r.regionkey = '4'
and p.size < 16

```

```

and o.orderpriority = '5-LOW';

select s.nationkey, l.supkey
from lineitem l, supplier s, customer c
where l.linenum = 3
and s.nationkey = c.nationkey
and l.supkey = s.supkey;

select *
from nation n, part p, orders o
where n.name = 'JORDAN'
and p.type = 'ECONOMY BRUSHED NICKEL'
and o.orderdate = date '1992-11-30';

select *
from orders o, partsupp ps, supplier s
where o.orderkey = '208321'
and ps.supplycost < 47000.00
and s.supkey = '5311'
and ps.supkey = s.supkey;

select ps.partkey, ps.supkey
from partsupp ps, supplier s, orders o
where ps.supkey = s.supkey
and o.orderstatus = 'F';

select n.regionkey, l.linenum
from nation n, customer c, lineitem l
where n.regionkey = '1'
and c.custkey = '32000'
and l.linenum <= 3
and c.nationkey = n.nationkey;

```

B.2. Skew Category

This category has non-skew, low-skew, mid-skew and high-skew queries.

1. Non Skew : Degree of skew is 0.

```

select regionkey from region
where regionkey = '3';

select regionkey from nation
where regionkey = '4';

select mfgr from part
where mfgr = 'Manufacturer#5';

select supkey, acctbal from supplier
where acctbal = 4641.48;

select p.partkey, ps.supkey
from partsupp ps, supplier s

```

```

where ps.supkey = s.supkey;

select n.regionkey, s.phone
from region r, nation n, supplier s
where r.regionkey = n.regionkey
and s.phone = '14-144-830-2814'
and s.nationkey = n.nationkey;

select ps.supkey
from supplier s, part p, partsupp ps
where ps.supkey = s.supkey
and p.mfgr = 'Manufacturer#2'
and p.partkey = ps.partkey;

select * from region
where regionkey > '1000';

select * from nation
where nationkey > '11';

select * from supplier
where acctbal < 3393.08;

```

2. Low Skew : Degree of skew is greater than 0 and less than or equal to 1.

```

select * from orders
where clerk = 'Clerk#000000028';

select * from orders
where orderdate = date '1992-02-22';

select * from part
where type = 'ECONOMY PLATED TIN';

select * from part where size = 29;

select * from part
where mfgr = 'Manufacturer#4';

select * from lineitem
where linenum = 6;

select l.partkey, s.nationkey
from lineitem l, part p, supplier s
where l.shipmode = 'MAIL'
and p.container = 'WRAP BOX'
and l.partkey = p.partkey
and l.supkey = s.supkey
and s.nationkey > '10';

select c.custkey
from customer c, part p
where c.mktsegment = 'MACHINERY'

```

```
and p.size = 30
and c.custkey = o.custkey;
```

```
select discount from lineitem
where discount > 0.04;
```

```
select discount from lineitem
where discount = 0.05;
```

3. Mid Skew : Degree of skew is greater than 1 and less than or equal to 2.

```
select * from orders
where orderstatus = 'P';
```

```
select * from orders
where orderpriority = '4-NOT SPECIFIED';
```

```
select tax from lineitem
where tax = 0.02;
```

```
select orderkey,partkey,suppkey
from lineitem
where returnflag = 'A';
```

```
select orderkey,partkey,suppkey,tax
from lineitem where tax <= 0.03;
```

```
select * from lineitem l,orders o
where o.orderstatus = 'F'
and l.tax > 0.00
and l.orderkey = o.orderkey;
```

```
select * from orders o,lineitem l
where o.orderpriority = '3-MEDIUM'
and l.returnflag < 'N'
and l.orderkey = o.orderkey;
```

```
select * from lineitem l,orders o
where o.orderstatus < 'O'
and l.returnflag = 'A'
and l.orderkey = o.orderkey;
```

```
select orderstatus from orders
where orderstatus > 'F';
```

```
select * from lineitem l,orders o
where o.orderpriority > '1-URGENT'
and l.tax = 0.06
and o.orderkey = l.orderkey;
```

4. High Skew : Degree of skew is greater than 2 and less than or equal to 6.

```
select * from lineitem
where shipdate = date '1992-02-22';
```

```
select * from lineitem
where commitdate = date '1992-04-20';
```

```
select * from lineitem
where shipdate > date '1992-02-29';
```

```
select * from lineitem
where commitdate > date '1992-06-22';
```

```
select * from lineitem
where shipdate < date '1992-02-13';
```

```
select * from lineitem
where commitdate < date '1992-05-31';
```

```
select * from lineitem
where shipdate <= date '1992-02-24';
```

```
select * from lineitem
where commitdate <= date '1992-04-26';
```

```
select * from lineitem
where shipdate >= date '1992-06-14';
```

B.3. Selectivity Category

In this section, we suggest queries for experiments in selectivity category.

1. Low Selectivity : Selectivity is between 0 and 0.2.

```
select * from region
where regionkey = '3';
```

```
select * from nation
where nationkey = '11';
```

```
select * from customer
where nationkey = '19';
```

```
select * from orders
where clerk = 'Clerk#000000038';
```

```
select * from supplier s,nation n
where s.nationkey = n.nationkey;
```

```
select * from part p,partsupp ps
where p.partkey = ps.partkey;
```

```
select *
from supplier s,nation n,region r
```

```

where n.regionkey = r.regionkey
and s.acctbal > 0.00
and s.nationkey = n.nationkey;

```

```

select *
from customer c,nation n,orders o
where c.nationkey = n.nationkey
and o.orderdate > date '1992-01-03'
and o.custkey = c.custkey;

```

```

select * from customer
where mktsegment = 'AUTOMOBILE';

```

```

select * from lineitem
where discount = 0.04;

```

2. High Selectivity : Selectivity is between 0.8 and 1.

```

select * from orders where shippriority = 0;

```

```

select * from orders
where clerk > 'Clerk#000000014';

```

```

select * from lineitem
where quantity > 10.00;

```

```

select *
from lineitem l,partsupp ps,
supplier s,nation n
where l.quantity > 8.00
and s.nationkey = n.nationkey
and ps.availqty > 10;

```

```

select * from part where brand < 'Brand#52';

```

```

select *
from partsupp ps,lineitem l,
part p,customer c
where ps.availqty > 5
and linenum < 7
and mfg > 'Manufacturer#1'
and c.mktsegment >= 'AUTOMOBILE';

```

```

select * from orders,lineitem
where linenum > 1
and orders.orderdate > date '1992-01-03';

```

```

select * from lineitem
where extendedprice > 910.00;

```

```

select *
from orders o,nation n,region r
where o.shippriority = 0
and n.nationkey > '0'

```

```

and r.regionkey > '0';

```

B.4. Query Conditions

In this section, we suggest queries for equality, range and join conditions for experiments.

1. Equality Conditions : The equality predicates (=) are used for selection operations in queries.

```

select partkey from part
where partkey = '10';

```

```

select supplycost from partsupp
where supplycost = 1.26;

```

```

select availqty from partsupp
where availqty = 32;

```

```

select suppkey from partsupp
where suppkey = '1001';

```

```

select type from part
where type = 'ECONOMY PLATED COPPER';

```

```

select size from part where size = 21;

```

```

select * from orders
where totalprice = 112986.49;

```

```

select * from orders
where orderdate = date '1992-01-18';

```

```

select * from orders
where orderstatus = 'P';

```

```

select * from lineitem
where linenum = 3;

```

2. Range Conditions : The range predicates (<, >, <=, >=) are used for selection operations in queries.

```

select * from region
where regionkey > '3';

```

```

select * from supplier
where acctbal < 10000.00;

```

```

select * from supplier
where acctbal <= 30000.00;

```

```

select * from part
where brand > 'Brand#34';

```

```

select * from part
where brand <= 'Brand#15';

select * from part where size > 21;

select * from part where size <= 39;

select * from partsupp
where supplycost > 20000.00;

select * from orders
where totalprice < 15000.00;

select * from lineitem
where linenumber >= 2;

```

3. Join Queries : The equality predicates (=) are used for join operations in queries.

```

select * from supplier s,nation n
where s.nationkey = n.nationkey;

select * from supplier s,partsupp ps
where s.supkey = ps.supkey;

select * from customer c,nation n
where c.nationkey = n.nationkey;

select * from orders o,customer c
where o.custkey = c.custkey;

select * from orders o,lineitem l
where o.orderkey = l.orderkey;

select * from lineitem l,supplier s
where l.supkey = s.supkey;

select * from customer c,supplier s
where c.nationkey = s.nationkey;

select * from lineitem l,part p
where l.partkey = p.partkey;

select * from lineitem l,partsupp ps
where l.supkey = ps.supkey;

select * from region r,nation n
where r.regionkey = n.regionkey;

```

B.5. Sampling Fractions

In this section, we suggest queries for various sampling fractions. All queries are same with all

sampling fractions for the accurate comparison. Queries consist of selection operations using equality (=) and range predicates (<, <=, >=, >).

```

select * from supplier
where nationkey >= '3';

select * from supplier
where nationkey = '21';

select * from partsupp
where supkey = '1015';

select * from partsupp
where supplycost >= 100.00;

select * from part
where brand = 'Brand#55';

select * from customer
where nationkey = '21';

select * from customer
where mktsegment = 'FURNITURE';

select * from orders
where custkey >= '108284';

select * from orders
where clerk <= 'Clerk#000000006';

select * from lineitem
where linenumber = 7;

```