ABSTRACT

KIM, KYOUNG HWA. Query Size Estimation through Sampling. (Under the direction of Rada Y. Chirkova).

Current Database management systems (DBMS) handle huge amounts of data and need fast query response time. DBMSs apply several strategies to execute user queries. Query optimizer in DBMSs compare costs for these strategies and choose the cheapest one. Materialized views are suggested to enhance query response time as one of the strategies in DBMSs. Cost of each strategy has to be accurately estimated to choose right strategy. Because the materialized view is stored as a one table, we consider that sequential scan is used for executing of the materialized view. Therefore, I/O and CPU costs to execute materialized views depend on the number of tuples for the result. Hence, Our focus will be the accuracy of estimation of the number of tuples in materialized views; that is a query-size estimation. Many researches have been proposed to find methods to estimate the cost of query. This thesis reviews these researches and compares good and bad aspects for each cost estimation method. We choose size estimation methods that are more accurate than others to implement. We suggest various query environments for experiments. We suggest a guideline by experimental results.

Query Size Estimation through Sampling

by

Kyoung Hwa Kim

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial satisfaction of the
requirements for the Degree of
Master of Science

Department of Computer Science

Raleigh

2005

Approved By:

Dr. Jaewoo Kang	Dr. Xiaosong Ma

Dr. Rada Y. Chirkova
Chair of Advisory Committee

To my parents,

Yong-Chan Kim

and

Chae-Young Kim.

Biography

Kyoung Hwa Kim was born and grew up in Seoul, Korea. She received her Bachelors degree in Computer Science from Catholic University, Bucheon, Korea, in 1998. She has been a Masters student in the Department of Computer Science at North Carolina State University since August 2002.

Acknowledgements

Many thanks to Dr. Rada Y. Chirkova, my advisor, for her guidance and support. I would like thank to Dr. Jaewoo Kang and Dr. Xiaosong Ma for being on my advisory committee members. I would like to thank Shalu Gupta and Charles Loftis for their advices and encouragements for our project and thesis preparation. I would like to thank my friends in the Triangle Korean Catholic Community for their concern and encouragements. I also would like to thank my parents for their support and endless love for entire my life. I appreciate their patient and trust on my plan of life. Finally, I sincerely thank God for his mercy and blessing.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview of Research

Most companies that use database management systems (DBMS) work with large amounts of data and want to use it efficiently. In particular, Data Warehousing that collects, organizes and makes data available for the purpose of analysis has to handle huge amounts of data with fast query response time. The amount of data maintained by DBMSs has grown enormously and the research to improve query response time has been proposed over the past decade [8, 18, 17, 5, 13].

In general, a SQL statement can be executed using various strategies such as full table scan, nested loops and hash joins. A query optimizer estimates the costs of executing a query using various strategies. The query optimizer then compares these costs and determines the strategy with the lowest cost. Most of the current commercial DBMSs [1, 2, 3] have introduced the concept of a materialized view to improve performance of the database. A materialized view is a set of copies or replicas of data based on SQL queries created in the same manner as dynamic views. Both materialized and dynamic views are created as a single table that is derived from other base tables or predefined views [10]. Although dynamic views and materialized views are similar concepts, dynamic views do not exist in physical form; it is considered a virtual table. On the other hand, a materialized view is pre-computed and physically stored as a table.

Cost of materialized view is decided by the number of tuples for the query result, I/O, and CPU costs to execute a query. Because materialized view is stored as a one table, we consider that sequential scan is used for executing of the materialized view. Therefore, I/O and CPU costs to execute materialized views depend on the number of tuples for the result. Hence, Our focus will be the accuracy of estimation of the number of tuples in materialized views; that is a query-size estimation. For the purpose of query size estimation, several methods have been studied in the past decade. In the beginning, query size is estimated using the number of distinct values and total tuples in the system R [6]. This method has been commonly used until now. However, this method may not be able to get a correct result because of the skewed data in tables. Histogram methods have been proposed to prevent the above problem. Moreover, there is another problem in practice. Most of tables contains a large number of tuples. If we look through all tuples in the tables to construct a histogram, we may encounter a time consuming problem. A good approach to solve time consuming problem is to use a sampling method. In particular, several different methods have been suggested for histogram and sampling. Through this thesis, we review the concepts of these methods, implement promising methods that we have selected, and suggest a guideline for choosing the best method in various environments using experimental results.

1.2 Thesis Contributions

In this thesis, we evaluate the best methods to estimate query sizes using various query types. We use reliable and promising techniques that have been researched so far. We recommend the absolutely most accurate and optimal methods in each query condition. The results have been reported after several experiments. We can flexibly apply these recommended size estimation methods by query types whenever materialized views are used. Using these results, we can believe that the estimated view size is accurate and an optimizer can correctly choose the cheapest strategy to execute queries.

1.3 Thesis Organization

Chapter 2 suggests related works to our research. Chapter 3 reviews the background of query size estimation methods discussed in chapter 2. It suggests several estimation techniques that has been researched in this area. In chapter 4, the selectivity estimation algorithms that we have implemented are explained. It introduces pseudocode and procedures with examples for selection and join conditions. Chapter 5 shows experimental results with graphs for the implemented methods. Finally, chapter 6 concludes this thesis and suggests recommended methods in various query environments.

Chapter 2

Related Work

A large number of researchers have studied query or view size estimation methods in database management systems. This work can be classified into three categories namely parametric [26, 6, 22, 27], histogram [19, 24, 16, 17], and sampling [11, 14, 15, 20, 21, 29] methods. We describe each of them briefly.

Parametric methods are based on underlying data distribution assumptions such as uniform, normal, poisson, zipf and so on. Selinger *et al.* [26, 6], Makinouchi *et al.* [22] and Swami and Schiefer [27] proposed parametric method for the uniform distribution assumption and Christodoulakis [9] relies on the Normal and Pearson Type 2 and 7 distributions for query size estimation of selections and joins. The drawback of this method is that we can not get accurate results if the model is not fit for the actual distribution because we can not know a priori the distribution of data.

A histogram is built by partitioning data distribution into mutually disjoint subsets called buckets and approximates data frequencies in each bucket. Several types of histograms have been proposed and evaluated for their accuracy using equi-width and equi-height [19, 24], maxdiff, compressed, end-biased and v-optimal histograms [17, 25]. Histograms are easy to implement and do not require underlying data distribution assumptions.

Sampling is one of the most recent size estimation method. Sampling estimates query size by collecting and processing random samples of the data. Lipton and Naughton [21] and Hass and Swami [11] proposed adaptive sampling and systematic sampling has

been proposed by Harangsri *et al* [23]. These researches show high accuracy on quality of estimation.

Since the purpose of our thesis is coming up with good guidelines for finding the absolutely most accurate size estimation methods in various environments, we review in more detail parametric, histogram and sampling methods in chapter 3 and we choose promising methods to implement and analyze.

Chapter 3

Basic Concepts of Query Size Estimation

The most important factor on query size estimation is the number of tuples for the result of a given query. Hence, we will focus on methods to estimate the number of tuples. In this chapter, we describe terminology, definitions and notations for the database systems and query size estimation. We then suggest our criteria to choose techniques to implement. We then explain basic concepts in order to understand techniques. We finally show several methods that have been studied for several years.

3.1 Terminology in Database Systems

Example 3.1

Suppose we have User table with name and ssn like following.

NAME	SSN
Alice	555-55-5555
Bob	222-22-2222

- **Tuple** : a row in a table. In example 3.1, we have two tuples : "Alice, 555-55-5555" and "Bob, 222-22-2222".
- **Attribute** : a column head for a table. In example 3.1, we have two attributes : Name and SSN.

Example 3.2

Q1 : **SELECT** S.NATIONKEY, N.REGIONKEY

FROM SUPPLIER S, NATION N

WHERE S.ACCTBAL = 3.78

AND S.NATIONKEY = N.NATIONKEY;

This query asks for nation key and region key of supplier where account balance of supplier is 3.78 and all nation keys in supplier table are in nation table.

- **Selection** : In example 3.2, S.ACCTBAL = 3.78 in the WHERE clause is a selection operation. It means that we want to get acctbal values from the supplier table that satisfy the value is 3.78.
- **Join** : In example 3.2, S.NATIONKEY = N.NATIONKEY in the WHERE clause is join operation. It means that we want to get nationkey values that are same in both of supplier and nation tables.
- **Projection** : In example 3.2, projection operations are S.NATIONKEY and N.REGIONKEY in SELECT clause. It means that we want to get nationkey in supplier table and regionkey attribute in nation table that satisfy conditions in WHERE clauses.

3.2 Definitions and Notations for View Size Estimation

- **Frequency** : the number of times an attribute value of R.b occurs in relation R. We use the term data distribution as a synonym for the term frequency distribution.

- **Size** : The term "size" used throughout the thesis is equivalent to a number of tuples.
- **Selectivity** : the term is the ratio of the size of the output relation due to a join or selection operation over cartesian product of sizes of all the relations which participate in a query.

3.3 Problem Statement

We consider materialized view a physical table and the table is sequentially scanned. Therefore, I/O and CPU costs are based on the number of tuples for the result of a query that consists of a materialized view. For our results to get the best methods in various conditions, we take many types of queries that are comprised of selection, join and projection conditions as a input. Query types are described in chapter 5. Then, we estimate sizes for input queries by several methods which are selected by our criteria. All size estimation methods depend on selectivities for selection, join and projection operations as described in section 3.4. We have four criteria to decide useful methods for our implementations in several studied techniques. The first criterion is the absolutely most accurate methods among all categories by several studied researches. The second thing is data distribution without uniform distribution assumption. The third thing is easy to implement. The forth thing is that we have to handle all kinds of data types as well as integer types. We investigate advantages and disadvantages for query size estimation methods that are explained in chapter 2 and compare methods for each category. Our implementations take accurate methods by these results. Finally, we get query result sizes for each method as a output.

3.4 Selectivity Calculation

Query size estimation methods are based on selectivities for selection and join operations first. The result size of projection operation is estimated after selection and join operations are applied. In the example 3.2, a selection condition "S.ACCTBAL = 3.78" and a join condition "S.NATIONKEY = N.NATIONKEY" in the WHERE clause are sequentially estimated and then the output of the projection operations "S.NATIONKEY and N.REGIONKEY" are estimated. In order to know the number of tuples for outputs

of selection and join operations, we have to calculate selectivities. The *selection selectivity* is the ratio of the result that satisfies selection operation over the total number of tuples of a given relation. The *join selectivity* is the ratio of the result that satisfies join operation over the cartesian product of two participated relations. When a query has both of selection and join operations, an input for estimation of join selectivity is the output of the selection operations. Table 3.1 shows definitions for selectivities for selection and join operations. Example 3.3 shows how the selectivities for selection and join operations are

sel(selection) : the number of tuples that satisfy selection condition selection selectivity = $\frac{sel(selection)}{N_R}$ sel(join) : the number of tuples that satisfy join condition join selectivity = $\frac{sel(join)}{N_R * N_S}$

Table 3.1: Selectivity Definitions (N_R : the number of tuples in the relation R, N_S : the number of tuples in the relation S)

calculated. When there are several selection and join operations, each size is resulted in previous operation. In the example 3.3, the final size after selection and join operations is estimated to (0.05 * 10000) * (0.2* 200)* 500* 0.0000025 = 2.5 $\approx$ 2.

Example 3.3

Table Schema : R(i,j,k), S(a,b), T(m,n) The number of tuples : R(1000) , S(500), T(200) Query : SELECT i,a FROM R,S,T WHERE j = 3 AND m = 2 AND k = b; The number of tuples that satisfy j = 3 for selection operation is 50. selection selectivity = $\frac{50}{1000} = 0.05$ The number of tuples that satisfy m = 2 for selection operation is 100. selection selectivity = $\frac{100}{500} = 0.2$ The number of tuples that satisfy k = b for the join operation is 25. join selectivity = $\frac{25}{1000*10000} = 0.0000025$

The projection operation is different from selection and join operations. The output sizes of projection operations need bytes for each data type. In general, this information

is stored in the system catalog. The final query size depends on the number of tuples for the results of selection and join operations and the bytes of data type for projection operations. The final size is the sum of number of bytes for all data types in projection operations times the number of tuples after selection and join conditions. In the example 3.3, if bytes for both of i and a are 4, the final size is (4 * 2) + (4 * 2) = 16 bytes.

3.5 Query Result Size Estimation Methods

There are various methods for query result size estimation. We separate these methods into three categories. The first category is the parametric method. The second is histogram and the third is sampling method.

3.5.1 Parametric Method

As we have explained in chapter 2, parametric methods have been studied by Selinger *et al.* [26, 6],Makinouchi *et al.* [22], and Swami and Schiefer [27]. We now describe the idea of the parametric method using uniform distribution assumption [26, 6]. A query result size is estimated as explained in example 3.4. The example explains the number of tuples in query result uses the number of tuples in a table and the number of distinct values for an attribute as inputs.

Example 3.4

Let there be the total number of values in table R is 1000 and the number of distinct values for attribute a in table R is 100.

The total number of values in table S is 2000 and the number of distinct values for

attribute a in table S is 50.

If we have a selection operation : R.a = 20 then this size is estimated that is $\frac{1000}{100} = 10$.

If we have a join operation : R.a = S.a, then this size is estimated that is $\frac{1000*2000}{100} = 20000$.

The main advantage of parametric method is that it can reduce elapsed time because it just need to look at the number of distinct values of each attribute and the number of tuples for each relation. In contrast, it assumes that underlying data is uniformly distributed. This assumption can not guarantee accuracy of size estimation. Therefore, we disregard parametric methods in this thesis.

3.5.2 Histogram Methods

Histogram methods have been studied in many literatures [19, 24, 17, 25] as we explain in chapter 2. We can mainly divide the methods into traditional and advanced histograms. Traditional histograms contain equi-width and equi-height histograms. This method is studied by Kooi, Piatetsky-Shapiro and Charles Connell [19, 24]. Advanced histograms contain maxdiff, compressed, end-biased and v-optimal histograms [17, 25]. The good aspect for histogram methods is it need not consider underlying data distribution. Therefore, we can handle skewed data easily. The disadvantage is that all histogram methods need to sort data and it takes too much time to sort. Moreover, because join operations need to merge histograms for two participating attributes and construct new histogram, it is complex to build histograms. Finally, multidimensional histograms for co-related attributes are difficult to construct.

Many database systems store a huge amount of data in their relations. They have storage and time complexity problems to construct and maintain histogram information. Also most methods need to consider data types that are used in query operations. A solution to solve this problems is histogram construction using sampling methods. There are several sampling methods that can be used with histogram methods. Sampling methods are described in the section 3.5.3. Also we can find that advanced histograms can handle all

data types that are matched in our forth criterion in section 3.3. In the advanced histograms category, the research [25] has proved that v-optimal and maxdiff histograms performed better than compressed histograms. In addition, they have proved maxdiff histogram is the best histogram. Moreover, it is important to reduce construction time. We also suggest maxdiff histogram with sampling method is the best to reduce construction time. The detailed histogram information is described in chapter 4.

3.5.3 Sampling Methods

Sampling methods are the most famous methods in the size estimation field. Sampling methods prevent looking through all the data in each relation. The basic idea behind sampling methods is that each tuple in a relation has the same probability to be selected into sample values. The query result size is estimated by sampled values. The most important issue in the sampling method is to decide stopping point. Sampling methods are classified by the stopping conditions. In general, sampling methods which are used for query size estimation are Simple Random Sampling with and without Replacements and systematic samplings. There are common advantages like below.

- They are simple to implement.
- They do not use stored statistical information.
- They do not depend on underlying data types.
- They do not use the system catalog information.

Sampling methods are basically simple to implement and does not need to consider underlying data types as we explained above. In addition, researches in chapter 2 explain these sampling methods have relatively accurate results. Therefore, sampling methods are very preferred methods and we choose two sampling methods; simple random sampling and systematic sampling. The detail implementation is explained in chapter 4.

3.5.4 Summary

The main problem related to query result size estimation is that it is hard to investigate all attribute values in a relation. To find good estimation methods, several

methods have been suggested. The parametric method is easy to implement but it cannot guarantee the accuracy when data is not evenly distributed. Histogram methods can handle data skew problems. We selected maxdiff histogram method with sampling by all satisfying criteria. Finally, sampling methods are preferred because they have several apparent good aspects such as no extra storage and convenience of implementations. In summary, we choose maxdiff histogram and simple random sampling and systematic sampling methods by considering the accuracy, construction time and underlying data distribution and data types.

Chapter 4

Implementations

In chapter 3, we discussed representative techniques of query size estimation. We can find good methods with advantages and disadvantages in each technique. By analyzing these factors, we can choose methods to implement. Because the purpose of this thesis is to find the best method in environments with varying degree of skew, sizes of tables and so on, we have to select better methods which are more accurate than others. In chapter 3, we can find sampling method is the best with many aspects. Also we know histogram and sampling methods are better than a parametric method because parametric method need to consider data distribution. Therefore, we choose two sampling methods; simple random sampling [21, 11] and systematic sampling [23]; and one histogram method [25]. We have investigated that histogram methods have some restrictions such as underlying data types. We choose maxdiff histogram among all histogram methods to solve this problem. Because all histogram methods need to sort values with a specific parameter, we use sampling method to reduce construction time. In this thesis, we implement all methods using postgresql [3] that is a relational database and open-source database. In this chapter, we first review concepts of each algorithm that we implement. We then describe pseudo codes and procedures. Finally, we suggest examples for each algorithm.

4.1 Simple Random Sampling Method

4.1.1 Overview

Simple random sampling can be subdivided into two methods. They are with and without replacement. The sampling with replacement is called SRSWR [12]. In this method, the tuple which has already been selected can be selected again. The method without replacement is called SRSWOR [12] which can not select same tuples again. In this implementation, we use SRSWR instead of SRSWOR.

Simple random sampling, called adaptive sampling, has been proposed by Lipton and Naughton [21]. This method is based on values sampled so far. They estimate the mean and variance for sample values and stop when stopping conditions are true. The main focus is choosing the stopping condition correctly. The stopping conditions are suggested by Peter J.Haas and Arun N.Swami [11] in Algorithm S2. We begin by defining the notation. In this notation, R is used for table name.

Notation

- N : the cardinality of a table R.
- n : the cardinality of samples R' from R.
- s : the number of tuples that satisfy the input query.
- $\bar{Y}$: population mean that is defined with selectivity for complex query predicates. $\bar{Y} = \frac{\sum_{i=1}^{N} y_i}{N}$ where $y_i = 1$ if the ith tuple in table R satisfies the query;otherwise $y_i = 0$.
- $\hat{\bar{Y}}$: sample mean that is defined with selectivity for complex query predicates. $\hat{\bar{Y}} = \frac{\sum_{i=1}^{n} y_i}{n}$ where $y_i = 1$ if the i th tuple in R satisfies the query;otherwise $y_i = 0$.
- S^2 : population variance $S^2 = \frac{\sum_{i=1}^{N}(y_i - \bar{Y})^2}{N-1}$.
- S_n^2 : sample variance $S_n^2 = \frac{\sum_{i=1}^{n}(y_i - \hat{\bar{Y}})^2}{n-1}$.
- ε : the relative error that is between 0 and 1.
- ψ : sanity bound to prevent oversampling.

- t : the abscissa of the normal curve that cuts off an area α at the tail and α is a risk of error not within the relative error ϵ given.
- β : the fraction of a table size to limit maximum sample size.

This method has three subconditions to stop sampling. These conditions are based on the variance and mean for sample values which are updated whenever a sample is acquired. With updated mean and variance, we apply the stopping conditions like below.

Stopping Conditions

1. n ≥ 1 : the number of sample values is greater than 0.
2. $S_n^2 > 0$: the sample variance is greater than 0.
3. ϵ max(s,nψ) $\geq$ t(nS_n^2)$^{1/2}$

   This condition finds the number of tuples that have to be sampled within relative error max(s,nψ). It assumes when sample size n is large, it follows from the Standard Central Limit Theorem. It prevents to exceed error bound by a new sample value. The reason why it is a better method than previous similar methods where parameters for stopping conditions are decided before starting sampling procedures is the values in parameters are decided by the mean and variance whenever a sample is obtained. The condition follows Algorithm S2 by Peter J.Haas and Arun N.Swami [11]. It assumes that when sample sizes are large, the distribution for samples follows the normal distribution. This condition means we take sample values within reliable areas in the normal distribution. A sanity bound(ψ) is proposed to prevent oversampling problem. The meaning of oversampling is that the size of selected sample(n) is too large. For example, if the number of values(s) that satisfy the query condition is too small, the sample size n can reach the allowed maximum size of sample (by β value). In this case, the oversampling problem occurs. The solution for this problem is applying sanity bound ψ in max(s,nψ). The value ψ restricts the sample size by terminating this algorithm.

4.1.2 Assumptions

In order to get tuples which are selected from simple random sampling, we have to know the physical position of each record on the disk. That means that a tuple is

stored in a physical block and offset in this block. Postgresql [3] checks the block number whenever a tuple is acquired by sequential scan and increments the number of tuples. When a block number is different from previous block number, we assume that one block contains the number of tuples that is obtained so far. If the block number is not changed even if sequential scan is reached at the end of table, we assume the table size is so small and we do not need to sample tuples because it is enough to consider all tuples for size estimation. In this case, we take all values in the table and estimate selectivities within these values.

4.1.3 Size Estimation for Selection Operation

As we discussed earlier, the size is based on selectivity estimation. We estimate selectivities for sample values and then estimate cardinality for whole table. We take sample values until stopping conditions which are discussed earlier become true. Selection operations contain five predicates ($=$, $>$, $<$, $\leq$, $\geq$). Whenever we take a sample value, we compare this value and constant value in selection operation by the given predicate and $\boldsymbol{n}$ is incremented by 1. If the sample value satisfies the condition, $\boldsymbol{s}$ is incremented by 1. We have two options to determine whether we use this sampling method. If the number of tuples in a table is small enough to sequentially scan whole table, we do not process simple random sampling. This decision factor is that if all tuples in a table are in one block, we scan all tuples in the table instead of using sampling method. Otherwise, we start to take sample values. The simple random sampling uses updated $\boldsymbol{s}$ and $\boldsymbol{n}$ values to calculate new variance S_n^2 whenever new sample is obtained. The updated variance is used for decision of the point to stop. When the sampling is terminated by any of the stopping conditions, we estimate the final selectivity by $\frac{s}{n}$. This final selectivity is applied to the size of a actual table to approximate the size of a given query. If the selectivity at the end is $\frac{1}{2}$ and the cardinality for a table in query is 1000, the final query size is approximately 500. Figure 4.1 is a pseudocode for selection size estimation. From line 2 to 9, we explain the selectivity estimation if table size is small enough to sequentially scan whole table. From line 11 to 29, simple random sampling is processed until stopping conditions are true. In line 29, if the number of tuples in samples are greater than or equal to the maximum number of tuples, the sampling procedure is stopped. If updated variance S_n^2 satisfies the condition in line 25, the sampling procedure is stopped where variance S_n^2 is $\frac{w}{n-1}$ in line 21 and $w = \frac{(n-1)}{n}w + \frac{(s-ny)^2}{(n+1)n}$ in line 19.

```
1  if all tuples are in one block then
2        repeat
3           Sequentially Scan a whole table
4           n++;
5           if the tuple satisfies query predicate then
6              s++;
7           endif
8        until the value in the table is NULL
9        return s/n
10 else
11       repeat
12          y = 0;
13          z = randomly sampled value;
14          if the z th attribute satisfies query then
15             s++;
16             y = 1;
17          endif
18          n = n + 1;
19          w = ((n-1)/n) w + (s-ny)^2/((n+1)n)
20          if n > 1 then
21             S_n^2 = w/(n-1)
22          else
23             S_n^2 = 0
24          endif
25          if S_n^2 > 0 and ε max(s,nψ) ≥ t(nS_n^2)^(1/2)
26 then
27             return s/n
28          endif
29       until n ≥ ⌈(β * N)⌉
30 endif
```

Figure 4.1: Pseudo Code for selectivity for selections

Example 4.1

The figure 4.1 illustrates when simple random sampling is stopped.

Query Q : select i from R where i = 5.

Size of table R : 10000.

The fraction $\beta = 0.1$. By this fraction, the maximum sample size is limited to 1000.

The relative error $\varepsilon = 0.1$.

The sanity bound $\psi = 0.1$.

For the value t, we select α is 0.05. By the t table for standard normal distribution, we can pick t value to 1.645.

No.	value	n	s	y	w	S_n^2	ε max(s,nψ)	$t(nS_n^2)^{1/2}$
1	5	1	1	1	0	0	0.01	0
2	13	2	1	0	0.167	0.167	0.01	0.949
3	9	3	1	0	0.194	0.097	0.01	0.883
4	5	4	2	1	0.346	0.115	0.02	0.758
5	40	5	2	0	0.41	0.103	0.02	1.178
6	21	6	2	0	0.347	0.069	0.02	1.061
7	5	7	3	1	0.583	0.097	0.03	1.355
8	5	8	4	1	0.73	0.104	0.04	1.5
9	5	9	5	1	0.827	0.103	0.05	1.584
10	5	10	6	1	0.889	0.099	0.06	1.629
11	5	11	7	1	0.929	0.093	0.07	1.663
12	41	12	7	0	1.166	0.106	0.07	1.856
13	5	13	8	1	1.213	0.101	0.08	1.885
14	5	14	9	1	1.245	0.096	0.09	1.907
15	5	15	10	1	1.266	0.09	1.00	1.911
16	5	16	11	1	1.277	0.085	1.1	1.918
17	5	17	12	1	1.284	0.08	1.2	1.918
18	5	18	13	1	1.286	0.076	1.3	1.925
19	5	19	14	1	1.284	0.071	1.4	1.91
20	5	20	15	1	1.28	0.067	1.5	1.9
21	5	21	16	1	1.274	0.0637	1.6	1.9
22	5	22	17	1	1.266	0.06	1.7	1.89
23	5	23	18	1	1.256	0.057	1.8	1.88
24	5	24	19	1	1.245	0.054	1.9	1.87

Table 4.1: Selection Selectivity Example for Simple Random Sampling

As we can see in the table 4.1 in example 4.1, ε max(s,nψ) is 1.9 and $t(nS_n^2)^{1/2}$ is 1.87. By the line 25 in figure 4.1, the stopping condition is satisfied because 1.9 is greater than 1.87. After terminating this algorithm, we get the selectivity as $\frac{19}{24}$. This example is terminated

very early rather than looking through most of tuples in the table R. The reason is the value 5 appears 19 times among 24 tuples. It means data is skewed and the variance is getting smaller. Therefore, we can meet the stopping condition earlier than the number of tuples that satisfy query condition is small.

4.1.4 Join Selectivity Algorithm

We assume the number of tables that are used in each join condition is two. For example, a join condition has the format of "R.a = S.i". The total number of tuples (n) in the sample is approximated by the product of the number of tuples in each table. The number of tuples (s) that satisfy the join condition is accumulated whenever each sample value is acquired. For example, if the sample sizes are 100 and 200 for table r and s respectably and the number of tuples that satisfy the join condition is 15, the final selectivity is approximated by $\frac{15}{100*200}$; that is 0.00075. The join selectivity algorithm follows same stopping conditions with selection conditions in section 4.1.3. In figure 4.2, line 2-8 illustrate the estimation procedures for small table sizes. From line 10 to 20, simple random sampling procedures are processed. Whenever we get sample values, $\boldsymbol{s}$ and $\boldsymbol{n}$ are updated. The selectivity and variance are thereby changed by updated $\boldsymbol{s}$ and $\boldsymbol{n}$ where variance S_n^2 is estimated by $\frac{n}{n-1}\hat{\mu}(1-\hat{\mu})$ (line 9-21).

Example 4.2

Query Q : select i from R,S where R.i = S.j.

Table Sizes of R : 10000 , S : 15000.

The fraction $\beta = 0.1$. By this fraction, the maximum sample sizes are 1000 for table R and 1500 for table S.

The relative error $\varepsilon = 0.1$.

The sanity bound $\psi = 0.1$.

For the value t, we select α is 0.05. By the t table for standard normal distribution, we can pick t value to 1.645.

1 **if** all tuples are in one block **then**
2 **repeat**
3 sequentially scan two whole tables that participate in join condition
4 n = $N_{R'_1} * N_{R'_2}$;
5 s = $\sum_{i_1=1}^{N_{R'_1}} \sum_{i_2=1}^{N_{R'_2}} (t_{i1} \bowtie t_{i2})$;
6 $\hat{\mu} = \frac{s}{n}$
7 **until** the values in two tables are NULL
8 **return** $\hat{\mu}$
9 **else**
10 **repeat**
11 obtain random sample value from R_1 and R_2
12 s = $\sum_{i_1=1}^{N_{R'_1}} \sum_{i_2=1}^{N_{R'_2}} (t_{i1} \bowtie t_{i2})$;
13 n = $N_{R'_1} * N_{R'_2}$;
14 $\hat{\mu} = \frac{s}{n}$;
15 $S_n^2 = \frac{n}{n-1}\hat{\mu}(1-\hat{\mu})$;
16 **if** $S_n^2 > 0$ and ε max(s,nψ) $\geq t(nS_n^2)^{\frac{1}{2}}$ **then**
17 **return** $\hat{\mu}$;
18 **endif**
19 **until** any of $N_{R'_i} \geq \lceil(\beta * N_{R_i})\rceil$ becomes true, where $i = 1$ and 2
20 **return** $\hat{\mu}$
21 **endif**

Figure 4.2: Pseudo Code for selectivity for joins

The example 4.2 shows how the join selectivity algorithm works. Because the variance for step number 1 and step number 2 is 0, the algorithm is continued. When we reach at 20th step, the value of ε max(s,nψ) becomes greater than the value of t * $(nS_n^2)^{\frac{1}{2}}$. This satisfies the stopping condition at line 19 in Figure 4.2. Thus, this algorithm is stopped and returns the estimated selectivity. In this example, the estimated selectivity whenever new sample is obtained is not small and the variance is gradually incremented. Therefore, the total number of sample values at the end of the sampling procedures is not large. If the estimated selectivity is very small (e.g,0.01), it takes a long time and needs a large number of sampled values to meet the stopping conditions and stopping condition needs the sanity bound (ψ) to solve the oversampling problem.

Step	value in R	value in S	n	s	$\hat{\mu}$	S_n^2	ε max(s,nψ)	t * (nS_n^2)
1	1	1	1	1	1	0	0.1	0
2	1	1	4	4	1	0	0.4	0
3	2	2	9	5	$\frac{5}{9}$	0.28	0.5	2.6
4	1	2	16	8	$\frac{1}{2}$	0.27	0.8	3.4
5	3	2	25	9	$\frac{9}{25}$	0.24	0.9	4.03
6	2	3	36	13	$\frac{13}{36}$	0.24	1.3	4.84
7	1	1	49	19	$\frac{19}{49}$	0.24	1.9	5.64
8	3	2	64	22	$\frac{11}{32}$	0.24	2.2	6.3
9	2	2	81	29	$\frac{29}{81}$	0.24	2.9	7.1
10	3	3	100	33	$\frac{33}{100}$	0.22	3.3	7.72
11	2	1	121	42	$\frac{42}{121}$	0.23	4.2	8.69
12	2	3	144	50	$\frac{25}{72}$	0.23	5	9.46
13	1	1	169	59	$\frac{59}{169}$	0.23	5.9	10.24
14	1	1	196	70	$\frac{70}{196}$	0.23	7	11.03
15	1	1	225	83	$\frac{83}{225}$	0.23	8.3	11.83
16	1	1	256	98	$\frac{98}{256}$	0.24	9.8	12.34
17	1	1	289	115	$\frac{115}{289}$	0.24	11.5	13.7
18	1	1	324	134	$\frac{134}{324}$	0.24	13.4	14.48
19	1	1	361	155	$\frac{155}{361}$	0.25	15.5	15.63
20	1	1	400	177	$\frac{177}{400}$	0.25	17.7	16.12

Table 4.2: Join Selectivity Estimation Example

4.2 MaxDiff Histogram Method

4.2.1 Overview

As mentioned in section 3.5.2, we have several types of histogram methods. Among known histogram methods, we have chosen maxdiff histogram method to implement. This method is based on value and frequency pair which is represented as a MaxDiff. The goal of MaxDiff histogram is to prevent values with vastly different frequencies are in the same bucket. It inserts bucket boundaries between adjacent frequencies. The construction procedures are first sorts values in the table and finds frequencies in each value. We then insert bucket boundaries as many as a given histogram numbers which are defined in the system catalog from the highest frequency differences. In each histogram bucket, we store the number of distinct values, frequency and an average frequency. As the histogram is built with the closest frequencies, the size of query can be estimated more correctly. Because the

histogram method takes too much time to run a whole huge data-set, we have used sampling method to construct. In the following example, we can see the way how to insert bucket boundaries and to make a maxdiff histogram.

Example 4.3

Let X = (4,133), (2,97), (3,89), (5,62), (6,52), (7,43), (1,39), (8,37), (9,12). The first value is value set and the second value is frequency set in each pair.

The the number of bucket boundary in system catalog(β) = 3.

The bucket boundaries are inserted like below.

1	*2,3,4*	*5,6,7,8,9*
39	*97,89,133*	*62,52,43,37,12*

The histogram is constructed like below.

Average Freq.	*Values*
39	*1*
106.3	*2,3,4*
41.2	*5,6,7,8,9*

4.2.2 Random Sampling Algorithm

In this section, we describe a sampling algorithm that is commonly used in selection and join selectivity algorithms. We first determine how many rows have to be sampled. This algorithm is based on the paper by Surajit Chaudhuri [7]. The corollary 1 to theorem 5 in the paper explain that the minimum random sample r = 4 * k * $\frac{ln(2*n/\gamma)}{f^2}$ where k is histogram size,n is table size,f is maximum relative error in bin size and γ is error probability. We take f is 0.5, γ is 0.05 and f is 0.01. The table size n and histogram size k follow system catalog information. We then start sampling procedures by Algorithm Z from Jeff Vitter's paper [28].It works by repeatedly computing the number of the next tuple we want to fetch, which will replace a randomly chosen element of the reservoir (current set of tuples). At all times the reservoir is a true random sample of the tuples we have passed over so far, so when we fall off the end of the relation we are done. The sampling procedures run before starting selection and join selectivity estimation algorithms.

4.2.3 Selection Selectivity Algorithm

Histograms basically take a look at a stored average value in each bucket to estimate query size. The basic idea is that we compare selection operation and starting and ending values in the bucket. When we meet starting and ending values that satisfy selection operation, we pick an average value in this bucket. The final number of tuples that satisfy a given operation is this average value in the bucket. Figure 4.3 is the pseudocode for estimation of selection selectivity. We first determine the number of buckets to construct histogram. In postgresql [3], the reasonable bucket numbers are stored in the system catalog. We take this information to construct histogram. Because the maxdiff histogram is based on frequency difference by sorted values, we should sort sample values by quick sort or mergesort to determine boundaries (line 1). From line 2 to 5, we find frequencies for all distinct values for samples.While we look each value for samples, we can find a distinct value and how many times each distinct value occur. Because maxdiff histogram depends on frequency differences, we find frequency differences between distinct values. In order to insert bucket boundaries, we need to know sorted frequency differences. From the largest frequency differences, we insert a boundary within values for this difference (line 6-9). Then we maintain average frequency, starting and ending values and the number of distinct values for each bucket (line 10-13). Finally, for a given selection operation, we look through starting and ending values in each bucket. If we find these values which satisfy constant values in selection operation, the final selectivity is estimated by accumulated average values.

Example 4.4

Let query = "select i from r where i = 3;"

Sample Values = {1,5,3,1,9,4,3,1,1,10,11,10,11,15,1,3,3,4,1,6}

Histogram Size = 3

Step 1 : Sort Sample Values

{1,1,1,1,1,1,3,3,3,3,4,4,5,6,9,10,10,11,11,15}

Step 2 : Calculate Frequencies

{1,6}, {3,4}, {4,2}, {5,1}, {6,1}, {9,1}, {10,2}, {11,2}, {15,1}

```
1  Sort by Quick or MergeSort method for sample values

2  repeat
3      Find distinct values
4      Find frequencies for all distinct values
5  until sample value is not NULL

6  repeat
7      Calculate differences between distinct values
8  until distinct value is not NULL

9  Sort for the calculated frequency differences

10 repeat
11     insert a bucket boundary between two values for this difference
12     histogram count++;
13 until histogram count doesn't exceed histogram size.
14 repeat
15     find satisfying ranges for selection operation
16 until histogram is not NULL
```

Figure 4.3: Selection Selectivity Pseudo Code

Step 3 : Calculate Frequency Differences

{2,2,1,0,0,1,0,1} ==> Sort by Descending Order: {2,2,1,1,1,0,0,0}

Step 4 : Include Bucket Boundaries and Make a histogram

Average Freq.	*Values*
6	*1*
4	*3*
1.43	*4,5,6,9,10, 11,15*

4.2.4 Join Selectivity Algorithm

The histogram for join selectivity merges histograms for each attribute. The construction mechanism and procedures histogram for each attribute are same with the case of selection operation. The reason why we have to merge histograms is that we need to make same bucket boundaries to improve the accuracy of the estimation. For example, we have a join operation R.i = S.a. In this case, we first make each histogram for R.i and

S.a separately by the same procedures with selection operation. We then merge these histograms and estimate frequencies again. We illustrate pseudocode in Figure 4.3. Line 5 in this figure makes histograms for each attribute for join operation. Then we merge these two histograms from line 6 to 12. We make new starting and ending values so as to have same bucket boundaries (line 12). We estimate new frequencies and the number of distinct values within each bucket in new histogram. There is no guaranteed method to estimate these values well. In our implementation, we used containment assumption. It explains that if the number of distinct values in histogram A is smaller than that of histogram B with same range, all distinct values in histogram A are contained in those of histogram B (line 13-15). We finally calculate the selectivity of join operation. The selectivity is estimated by sum of total frequencies for all buckets. Let R' is the sample size of R, S' is the sample size of S and s is the sum of total frequencies for all buckets. From line 16 to 18, we estimate the number of values that satisfy join operation in sample. Then, we estimate selectivity; that is $\frac{s}{R' * S'}$.

Example 4.5

Let query = "select i,a from R,S where R.j = S.b"

We have histograms for R.j and S.b like below.

Step 1 : Make histograms for attributes in join operation(Table 4.4). This table maintains value ranges and average frequencies.

Step 2 : We align the histogram buckets so that their boundaries are same. For instance, the first bucket in Table 4.4 share starting value but ending value is different. We split the first bucket for attribute j into (1 - 3) and (4 - 6). With same manner, we update histograms like Table 4.5.

Step 3 : Constructs new histogram with new value ranges. The average frequencies in new histogram are estimated by the product of average frequencies in attribute j and b. For example, the average frequencies in the first bucket of new histogram is 2 * 2 from (1 -

```
1  newHis = new histogram structure
2  R' = sample size R, S' = sample size S;
3  s = total size that satisfies operation;
4  n = the product of R' and S';

5  Make Histograms for join attributes

6  repeat
7        newHis = Add bucket
8  until Histogram bucket is NULL for attribute1

9  repeat
10       newHis = Add bucket
11 until Histogram bucket is NULL for attribute2

12 Make new bucket boundaries for new histogram

13 repeat
14       Estimate frequencies for each new bucket
15 until newHis is not NULL

16 repeat
17       s = s + total frequencies
18 until newHist is not NULL

19 return s/n
```

Table 4.3: PseudoCode for join selectivity algorithm

3) bucket in Table 4.5. The Table 4.6 is a final constructed histogram.

Step 4 : Estimate join selectivity

s = (4 * 3) + (12 * 3) + (2000 * 2) + (400 * 1) = 4448.

Sample size R' : 412, Sample size S' : 84.

$\hat{\mu} = \frac{4448}{34608} = 0.129.$

Attribute j		Attribute b	
Ranges	Average Frequencies	Ranges	Average Frequencies
1 - 6 (6)	2	1 - 3 (3)	2
7 - 8 (2)	100	4 - 6 (3)	6
9 - 9 (1)	20	7 - 9 (3)	20

Table 4.4: Histograms for join operations

Attribute j		Attribute b	
Ranges	Average Frequencies	Ranges	Average Frequencies
1 - 3 (3)	2	1 - 3 (3)	2
4 - 6 (3)	2	4 - 6 (3)	6
7 - 8 (2)	100	7 - 8 (2)	20
9 - 9 (1)	20	9 - 9 (1)	20

Table 4.5: Aligned Histograms for join operations

4.3 Systematic Sampling Method

Systematics sampling method is proposed by Harangsri *et al* [23]. Systematic sampling takes tuples with k interval from a relation with size N whose tuples are sorted in ascending or descending order. Because this method is based on sorted values, we do not need to use the method unless underlying data is sorted. If data is not sorted, we have to sort first. However, we can look through all data during sorting procedure. Therefore, we handle only sorted data for this method. Suppose that N tuples in the table are numbered from 1 to N in some order. To select a systematic sample of n samples, if k = $\lceil \frac{N}{n} \rceil$ then every k-th tuple is selected commencing with a randomly chosen number between 1 to k. For instance, N = 2000, n = 200 therefore k = 2000 / 200 = 10. Therefore, every 20th tuple from randomly chosen number 5. Systematic sampling method has a same procedure and code to gather samples and only selectivity estimation is different. We will describe common procedures and codes. Figure 4.4 illustrates the procedures for the systematic sampling. We first calculate a sample size n = β * N. The β means a sampling fraction ($0 < \beta \leq 1$). We then calculate an interval k = $\lceil \frac{N}{n} \rceil$ (line 3). If attributes for selection and join operations are not sorted, sort these attributes. (line 5-7). The first value to be sampled is an initial value within k value (line 8). We continue this procedures until sample size obtained so far is β * N.

Ranges	Average Frequencies
1 - 3 (3)	4
4 - 6 (3)	12
7 - 8 (2)	2000
9 - 9 (1)	400

Table 4.6: Final merged histogram for join operations

```
1  N = table size;
2  β = sampling fraction;
3  k = N/n;
4  i = initial number

5  if(attribute is not sorted)
6        sort tuples in the relation;
7  endif

8  i = a randomly generated number within k;
9  get a i-th tuple

10 repeat
11       get tuple with an interval k;
12 until n = β * N
```

Figure 4.4: PseudoCode for systematic sampling

Example 4.6

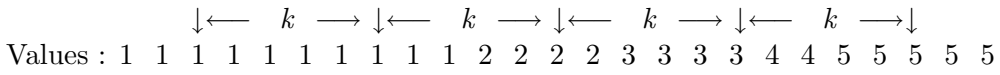

Figure 4.5: Systematic Value Sets

Let query : select i from R where i = 3;

Table Size N : 25.

Sample fraction β : 0.2.

Sample Size n : 25 * 0.2 = 5 by sample fraction.

We can get an interval k by $\frac{25}{5} = 5$.

Initial randomly generated number i : 3.

Based on Figure 4.5, we get sample values by k interval from the initial value.

Then we get sample values ; {1,1,2,3,5}

In this samples values, we can find that the number of value 3 is one.

Then we can estimate selectivity; $\mu = \frac{1}{5} = 0.2$.

Finally, the query size is 0.2 * 25 = 5.

The result is very similar to real value 4.

Chapter 5

Experimental Results

In this chapter, we describe the several experiments that we have conducted to study the accuracy of various methods in estimating query sizes. We evaluate proper methods using various experimental conditions as well as performance.

This chapter starts out with experimental setup with various aspects and move on to the results with implication.

5.1 Notations and Definitions

- **Simple predicate** : A simple predicate on a relation is a condition specified on a single attribute to select the tuples of the relation which satisfy the condition.
- **Complex predicate** : A complex predicate on a relation is formed by any combination of conjunctive and disjunctive simple predicate on the relation.

5.2 Experimental Setup

The algorithms in this thesis were implemented by extending and modifying PostgreSQL version 7.3.4 [3] which are open-source relational data-management system. The tests were performed on a single processor 2.8 GHz Pentium-IV machine with 512MB memory and 80GB hard space running RedHat 9 Linux. The workload for the experiments is

Name	Size
Region	5
Nation	25
Supplier	10,000
Customer	150,000
Part	200,000
PartSupp	800,000
Orders	1,500,000
Lineitem	6,001,215

Table 5.1: TPC-H Tables

based on the TPC-H benchmark at scale of 1 (i.e., 1 GB total size). The schemas and sizes (i.e.,the number of tuples) of TPC-H [4] tables are shown in Table 5.1. We have conducted experiments with following techniques and query types. In each category, we have used at least 30 queries for experiments using integer, varchar, char, decimal and date types.

- Techniques : The parameters for the simple random sampling (Section 4.1) are sanity bound (ψ), relative error (ϵ), t value which is the abscissa of the normal curve off an area α at the tail and α is a risk of error within relative error and maximum sampling fraction (β). In this thesis, we used that ψ, ϵ and β are all 0.1 and t value is 1.645 or 0.1. When α value is 0.05 and query sizes are infinite, t value is 1.645. Otherwise, if query sizes are less than 100, t value is 0.1. The parameters for the maxdiff histogram with random sampling method (Section 4.2) are histogram size k, maximum relative error in bin size f and error probability γ to decide minimum random sample size. In this thesis, k depends on system catalog in PostgreSql that is target number of histogram bins to create. f value is 0.5 and γ value 0.01. The parameter for the systematic sampling (Section 4.3) is the sampling fraction (β). β is set to 10% for each relation participating in the selections and joins.

- Table Sizes : Several different table sizes are modeled using TPC-H tables. As we can see in the Table 5.1, table sizes are various. We divide TPC-H data into small, medium and large size of tables. When the number of tuples for the table are greater than 0 and less than or equal to 10,000, it is considered as a small-size table. When the number of tuples for the table is greater than 10,000 and less than or equal to 800,000, it is considered a medium-size table. When the number of tuples for the

table is greater than 800,000 and less than or equal to 6,001,215, it is considered a large-size table. We used queries that have single and mixed sizes of tables for the experiments. The first query type is small size of table where table names are region, nation and supplier. The second query type is medium size of table where table names are Customer, Part and PartSupp. The third query type is large size of table where table names are Orders and Lineitem. The forth query type is small and medium pair where Region, Nation, Supplier, Customer, Part and PartSupp tables are mixed. The fifth query type is medium and large table pair where Customer, Part, PartSupp, Orders and Lineitem are mixed. The sixth query type is small and large table pair where Region, Nation, Supplier, Orders and Lineitem tables are mixed. The eighth query type is small, medium and large tables are participated where all tables are used. We have conducted 45 times to estimate query sizes.

- Data Distribution : Experiments are conducted using several degrees of skew for the attributes. The degrees of skew for the experiments varied between 0 and 6. The degree of skew is decided by Excel application. The equation for skewness is defined as :

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

  where n is the number of data, x_i is value, $\bar{x}$ is an average value and s is standard deviation.

  We categorize the degrees of skew into four groups in Table 5.2. When the degrees of skew is 0, it means data is uniformly distributed. In contrast, the degrees of skew have high value, it means data distribution is highly skewed. Furthermore, when data is evenly distributed, we can apply one more option. In our experiments, we also separately apply non-key case on non-skew results. We have conducted 30 times to estimate query sizes.

- Selectivities : Experiments are conducted using two different types of selectivities for 45 times. If the selectivity for queries is distributed between [0,0.2], it is considered a low-selectivity. If the selectivity for queries is distributed between [0.8,1], it is considered a high-selectivity. We ignored selectivities between 0.2 and 0.8 because the probabilities to be selected are almost same. We would like to estimate extreme two cases.

Range	Group
0	Non-Skew
(0-1]	Low Skew
(1-2]	Medium Skew
(2-6]	High Skew

Table 5.2: Degrees of Skew

- Query Conditions : In this experiments, query conditions allow simple Selection, join and projection operations. For the accurate result for each case of condition, equality and range query ($>$, $<$, $\geq$, $\leq$) for selection operation and join operations are considered. We have conducted 100 times.

- Sample Sizes : In our experiments, we investigate the accuracy of query result using various sample fractions. We use 5%, 10% and 15% of sample fraction to the number of tuples in a table. In common, we can get more accurate result when we apply more sample values. However, simple random sampling method is adaptively stopped by updated variance by new sample value. Therefore, we do not need to consider simple random sampling for this experiments. We conduct 30 times to estimate query sizes.

- Elapsed Time : We show elapsed time when various table sizes and various sampling fractions are applied. Taken time is different by table and sample sizes. Therefore, we use 10% of sampling fraction to all TPC-H tables to estimate taken time for various table size conditions. Because stopping conditions for simple random sampling do not depend on table sizes, taken time does not based on table size. Therefore, this method is ignored for this experiment. Moreover, 'Nation' and 'Region' tables are omitted because they are too small to estimate to estimate elapsed time. In addition, we use 'Orders' table by 5%, 10% and 15% of sampling fraction to estimate taken time for various sampling fraction conditions. We conduct 30 times to estimate query sizes.

- Mean Relative Error : All experimental results are based on mean relative error. It is defined as following :

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

where S is the number of samplings, μ is the actual selectivity, $\hat{\mu}$ is the estimated selectivity which results from the ith sampling, $\hat{\mu}_i N$ is the result size estimate of the query with the ith sampling and μN is the actual result size of query.

5.3 Experimental Results

5.3.1 Effect of Table Size

Figure 5.1 shows the average relative error between actual and the result of each method as the categories of table sizes. We denote small size tables to S, medium size tables to M, large tables to L and other cases to combinations of S, M and L by experimental setup Section 5.2 in Figure 5.1. Overall, systematic sampling is better than maxdiff and simple random sampling methods. Systematic sampling method shows almost stable results for all cases except small table size. We explain more detail for the specific cases. In the case of small table sizes, maxdiff histogram method shows very accurate result that is approximately 0% relative average error. In other cases, systematic sampling is the best method. In particular, if large size of tables are used, systematic sampling has a great accuracy with a big difference to simple random sampling and maxdiff histogram. In summary, we can use maxdiff histogram when table size is greater than 0 and equal to 10,000. In other cases, it is better to use systematic sampling than other methods.

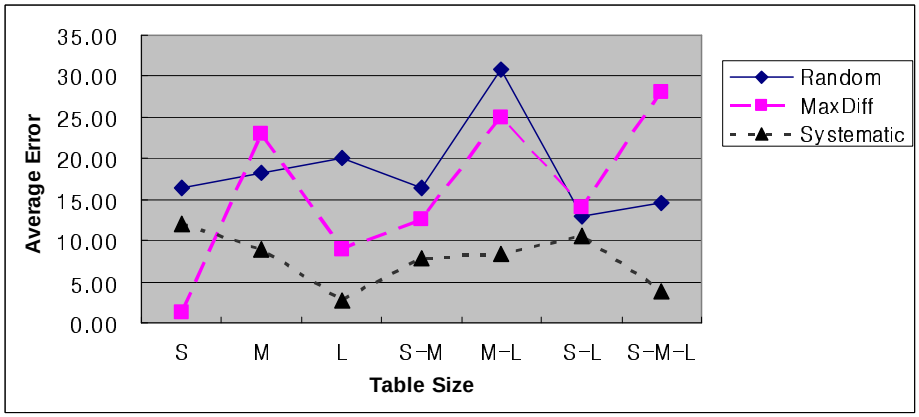

Figure 5.1: Table Size Set Estimation

5.3.2 Effect of Frequency skew

Figure 5.2 shows the average relative error as categories of the degree of skews of frequency set. We denote non skew frequency to Non, low skew to Low, medium skew to Mid and high skew to High by experimental setup Section 5.2 in this figure. Overall, results for all methods depict that the more frequencies are skewed, the more accurate the results are. MaxDiff and systematic sampling methods show similar results which have more accurate results than simple random sampling. Two methods show less than 5% average relative error. When frequencies are highly skewed, all three methods perform essentially no error (i.e.,within 3%). In summary, when data frequencies are highly skewed or not skewed, we can choose any of three methods because the average errors are almost same. On the other hand, simple random sampling is not a good method when data frequencies are low-skewed or medium-skewed.

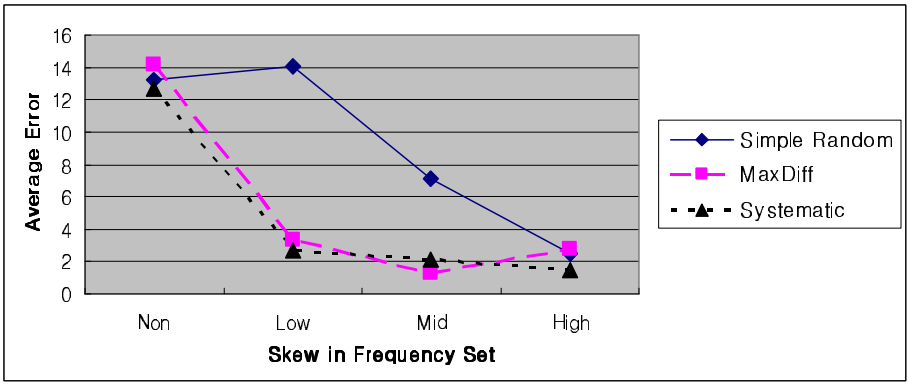

Figure 5.2: Frequency Set Estimation

We show another result for uniformly distributed data. In our experiments, data for this case is almost primary key in each table. Therefore, we suggest another result for non-key values. As we can see in Figure 5.3, we can get different result from Figure 5.2. A reason for the different result is that primary key just has low probability to be selected as a sample because same value does not exist. Moreover, because we use selectivity to predict a result for a whole table, the estimated result is larger than actual result although matching value exist in sample value. Therefore, when we mix primary key and non-primary key for experiments, estimated relative error is worse than the case of non-primary key. When we

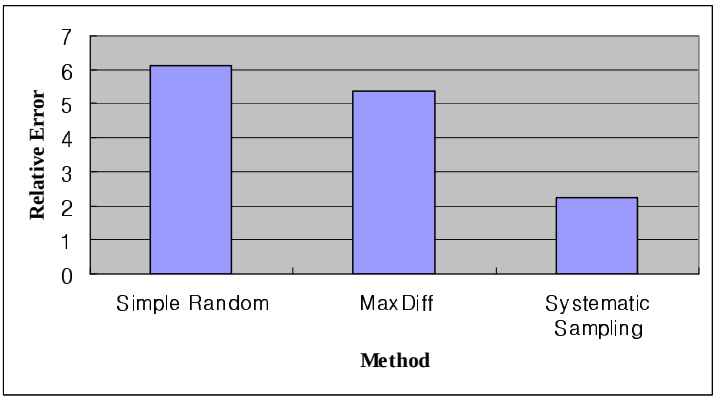

Figure 5.3: Non-Skew Data without primary key

see only values on non-primary key, systematic sampling shows the absolutely most accurate result than those of two methods with a low relative error.

5.3.3 Effect of Selectivity

Figure 5.4 demonstrates the average relative error as categories of selectivities. In the low selectivity environment, Maxdiff histogram and systematic sampling methods have relatively small errors. In contrast,high selectivity environment show opposite results. That means that simple random sampling performs slightly better than systematic sampling and maxdiff histogram. In brief, when a query has a low selectivity, it is better to choose systematic sampling than two other methods. Otherwise, we can choose simple random sampling rather than Maxdiff histogram and systematic sampling although all three methods have low errors.

5.3.4 Effect of query conditions

The average relative errors for various query conditions are plotted in Figure 5.5. For the equal conditions, systematic sampling has shown almost no error. Moreover, MaxDiff histogram has better result than simple random sampling method. The results for the range queries have demonstrated excellent accuracies in three methods. Finally, systematic sampling method has lower relative error than other two methods. The reason for accurate

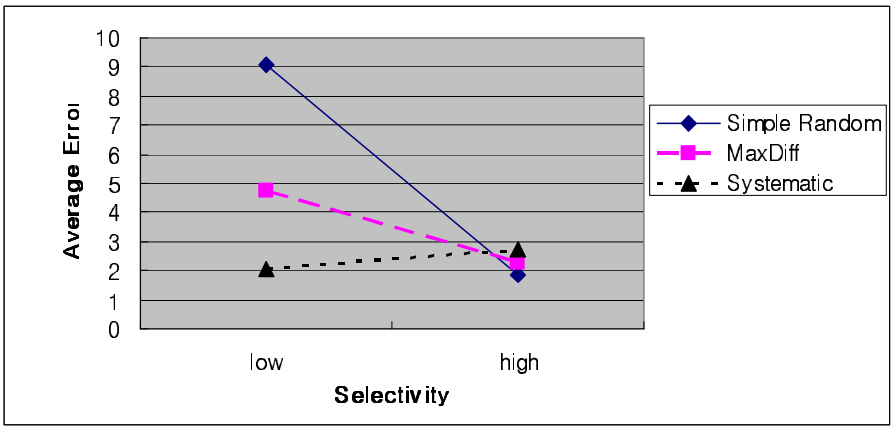

Figure 5.4: Selectivity Estimation

results in range queries is that we have many candidate samples. Therefore, if many same values are in the table, they can be selected with high probabilities. Consequentially, the best method for equal query conditions is systematic sampling. We also choose all three methods for range queries and systematic sampling for join operations.

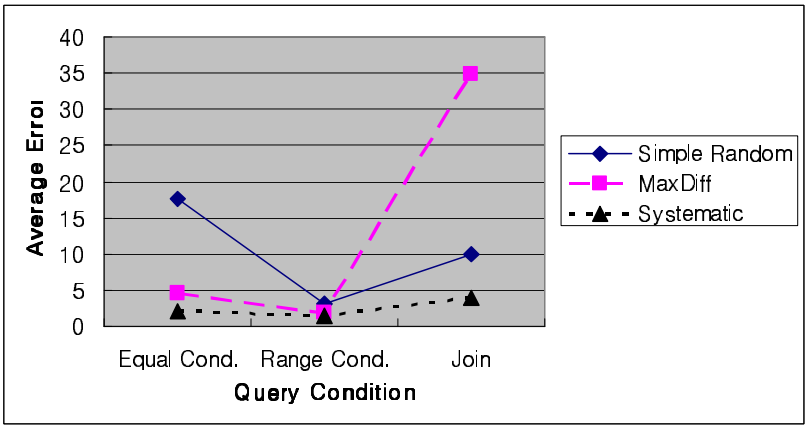

Figure 5.5: Query Condition Estimation

5.3.5 Effect of Sampling Fraction

We applied 5, 10 and 15% of sampling fraction for maxdiff and systematic sampling methods. For this experiments, maxdiff histogram set the number of sample to given sampling fraction instead of using Chaudhuri's algorithm. Maxdiff histogram in Figure 5.6 shows that we can get more accurate result when we apply 10% of sampling fraction than the case of 5%. However, we get worse result when we apply 15% of sampling fraction than the case of 10% of sampling fraction. It shows that it is not a good guideline to use too many sample values. Therefore, appropriate sampling sizes are applied rather than many sample values are used. Systematic Sampling in Figure 5.6 shows whenever we sample many values, we can get more accurate result. Therefore, the number of samples are affected on the accuracy of results.

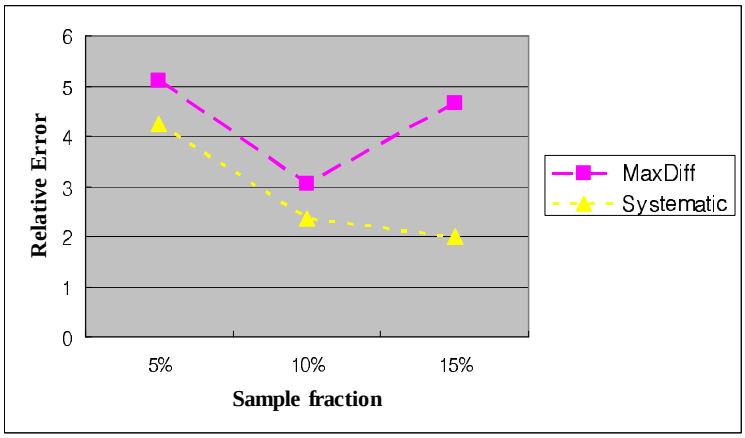

Figure 5.6: Accuracy by Sampling Fraction

5.3.6 Elapsed Time

We can estimate taken time to execute each of three methods by various table and sample sizes. We ignored simple random sampling method for the case of table sizes because it can be terminated when many sample values are matched early. Therefore, maxdiff and systematic sampling methods are estimated. In Figure 5.7, we can know systematic sampling method takes too much time to find specific position for sampled tuple. In contrast, because maxdiff histogram randomly choose sample values, it does not

take much time to sample although elapsed time is exponentially growing when table size are larger.

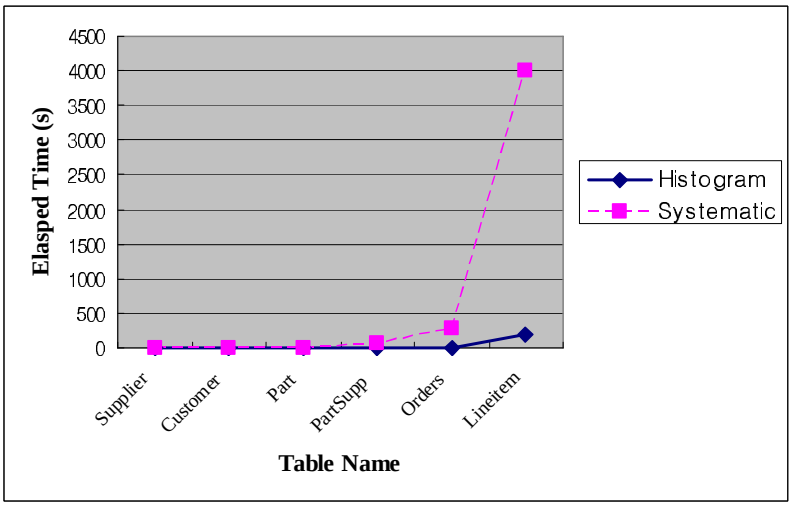

Figure 5.7: Elapsed time by Table sizes

Another experiments is estimation of taken time for sample sizes. Figure 5.8 shows that simple random sampling method takes more time than other methods. When many sample values are taken in simple random sampling, it takes much time to compare and re-calculate mean and variance whenever new sample value is taken. Systematic sampling and maxdiff histogram show same patterns with the case of elapsed time for table sizes.

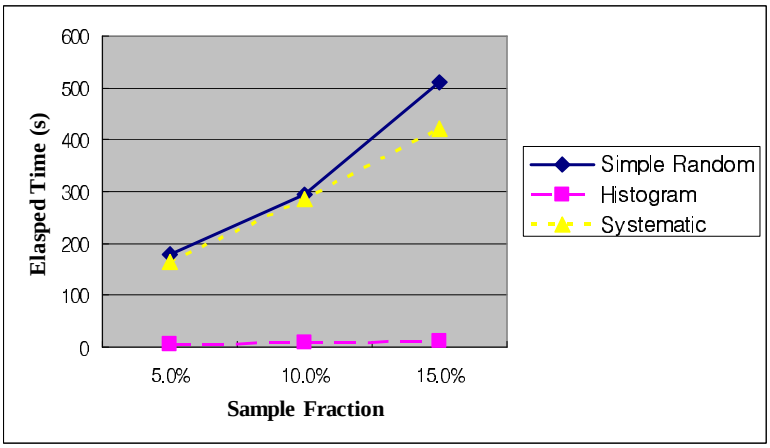

Figure 5.8: Elapsed time by Sample sizes

5.3.7 Summary

We have analyzed the accuracy of given queries by simple random sampling, maxdiff histogram and systematic sampling techniques. The accuracy of queries is based on the difference between actual sizes and size of each method. We estimate query sizes by our implemented methods and actual SQL statement. We then estimate relative error between actual query size and each of three methods. Finally, we compare average relative errors among three our methods. The purpose of our implementations is to suggest guidelines in various query environments. We have proposed these environments by table size, degree of skew, selectivity, query predicates and sample size fractions. Moreover, we estimate taken time to run each of methods by table sizes and sample sizes because elapsed time depends on these two sizes. The category for table size is divided by small, medium and large sizes. We used these sizes separately and combined sizes. As we can see in figure 5.1, we can recommend maxdiff histogram when table size is small where small size is between 0 to 10,000. In other sizes, systematic sampling method gives us relatively better result than other two methods. The second category consists results by the degree of skew. In section 5.3.2, we compare average relative errors among three our techniques. We have non-degree, low, medium and high skews of data frequency set. In table 5.2, we can see the range of the degree of skew that defines each set. As we can see in figure 5.2, we can not guarantee accurate query result size in all three our methods when data is evenly distributed (non-skew). In this case, we mixed values for primary key and non-primary key. Therefore, we can not get accurate result. For this reason, we apply non-key values for non-skewed separately. In this case, systematic sampling, maxdiff histogram and simple random sampling methods show sequentially the more accurate result. However, maxdiff histogram and systematic sampling methods have very accurate results that average relative errors are less than 5%. When the degree of skew of frequencies is high, simple random sampling also can be recommended. The third category is separated by query selectivity. Our experiments in this category are conducted by low and high selectivities. When the selectivity is between 0 to 0.2, it is considered low selectivity. In this case, maxdiff histogram and systematic sampling have a good result. In particular, systematic sampling method has relatively no error. When the selectivity is between 0.8 and 1, all three methods show accurate results. In particular, simple random sampling result shows almost no error. Then, we can recommend simple random sampling in this case. The final category contains various query

conditions with equality and range selection predicates and join operation. When we use systematic sampling and maxdiff histogram methods for equality conditions, we can get accurate results and systematic sampling method has better result than the result of maxdiff histogram. For range queries, all three methods show very accurate results. However, maxdiff histogram is not a good method to recommend for join operation. It has really big difference with actual query sizes. We can also recommend systematic sampling method for join operations. When we apply 5%, 10% and 15% of sampling fraction on maxdiff and systematic sampling methods, we can get more accurate result for 10% of sampling fraction than that of 5% of sampling fraction. However, 15% of sampling fraction shows worse result than 10% of sampling fraction. It means appropriate sampling fraction is needed for maxdiff histogram construction. The more sampling fraction on systematic sampling method shows more accurate results than less sampling fraction. Therefore, the accuracy of systematic sampling depends on sampling fraction. When we investigate taken time by table and sample sizes, the time for systematic sampling is exponentially growing by table and sample sizes. However, the time for maxdiff histogram is slowly growing. Overall, the systematic sampling is the best method in almost of our experimental cases if data is sorted and elapsed time is ignored. The maxdiff histogram method is preferred when table size is small and the degree of frequency skew is (1-2]. Finally, the simple random sampling is preferred when query selectivity is distributed between 0.8 and 1. When data is not sorted, systematic sampling is not a good method because this method look at all tuples in a table. In this case, we can not find a reason to sample because all values in a table are already checked in sorting procedure. Moreover, systematic sampling method is not good if elapsed time is also considered because it takes too much time to find specific positions. By tested results, an acceptable relative error is decided by user requirements.

Chapter 6

Conclusion

Decision whether materialized views will be used for a strategy for executing a query is based on the number of tuples for the query result. If a materialized view is more expensive than other available strategies, the view may not be able to be used to execute a query. When we estimate view sizes, one of the factors is a query size that consists of the view; that is the number of tuples. If the number of tuples are not correctly estimated, consuming CPU and I/O costs may be more expensive than actual costs. Therefore, estimation of the number of tuples is a basic factor to be considered.

In this thesis, we presented three approaches for estimation of query result sizes. They are simple random sampling, maxdiff histogram and systematic sampling methods. We have used efficient methods for query size estimation in the current researches and we evaluated the accuracy of these methods in several environments. The results can be used as guidelines to choose the best method in each condition.

Our experimental results show that systematic sampling is the absolutely most accurate and stable method. When we consider taken time and sorting procedure, systematic sampling method has to be ignored because it takes too much time to choose specific positions by table and sample sizes. In addition, we can look at all tuples when sorting procedure is applied. Therefore, we can choose the best method without systematic sampling method in this case. We can summarize the recommended methods in each category when sorting procedure and taken time are ignored as follows :

- Size : We can recommend MaxDiff histogram with sampling if the table sizes are small where small is between 0 and 10,000. Otherwise, systematic sampling is preferred with approximately 10% average relative error.

- Degree of skew : In this category, results show promising differences by degrees of skew. Overall, we can choose any of the three methods if the degree of skew is high or none where high is between 2 and 6 and none is between 0 and 1. In other cases of estimations, MaxDiff histogram and systematic sampling methods are preferred.

- Selectivity : We can recommend systematic sampling if query selectivities are distributed between 0 and 0.2 and simple random sampling if selectivities are distributed between 0.8 and 1.

- Condition : We can recommend systematic sampling in all kinds of query conditions. In specific, we can select any of three methods in range queries because all methods have slight differences and there are essentially no error.

- Sampling fraction : We can recommend systematic sampling in all 5, 10 and 15% of sampling fractions. Maxdiff histogram shows appropriate sampling fraction is needed because the accuracy of result does not depend on sample sizes.

Bibliography

[1] Microsoft SQL. http://www.microsoft.com/sql/.

[2] Oracle. http://www.oracle.com.

[3] PostgreSql. http://www.postgresql.org.

[4] TPC-H. http://www.tpc.org.

[5] Elena Baralis, Stefano Paraboschi, and Ernest Teniente. Materialized views selection in a multidimensional database. In *The VLDB Journal*, pages 156–165, 1997.

[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. *Commun. ACM*, 24(10):632–646, 1981.

[7] Surajit Chaudhuri, Rajeev Motwani, and Vivek Narasayya. Random sampling for histogram construction: how much is enough? In *Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, pages 436–447. ACM Press, 1998.

[8] Fa-Chung Fred Chen and Margaret H. Dunham. Common subexpression processing in multiple-query processing. *IEEE Trans. Knowl. Data Eng.*, 10(3):493–499, 1998.

[9] S. Christodoulakis. Implications of certain assumptions in database performance evauation. *ACM Trans. Database Syst.*, 9(2):163–186, 1984.

[10] Ramez Elmasri and Shamkant B. Navathe. *Fundamentals of Database Systems.* Addison-Wesley, 3 edition, 7 1999.

[11] Peter J. Haas and Arun N. Swami. Sequential sampling procedures for query size estimation. In *Proceedings of the 1992 ACM SIGMOD international conference on Management of data*, pages 341–350. ACM Press, 1992.

[12] Banchong Harangsri. *Query Result Size Estimation Techniques in Database Systems.* PhD thesis, 1998.

[13] Venky Harinarayan, Anand Rajaraman, and Jeffrey D. Ullman. Implementing data cubes efficiently. In *Proceedings of the 1996 ACM SIGMOD international conference on Management of data*, pages 205–216. ACM Press, 1996.

[14] Wen-Chi Hou and Gultekin Ozsoyoglu. Statistical estimators for aggregate relational algebra queries. *ACM Trans. Database Syst.*, 16(4):600–654, 1991.

[15] Wen-Chi Hou, Gultekin Ozsoyoglu, and Baldeo K. Taneja. Processing aggregate relational queries with hard time constraints. *SIGMOD Rec.*, 18(2):68–77, 1989.

[16] Yannis E. Ioannidis. Universality of serial histograms. In *Proceedings of the 19th International Conference on Very Large Data Bases*, pages 256–267. Morgan Kaufmann Publishers Inc., 1993.

[17] Yannis E. Ioannidis and Viswanath Poosala. Balancing histogram optimality and practicality for query result size estimation. In *Proceedings of the 1995 ACM SIGMOD international conference on Management of data*, pages 233–244. ACM Press, 1995.

[18] Younkyung Cha Kang. *Randomized algorithms for query optimization.* PhD thesis, 1991.

[19] Robert Philip Kooi. *The 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 the Eleventh International Conference on Data Engineering*, pages 532–539. IEEE Computer Society, 1995.

[21] Richard J. Lipton, Jeffrey F. Naughton, and Donovan A. Schneider. Practical selectivity estimation through adaptive sampling. In *Proceedings of the ACM SIGMOD 1990 Conference*, 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. *Distrib. 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 the 1984 ACM SIGMOD international conference on Management of data*, pages 256–276. ACM Press, 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 the ACM SIGMOD 1996 Conference*, pages 294–305, 1996.

[26] P. Griffiths 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 the 1979 ACM SIGMOD international conference on Management of data*, pages 23–34. ACM Press, 1979.

[27] Arun Swami and K. Bernhard Schiefer. On the estimation of join result sizes. In *Proceedings of the 4th international conference on extending database technology on Advances in database technology*, pages 287–300. Springer-Verlag New York, Inc., 1994.

[28] Jeffrey S. Vitter. Random sampling with a reservoir. *ACM Trans. Math. Softw.*, 11(1):37–57, 1985.

[29] 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.

Appendix A

A.1 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

Column Name	Data Type
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

Column Name	Data Type
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

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

CUSTOMER Table

Column Name	Data Type
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

Column Name	Data Type
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

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

REGION Table

Column Name	Data Type
REGIONKEY	identifier
NAME	identifier
COMMENT	variable text, size 152
Primary Key	REGIONKEY

LINEITEM Table

Column Name	Data Type
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

A.2 Queries for Experiments

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

A.2.1 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.regioney = 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 linenumber from   lineitem where  linenumber = 4;

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

select orderstatus from   orders where  orderstatus <= 'O';

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.linenumber from   lineitem l,customer c
where l.linenumber >= 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.suppkey,l.returnflag
from   partsupp ps,lineitem l where ps.suppkey = l.suppkey;

select * from   partsupp ps,lineitem l
where  ps.availqty > 1000000 and l.linenumber < 4
and    ps.suppkey = l.suppkey;
```

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.suppkey, l.linenumber
from   lineitem l,supplier s,customer c
where  l.linenumber = 3 and s.nationkey = c.nationkey
and    l.suppkey = s.suppkey;

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.suppkey = '5311'
and    ps.suppkey = s.suppkey;

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

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

A.2.2 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 suppkey,acctbal from   supplier where acctbal = 4641.48;

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

```
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.suppkey from supplier s,part p,partsupp ps
where  ps.suppkey = s.suppkey 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 linenumber = 6;

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

A.2.3 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 linenumber < 7
and    mfgr > 'Manufacturer#1' and c.mktsegment >= 'AUTOMOBILE';

select * from   orders,lineitem
where  linenumber > 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';
```

A.2.4 Query Conditions

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

1. Equality Conditions

```
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  linenumber = 3;
```

2. Range Conditions

```
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

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

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

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.suppkey = s.suppkey;

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.suppkey = ps.suppkey;

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

A.2.5 Sampling Fractions

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

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

select * from   supplier where  nationkey = '21';

select * from partsupp where suppkey = '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;
```