# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST OF TABLES</td>
<td>6</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>7</td>
</tr>
<tr>
<td>CHAPTER</td>
<td></td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>9</td>
</tr>
<tr>
<td>1 INTRODUCTION</td>
<td>11</td>
</tr>
<tr>
<td>1.1 Problem Description</td>
<td>11</td>
</tr>
<tr>
<td>1.1.1 How Correlations Affect the GROUP BY Estimates</td>
<td>12</td>
</tr>
<tr>
<td>1.1.2 A Simple but Impractical Solution</td>
<td>14</td>
</tr>
<tr>
<td>1.2 Major Contributions of This Thesis</td>
<td>15</td>
</tr>
<tr>
<td>1.3 Computing the Covariance between a Pair of Group-wise Estimates</td>
<td>16</td>
</tr>
<tr>
<td>1.4 Providing Correlation-Aware Simultaneous Confidence Bounds</td>
<td>18</td>
</tr>
<tr>
<td>1.5 Returning Top-k Groups with High Probability</td>
<td>19</td>
</tr>
<tr>
<td>1.6 Organization</td>
<td>21</td>
</tr>
<tr>
<td>2 RELATED WORK</td>
<td>22</td>
</tr>
<tr>
<td>2.1 Database Sampling</td>
<td>22</td>
</tr>
<tr>
<td>2.1.1 Obtaining and Maintaining Random Database Samples</td>
<td>22</td>
</tr>
<tr>
<td>2.1.2 Sampling-Based Estimation</td>
<td>25</td>
</tr>
<tr>
<td>2.1.2.1 Selectivity and size of join estimation</td>
<td>25</td>
</tr>
<tr>
<td>2.1.2.2 Approximate query processing</td>
<td>29</td>
</tr>
<tr>
<td>2.1.3 Comparison to Survey Sampling</td>
<td>34</td>
</tr>
<tr>
<td>2.2 Top-k Query Processing in Relational Databases</td>
<td>35</td>
</tr>
<tr>
<td>2.2.1 Top-k Query Processing for SELECT-PROJECT-JOIN Queries</td>
<td>36</td>
</tr>
<tr>
<td>2.2.2 Top-k Query Processing for GROUP BY Queries</td>
<td>40</td>
</tr>
<tr>
<td>2.3 Related Work in Statistics Literature</td>
<td>41</td>
</tr>
<tr>
<td>2.3.1 Multiple Inference</td>
<td>41</td>
</tr>
<tr>
<td>2.3.2 Error Guarantees for Monte-Carlo Methods</td>
<td>46</td>
</tr>
<tr>
<td>3 PRELIMINARIES</td>
<td>49</td>
</tr>
<tr>
<td>3.1 Estimators and Their Properties</td>
<td>49</td>
</tr>
<tr>
<td>3.2 Correlations</td>
<td>51</td>
</tr>
<tr>
<td>3.3 Confidence Intervals and Confidence Regions</td>
<td>52</td>
</tr>
<tr>
<td>4 COVARIANCE COMPUTATION</td>
<td>54</td>
</tr>
<tr>
<td>4.1 Covariance Analysis</td>
<td>54</td>
</tr>
<tr>
<td>4.1.1 Analysis for SUM over Two Database Tables</td>
<td>54</td>
</tr>
<tr>
<td>4.1.2 Analysis for SUM over Multiple Database Tables</td>
<td>58</td>
</tr>
</tbody>
</table>
4.1.3 An Unbiased Covariance Estimator ................................. 59
4.2 Computational Considerations for Covariances .................... 60
4.3 Experimental Evaluation ............................................. 65
  4.3.1 Experimental setup ............................................. 65
  4.3.2 Results ....................................................... 69
  4.3.3 Discussion ................................................... 69
4.4 Conclusion .......................................................... 70

5 PROVIDING CORRELATION-AWARE SIMULTANEOUS CONFIDENCE
  BOUNDS ........................................................................ 71
5.1 The SI Problem and GROUP BY Queries .............................. 71
  5.1.1 GROUP BY and Multiple Inference ............................ 71
  5.1.2 The SI Problem .................................................. 73
5.2 Applying the Sum Inference Problem to GROUP BY Queries .... 76
5.3 Solving the SI Problem ................................................ 77
  5.3.1 A Solution Using Monte Carlo Re-sampling .................. 77
  5.3.2 Solving the SI Problem Using Moment Analysis ............. 80
  5.3.3 Solution Using Approximate Moment Analysis ............... 85
5.4 Experimental Evaluation .............................................. 88
  5.4.1 Running Time Experiments ..................................... 89
    5.4.1.1 Experimental setup ....................................... 89
    5.4.1.2 Results ................................................... 91
    5.4.1.3 Discussion ................................................ 91
  5.4.2 Correctness Experiments ......................................... 94
    5.4.2.1 Experimental Setup ....................................... 94
    5.4.2.2 Results ................................................... 95
    5.4.2.3 Discussion ................................................ 96
5.5 Conclusion .......................................................... 101

6 RETURNING TOP-k GROUPS WITH HIGH PROBABILITY .......... 102
6.1 Introduction .......................................................... 102
6.2 Confidence Region and the Region Top-k Problem ................ 105
  6.2.1 How Confidence Regions may Help Return all Top-k Groups 106
  6.2.2 The Region Top-k Problem .................................... 107
  6.2.3 Applying the Region Top-k Problem to the GROUP BY Top-k Query 108
6.3 Solutions for Returning All Top-k Groups with High Probability 109
  6.3.1 A Solution Using Interval Sweeping .......................... 109
  6.3.2 A Pruning-Based Solution ..................................... 113
  6.3.3 A Two-Stage Algorithm ....................................... 115
6.4 Various Issues for Building Confidence Regions .................. 116
6.5 Experimental Evaluation ............................................. 119
  6.5.1 Running Time Experiments ..................................... 119
    6.5.1.1 Experimental setup ....................................... 119
    6.5.1.2 Results ................................................... 122
<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-1</td>
<td>Two database tables for the example SQL query: PRODUCT and SALES.</td>
<td>14</td>
</tr>
<tr>
<td>4-1</td>
<td>Example results of query 1</td>
<td>62</td>
</tr>
<tr>
<td>4-2</td>
<td>Example results of query 2</td>
<td>62</td>
</tr>
<tr>
<td>4-3</td>
<td>Running time for Covariance Computation under different parameter values</td>
<td>70</td>
</tr>
<tr>
<td>5-1</td>
<td>Running time under different parameter values</td>
<td>92</td>
</tr>
<tr>
<td>6-1</td>
<td>Running time for two extreme cases for three algorithms. The first case is</td>
<td>123</td>
</tr>
<tr>
<td></td>
<td>that the associated $g()$ values are almost identical, and the second case</td>
<td></td>
</tr>
<tr>
<td></td>
<td>is that the associated $g()$ values are highly separable. Both low</td>
<td></td>
</tr>
<tr>
<td></td>
<td>correlation and high correlation combinations for 100 groups are provided,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>and only show the first case with low correlation, the second case with</td>
<td></td>
</tr>
<tr>
<td></td>
<td>high correlation combinations for 1000 and 10000 groups are also provided.</td>
<td></td>
</tr>
<tr>
<td>6-2</td>
<td>Number of returned groups for two extreme cases for three algorithms. The</td>
<td>124</td>
</tr>
<tr>
<td></td>
<td>first case is that the associated $g()$ values are almost identical, and</td>
<td></td>
</tr>
<tr>
<td></td>
<td>the second case is that the associated scores are highly separable.</td>
<td></td>
</tr>
<tr>
<td>6-3</td>
<td>Number of returned groups for the middle case for three algorithms. The</td>
<td>124</td>
</tr>
<tr>
<td></td>
<td>total number of groups is 100. The parameters are chosen so that the number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>of groups returned by the interval sweeping algorithm is approximately 50.</td>
<td></td>
</tr>
<tr>
<td>6-4</td>
<td>Number of times that correctly returns all top-5 groups under 100</td>
<td>126</td>
</tr>
<tr>
<td></td>
<td>independent tries.</td>
<td></td>
</tr>
</tbody>
</table>
LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-1</td>
<td>Three possible relationships between the region and the line ( x_0 = x_1 ): (a) The region is above the line. (b) The region touches the line. (c) The region is below the line. If the true ranking score vector is in the region, in case (a), the top-1 group is the second group because for any points in the region ( x_1 &gt; x_0 ). Similar, in case (c), the top-1 group is the first group. In case (b) we cannot say anything.</td>
<td>20</td>
</tr>
<tr>
<td>3-1</td>
<td>Standard Bivariate Normal(coef=0.5(left) and -0.5(right)).</td>
<td>52</td>
</tr>
<tr>
<td>4-1</td>
<td>The Algorithm for computing ( V_{FF} ).</td>
<td>63</td>
</tr>
<tr>
<td>5-1</td>
<td>A GUI for GROUP BY query.</td>
<td>73</td>
</tr>
<tr>
<td>5-2</td>
<td>Shapes of Beta distribution (left), the empirical estimate of pdf of ( F[k] ) for 100 groups based on 10000 Monte Carlo samples (top right), and the approximation of the pdf of ( F[k] ) using Beta(31.123,66.6907) (bottom right).</td>
<td>82</td>
</tr>
<tr>
<td>5-3</td>
<td>Algorithm to calculate the estimated variance.</td>
<td>88</td>
</tr>
<tr>
<td>5-4</td>
<td>Correctness of three methods over dataset one: All parameters use their default values. The error bar is a 95% two-sided confidence interval from the corresponding binomial distribution ( Binomial(100, p) ), where each ( p ) value is shown in the x axis.</td>
<td>96</td>
</tr>
<tr>
<td>5-5</td>
<td>Correctness of three methods over dataset two: Except for the fact that the sampling ratio is set to 0.5%, three other parameters use their default values.</td>
<td>97</td>
</tr>
<tr>
<td>5-6</td>
<td>Correctness of three methods over dataset three: Except for the fact that the skew is set to 0.3, three other parameters use their default values.</td>
<td>98</td>
</tr>
<tr>
<td>5-7</td>
<td>Correctness of the three methods over data set four: 20 groups and a skew of 0.3 are used.</td>
<td>99</td>
</tr>
<tr>
<td>5-8</td>
<td>Correctness obtained assuming independence on data set four. The right plot shows the difference between the SI solution obtained using independence (the solid line) and the Beta approximation (the dash line).</td>
<td>99</td>
</tr>
<tr>
<td>6-1</td>
<td>Three possible cases for the confidence region, classified by using the relationship between the region and the line ( first-group=second-group ): (a) The region is above the line. (b) The region touches the line. (c) The region is below the line.</td>
<td>106</td>
</tr>
<tr>
<td>6-2</td>
<td>A framework for solving the GROUP BY top-( k ) using a solution to the region top-( k ) problem.</td>
<td>109</td>
</tr>
<tr>
<td>6-3</td>
<td>An example of the interval sweeping algorithm: (a) The starting point. (b) The ending point. The return set is ( {1,2,4} ).</td>
<td>111</td>
</tr>
</tbody>
</table>
Three possible relationships between the region and the line $first-group = second-group$. The region is a region such that each of its edges is parallel to one of the eigenvectors. (a) The region is above the line. (b) The region touches the line. (c) The region is below the line. The dash rectangle is the bounding box.

The Pruning-Based Algorithm

The Two-Stage Algorithm
CORRELATION-AWARE STATISTICAL METHODS FOR SAMPLING-BASED GROUP BY ESTIMATES

By

Fei Xu

August 2009

Chair: Christopher Jermaine
Major: Computer Engineering

Over the last decade, Data Warehousing and Online Analytical Processing (OLAP) have gained much interest from industry because of the need for processing analytical queries for business intelligence and decision support. A typical analytical query may require long evaluation time because analytical queries are complicated, and because the datasets used to evaluate analytical queries are large. One key problem arising from long evaluation time is that no feedback is given until the query is fully evaluated. This is problematic for several reasons. First, this makes query debugging very difficult. Second, the long running time also discourages users to explore the data interactively. One way to speed up the evaluation time is to use approximate query processing techniques, such as sampling. Researchers have developed scalable approximate query processing techniques for SELECT-PROJECT-JOIN-AGGREGATE queries. However, most work has ignored GROUP BY queries. This is a significant hole in the state-of-the-art, since the GROUP BY query is an important type of OLAP query. For example, more than two thirds of the public TPC-H benchmark queries are GROUP BY queries. Running a GROUP BY query in an approximate query processing system requires the same sample to be used to estimate the result of each group, which induces correlations among the estimates. Thus from a statistical point of view, providing estimation information for a GROUP BY query without considering the correlations is problematic and probably misleading. In this thesis, I formally address this problem and provide correlation-aware statistical methods
to answer sampling-based GROUP BY queries. I make three specific contributions to the state-of-the-art in this area. First, I formally characterize the correlations among the groupwise estimates. Second, I develop methods to provide correlation-aware simultaneous confidence bounds for GROUP BY queries. Finally I develop correlation-aware statistical methods to return all “top-$k$” groups with high probability when only database samples are available.
CHAPTER 1
INTRODUCTION

Over the last decade, data warehousing and online analytical processing (OLAP) \cite{46} have gained much interest from industry, since there is a need for processing analytical queries for business intelligence and decision support. Unfortunately, evaluating analytical queries is very time-consuming. Based on the most recent public TPC-H \cite{100} benchmark results, it is clear that evaluating an analytical query can take hours or even days. This long processing time is because of two reasons. First, analytical queries are more complicated than normal database queries. Second, the datasets that are used for evaluating analytical queries are huge. The volume of data for a typical data warehouse is at least several terabytes, and these queries generally require accessing a majority of the dataset.

One key problem arising from such long evaluation time is that no feedback is given until the query is fully evaluated. This is problematic for several reasons. First, this makes query debugging very difficult. Only when a query is fully evaluated does a query designer see the result. If the query is incorrectly formulated, the query designer may need to wait for hours or days to see the result and discover the error. The long running time also discourages users from exploring the data interactively. Users may want to explore the data by iteratively modifying the selection conditions for a query, and watching how the query results vary. This helps in finding interesting aspects of the data. However, the long running time makes interactive data exploration impossible.

1.1 Problem Description

One way to speed up the evaluation time is to use approximate query processing techniques \cite{57, 73}. Unlike traditional query processing techniques \cite{41}, most approximate query processing techniques use some statistics, such as a sample \cite{51, 53, 57, 62, 63, 79}, to answer a query, and only require a small amount of time to provide an approximate result with statistical guarantee. The approximate result is called an estimate. Note that
if the query is approximately evaluated many times, many different estimates may be obtained. These estimates are typically characterized using a random variable [18], which is called an estimator [50, 51, 53, 62, 63]. The estimators and their properties are further discussed in Section 3.1. Most of the statistical guarantees are provided through confidence intervals [18]. A typical statistical guarantee looks like “with 95% confidence, the result is in \([2.33 \times 10^5, 2.42 \times 10^5]\)”. Confidence intervals are further discussed in Section 3.3.

For the purpose of query debugging and data exploration, an approximate result may be sufficient because an approximate result is enough for an user to determine the correctness of the input query. Furthermore, during data exploration, an approximate result may help to identify interesting selection conditions. Once the user is reasonably sure the query is correct or an interesting selection condition is found, he or she can then ask the system to fully evaluate the query.

This thesis is focused on approximate query processing for one particularly important type of query: the GROUP BY query [87]. A GROUP BY query may be viewed as a large set of subqueries asked simultaneously. These subqueries partition the data into groups based on one or more user specified attributes such that each group contains all the records whose grouping attributes have the same values. GROUP BY queries are ubiquitous. More than two thirds of the TPC-H benchmark queries are GROUP BY queries. The fundamental problem addressed in this thesis is that existing online approximate query processing systems are not designed for GROUP BY queries in the sense that these systems provide estimates for each group without considering the correlations [18] among the groups while using the same samples to estimate every group [57, 89]. More information about correlations is given in Section 3.2.

1.1.1 How Correlations Affect the GROUP BY Estimates

All existing online approximate query processing systems use sampling [91]. An approximate query processing system is an online system if the system constantly estimates the query answer and updates the estimates and associated confidence intervals,
along with the actual query processing [57, 73]. Generally the estimates converge to the true query answer and the associated bounds become 0 when the actual query processing terminates [73]. The key problem that existing approaches fail to address is that the same samples are shared to estimate the query answer for each and every group. This is problematic because sharing samples introduces correlations among the group-wise estimates. However, no online approximate query processing system has considered the correlations induced by the sample sharing while providing the group-wise estimates.

To illustrate why correlation must be taken into account, we first consider the query: “What were the total sales per region in 2004?” This query may be written in SQL as:

\[
\text{SELECT SUM(p.COST), s.REGION} \\
\text{FROM PRODUCT p, SALES s} \\
\text{WHERE p.PROD = s.PROD AND s.YR = 2004} \\
\text{GROUP BY s.REGION}
\]

Imagine that the answer to this query is guessed by using random samples from the two database tables shown in Table 1-1. The random samples are used as input into a sampling-based estimator for the answer to the query [49]. In order to apply the simplest estimator, the sample from each of the two tables would be joined, and the result scaled up by the inverse of each sampling fraction. The problem we face is that the same sample of relation \text{PRODUCT} is effectively used to simultaneously answer three queries, one for each group – in this case, there is one group associated with the region \text{Europe}, one with the region \text{USA}, and one with the region \text{Asia}. If the first estimate was inaccurate, then the second and third estimates are likely to be inaccurate as well. Note that each of the three regions described in the database has experienced a sale of the product \text{thingy} in 2004, and \text{thingy} is the most expensive product available. As a result, if \text{thingy} has not been sampled from the \text{PRODUCT} table, it is likely that the estimates for \text{Europe}, \text{Asia}, and \text{USA} will simultaneously be too low. Such correlation may be of great concern if an end-user will make decisions based upon the results of several correlated approximations.
<table>
<thead>
<tr>
<th>PRODUCT</th>
<th>SALES</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROD</td>
<td>PROD</td>
</tr>
<tr>
<td>thingy</td>
<td>thingy</td>
</tr>
<tr>
<td>gadget</td>
<td>widget</td>
</tr>
<tr>
<td>widget</td>
<td>thingy</td>
</tr>
<tr>
<td>dohicky</td>
<td>gadget</td>
</tr>
<tr>
<td>thingy</td>
<td>widget</td>
</tr>
<tr>
<td>gadget</td>
<td>thingy</td>
</tr>
<tr>
<td>dohicky</td>
<td>gadget</td>
</tr>
<tr>
<td></td>
<td>Year</td>
</tr>
<tr>
<td>$10</td>
<td>2004</td>
</tr>
<tr>
<td>$2</td>
<td>2004</td>
</tr>
<tr>
<td>$3</td>
<td>2004</td>
</tr>
<tr>
<td>$4</td>
<td>2003</td>
</tr>
<tr>
<td></td>
<td>Region</td>
</tr>
<tr>
<td></td>
<td>Asia</td>
</tr>
<tr>
<td></td>
<td>Asia</td>
</tr>
<tr>
<td></td>
<td>USA</td>
</tr>
<tr>
<td></td>
<td>USA</td>
</tr>
<tr>
<td></td>
<td>Europe</td>
</tr>
<tr>
<td></td>
<td>Europe</td>
</tr>
</tbody>
</table>

Table 1-1. Two database tables for the example SQL query: PRODUCT and SALES.

A user may ask a query that returns answers for 10 different groups, and receive 10 answers with 90% confidence bounds for each. The user may then assume that since each estimate is correct with 90% probability, that most of the approximations will be correct. However, if the various estimates are strongly correlated, the statistical reality may be that there is a 10% chance that each and every one of the 10 approximations is incorrect. This example clearly shows that we need to take the correlations into account when providing group-wise estimates to users. Specifically, if there is a 10% chance that each and every one of the 10 approximations is incorrect, we must explicitly tell an end user this information when providing group-wise estimates.

1.1.2 A Simple but Impractical Solution

One way to avoid this problem is to use independent samples for each group. But it is easy to argue that using independent samples for each group is not a practical option for real queries. Let us consider a 3-table-join-based GROUP BY query with 1000 groups. In order to provide independent samples to each group, assuming the sample size is fixed, we need to prepare and evaluate 1000 samples, instead of one sample used in the sharing sample solution. The overall time spent to evaluate these samples is approximately 1000 times as large as the running time for the sharing sample solution. The purpose of sampling is to save the query evaluation time. However, taking independent samples increases the overall running time dramatically, which significantly degrades the benefit of sampling. A real query may return thousands or even millions of groups, leading to
even longer query evaluation time. Thus, providing independent samples to each group is not a practical solution. Sharing the samples among groups is the only practical choice, requiring that we take the correlations into account when providing estimates.

1.2 Major Contributions of This Thesis

The main contribution of this thesis is providing correlation-aware methods for answering sampling-based GROUP BY queries in an online approximate query processing system. I make three specific contributions to the state-of-the-art in this area.

First, in order to provide correlation-aware methods for answering sampling-based GROUP BY queries and to solve the problems in the current online approximate query processing system, we need to characterize the correlations among group-wise estimates. The natural way is to compute the covariances among the group-wise estimators. Covariance is used to quantify how a pair of group-wise estimators are correlated. In this thesis, I derive the covariance formula for a pair of group-wise estimators. Unfortunately a straightforward computation of the covariance is too time-consuming. Thus, I develop an efficient algorithm for computing the covariance, and experimentally evaluate the efficiency of the algorithm.

Second, once the covariance computation is complete, the covariance must be used to provide simultaneous confidence intervals that takes correlations among group-wise estimates into account. For the purpose of computing such simultaneous confidence intervals in a principled fashion, I introduce the Sum-Inference (SI) problem, which is used to infer the distribution that the intervals are simultaneously wrong. The SI problem is formally defined in Section 5.1. I also develop an algorithm that uses the SI problem to provide simultaneous confidence intervals for GROUP BY queries, and propose several efficient approximate methods to solve the SI problem. Finally, I experimentally evaluate the efficiency and correctness of the proposed solutions.

Finally, I develop algorithms that are guaranteed to return all top-\(k\) groups with high probability, where “top-\(k\)” refers to the best or most interesting groups according to some
user defined criteria. Correctly returning all top-k groups requires taking into account
correlations. I propose various algorithms for doing this. All of the proposed algorithms
require building a confidence region [33] for the group-wise estimates. A “confidence
region” with confidence $p$ is a random region such that with probability $p$, the query
answer lies within it. I develop algorithms that use several types of confidence regions.
These algorithms are carefully compared via an experimental evaluation.

In the next several sections, each of the problems are described in more detail.

1.3 Computing the Covariance between a Pair of Group-wise Estimates

The very first problem is computing the covariance between a pair of group-wise
estimators. In statistics, the covariance between a pair of group-wise estimators describes
the correlation between these two estimators. A positive covariance means there is a
positive correlation between them. That is, when the estimate of one group is high, the
estimate of the other group is also high, and vice versa. A negative covariance means
there is a negative correlations between the two group-wise estimators, so that when the
estimate of one group is high, the estimate of the other group is low, and vice versa.

The derivation of the covariance formula is nontrivial but not too difficult. The full
derivation will be presented in Section 4.2. The real difficulty is computing the covariance
efficiently. To see why the straightforward way is not efficient, I outline some of the most
important properties of the covariance formula without delivering the detail.

Assume we have a GROUP BY query that involves $k$ base relations: $R_1, R_2, ..., R_k$.
The total number of records in relation $R_i$ is $n_i$. The covariance formula of two estimators
for group $a$ and $b$ has the following two important properties:

1. The formula is a summation of $2^k$ components.
2. Each component takes the form

$$c \sum_{i_1=1}^{n_1} \sum_{i'_1=1}^{n_1} ... \sum_{i_k=1}^{n_k} \sum_{i'_k=1}^{n_k} I(i_1, i'_1, ..., i_k, i'_k) f_a(i_1, ..., i_k) f_b(i'_1, ..., i'_k),$$
where $c$ is a constant, $I(i_1, i'_1, ..., i_k, i'_k)$ is an indicator function for some boolean expression, and $f_a$ and $f_b$ are two functions that compute numerical values among the joined results that are use for aggregation.

In general, the number of relations involved in a GROUP BY query is less than 10. Thus, the total number of components in the overall formula is generally not more than 1000. On the other hand, the computation associated with each component is more difficult. The most straightforward way to compute each of the component is to use nested loops to implement the nested summations. This approach is simple but very time-consuming. For example, assume there are 5 relations, each having 1000 tuples. If we assume that to compute the body of the loop requires 10 CPU cycles, we need $10^{31}$ cycles to compute a covariance. Since no modern computer can process more than $10^{20}$ cycles each second\(^1\), the total amount of time to compute a covariance requires more than $10^{11}$ seconds. Thus, developing an efficient algorithm to compute the covariance is vital.

To do this, a key observation is that for most of the combinations of $i_1, i_2, ..., i_k$, $f_a$ evaluates to 0, because only a small number of these combinations satisfy the join condition, and thus produce a result tuple for aggregation. The same is true for $f_b$. A simple way to retrieve these output tuples is to join the samples from each base relation and group the output tuples based on the grouping attributes. A hash-based implementation may then be used to check whether the boolean expression associated with the indicator function $I(i_1, i'_1, ..., i_k, i'_k)$ is satisfied. Thus, the covariance may be efficiently computed. The detail is given in Section 4.3.

In order to evaluate how efficient the proposed algorithms are, I experimentally evaluate the algorithms on different datasets, which is shown in Section 4.4. The effect of skewness, sampling ratio, number of groups are all evaluated.

\(^1\) A typical PC CPU from intel or AMD is about 2GHZ. Thus in one second, it can run two billion CPU cycles. Even considering the pipeline technique, and a big cluster with 10000 PCs, there is still no way for the system to process $10^{20}$ cycles in a second.
1.4 Providing Correlation-Aware Simultaneous Confidence Bounds

The second problem I consider is how to provide simultaneous confidence bounds to approximately answer GROUP BY queries. Currently, online approximate query processing systems provide one confidence interval for each group separately [57, 89]. This is problematic. A user may ask a query that returns answers for 10 different groups, and receive 10 answers with 90% confidence bounds for each. If the various estimates are independent, the probability of seeing all 10 groups wrong is $0.1^{10} \approx 0$. However, if the various estimates are strongly correlated, it is possible that there is a 10% chance that each and every one of the 10 approximations is incorrect. Regardless of which of these two extreme cases is the true case, the confidence intervals provided by existing online approximate query processing systems are the same. This shows that important information has been ignored.

I would like to extend the traditional guarantee and present a guarantee taking the following form: “With probability $p$, at least $k$ of the $n$ groups are within the ranges $(l_1$ to $h_1)$, $(l_2$ to $h_2)$, ..., and $(l_n$ to $h_n)$, respectively.” The extended guarantee takes the correlations among the group-wise estimates into account. Thus, it provides more information.

Given this sort of the extended guarantee, the key problem is how to compute them. For the purpose of computing simultaneous confidence bounds in a principled fashion, I introduce the *Sum-Inference* (SI) problem.

Instead of computing the simultaneous confidence bounds that satisfy the user’s requirement, the SI problem models how these intervals are simultaneously incorrect. In the SI problem, we are given $k$ normally distributed random variables $N_1, N_2, ..., N_k$, where each random variable models an estimator associated with a particular group. Along with these estimators, a set of $k$ intervals are given, one for each group. The $i$th region is defined to be from $l_i$ to $h_i$. We also attach $k$ penalties $s_1, s_2, ..., s_k$, one for each group. If the $i$th range does not contain the query answer, then a penalty of score $s_i$ is added to the
total penalty. We then define the following random variable:

\[ \zeta = \sum_{i=1}^{k} I(N_i \not\in [l_i, h_i])s_i. \]

In the above expression, \( I \) is an indicator function that returns 1 if the boolean condition is true and 0 otherwise. If we define a distribution function \( F \) such that \( F[k] \) gives the probability that \( \zeta \) evaluates to \( k \), then the SI problem is the problem of inferring \( F \).

The SI problem is the key step to providing simultaneous confidence intervals for GROUP BY queries. Because solving the SI problem exactly is NP-hard, I develop three approximate algorithms:

1. **Monte-Carlo Resampling.** In this method, given the covariance matrix, in order to approximate \( F \), we repeatedly sample from the joint distribution of the groupwise estimators.

2. **Moment Analysis.** This method computes the mean, variance, the lower bound, and the upper bound of the domain of \( F \). Then, an appropriate parametric distribution is chosen to fit \( F \).

3. **Approximate Moment Analysis.** When the covariance matrix is very large, computing the entire covariance matrix may require too much time. Thus a sampling based algorithm is developed to approximate the moments of \( F \) by only computing the variances and sampling entries from the covariance matrix.

In order to test the utility of each of the approximate solutions to the SI problem, I design experiments to evaluate these algorithms. For efficiency, the running time of each of the three methods is computed and compared to the time of evaluating the query over the entire database. Monte-Carlo experiments are used to test whether the three algorithms work correctly in practice.

### 1.5 Returning Top-k Groups with High Probability

The final problem I consider is how to return top-k groups with high probability. Current approximate query processing systems monitor all groups simultaneously [57, 89]. This is problematic for two reasons. First, updating thousands of groups for a real world query simultaneously is time-consuming. Second, returning just a few groups
Figure 1-1. Three possible relationships between the region and the line $x_0 = x_1$: (a) The region is above the line. (b) The region touches the line. (c) The region is below the line. If the true ranking score vector is in the region, in case (a), the top-1 group is the second group because for any points in the region $x_1 > x_0$. Similarly, in case (c), the top-1 group is the first group. In case (b) we cannot say anything.

with respect to a user-defined ranking function is often more than sufficient [76]. In this thesis, I develop algorithms that use a database sample to return top-$k$ groups with high probability.

One way to determine the top-$k$ groups with respect to a user-defined statistic is to use “confidence regions” [33]. A confidence region with confidence $p$ is a random region in the $n$-dimensional space such that, with probability $p$, the query answer lies within it. To see how a confidence region may help us find top-$k$ groups, let us consider a simple case: a query with two groups. We are interested in the top-1 group. There are three cases, shown in Figure 1-1, where $x_0$ and $x_1$ are dimensions for the first and second group, respectively. If the query answer is in the region, it is obvious that in case (a), the top-1 group is the second group because for any points in the region $x_1 > x_0$, thus the ranking score associated with the second group is larger than the ranking score associated with the first one. Similarly, in case (c), the top-1 group is the better group. Only in case (b) is it impossible to determine which group is preferred with high probability.
The confidence region in Figure 1-1 is a cube such that each edge is parallel to one of the axes. A benefit of this sort of region is that to build it, only variances are needed. I develop an *Interval Sweeping* algorithm that works with this type of confidence region.

When all the covariances are available, it is possible to build tighter confidence regions. The interval sweeping algorithm returns too many groups for these regions. To overcome this problem, I develop a *pruning-based* algorithm that takes into account correlations. When the confidence region is tight, this algorithm returns fewer groups than the interval sweeping one. However, building such a confidence region requires knowing all the covariances in advance, which is known to be expensive [105].

The final algorithm I develop combines the benefits of these two algorithms. It runs in two stages. In the first stage, only the variances are needed to build a confidence region. The interval sweeping algorithm is used to return a few candidate groups. Covariances among the candidates are computed to build a tighter confidence region, and the pruning-based algorithm is used to return the final results.

I design experiments to evaluate algorithms accuracy (in terms of the number of groups returned) and efficiency (in terms of the running time), and prepare several different datasets. In particular I test the effect of strength of the correlations among groupwise estimators.

### 1.6 Organization

The rest of this study is organized as follows. Chapter 2 provides a survey of work related to the problems addressed in this thesis. Chapter 3 reviews a few statistical preliminaries that are used in this thesis. Chapter 4 describes the derivation of the covariance formula between a pair of groupwise estimators, and an efficient algorithm to compute the covariances. Chapter 5 discusses the problem of providing simultaneous confidence intervals for GROUP BY queries. Chapter 6 describes various solutions for returning all top-k groups with high probability.
CHAPTER 2
RELATED WORK

This chapter presents a survey of previous work in the literature related to the problems in this thesis. The survey begins with previous work in database sampling [3, 48, 49, 51–54, 56, 57, 62, 63, 69–73, 79, 85, 86, 89]. In database sampling, I discuss both how to obtain database samples [72, 85, 86] and how to use samples to estimate various database statistics [3, 48, 49, 51–54, 56, 57, 62, 63, 69–71, 73, 79, 89]. A brief comparison between survey sampling [91] and database sampling ends the review of database sampling. This chapter then discusses related work in top-

\[ k \] queries [36–38, 67, 76, 84, 95]. Both finding top-

\[ k \] tuples for SELECT-PROJECT-JOIN queries [36–38, 67, 84] and finding top-

\[ k \] groups for GROUP BY queries [76, 95] are discussed. Finally this chapter reviews related work in statistics literature. The review begins with a survey in multiple inference [13, 14, 26, 32, 58–60, 64, 82, 90, 96, 98, 99, 103]. And then it briefly surveys the literature related to error guarantees for Monte Carlo methods [12, 28, 81].

2.1 Database Sampling

The study of sampling has a very long history in databases. Researchers mainly focus on two different (and related) problems:

1. How to obtain and maintain random samples from database tables [72, 85, 86]? 
2. How to use random samples to estimate various database statistics such as selectivity, size of join, and answers to database queries [3, 48, 49, 51–54, 56, 57, 62, 63, 69–71, 73, 79, 89]? 

The work in my thesis falls into the second category. In this section, I first give a brief review of the first category, then move to the second category.

2.1.1 Obtaining and Maintaining Random Database Samples

The most notable early work to obtain random samples from database files is due to Olken et al. [85, 86]. In [85], Olken et al. study the problem of obtaining simple random samples from B+ trees [41]. Olken et al. propose several acceptance/rejection-based [91] sampling methods to obtain random samples from B+ trees without requiring any
additional data structures or indices. In [86], Olken et al. study the problem of sampling from hash files [41]. Olken et al. propose sampling methods for both static and dynamic hash files [41]. For static hash files, they consider open addressing hash files and hash files with a separate overflow chain. For dynamic hash files, they consider both linear hash files [41] and extendible hash files [41].

Besides the research work done by Olken et al., Antoshenkov Gennady [7] propose a sampling method based on pseudo-ranked B+ trees. The B+ trees are augmented so that they contain information that allows calculation of upper and lower ranks satisfying the nested bound conditions for all parent-child pairs—that is, the bound of any child is nested inside of the bound of any parent in the B++ tree. An acceptance/rejection-based sampling method uses this information to speed up the sampling and reduce the rejection rate.

DeWitt et al. [29] propose the extent map sampling for handling skewness in parallel joins [94]. The database table is stored by one or more pages called extents. Records are store in each extent. An in-memory data structure maintains information on how these extents are distributed. The address of a page within an extent can be found by adding an offset to the address of the first page of this extent. In order to obtain a random page, a number \( r \) between 1 and the total number of pages is randomly selected, and the corresponding page is returned. To obtain a random record, once a page is selected, a random record is selected from the page. From statistical point of view, only when the records in the table are pre-randomized is the obtained sample a random sample of the database table. However, this is not discussed in depth in the paper.

These papers all consider how to obtain a small random sample from a database table. The sample itself is small and is not stored and managed on disks. Jermaine et al. [72] propose the geometric file to maintain large on-disk random samples. The algorithm uses the reservoir sampling [102]. It collects samples in an in-memory buffer. The items in the buffer are randomly permuted with and then divided into segments of
geometrically decreasing size. When a buffer flush is required, each segment in the buffer overwrites an on-disk segment of the same size. Segments smaller than a certain size are stored and updated within main memory. The organization of the geometric file avoids many random accesses to the disk because only one random access per segment is required to find the beginning place of the in-place update. Records within a segment are written sequentially.

Besides this pioneering work, Gemulla and Lehner [42] propose several deferred maintenance strategies for disk-based random samples with a bounded size. The approach is based on reservoir sampling. The authors assume that a uniform sample of size $M$ has been computed previously. The incremental strategies are comprised of two phases: a log phase that captures all the insertion to the database table and a refresh phase that uses the log file [41] to update the sample. The authors consider both full logging and candidate logging that only keeps a “sample” of the operations on the dataset. They further develop an algorithm for deferred refresh, which performs only fast sequential I/O operations and does not require any main memory. They experimentally compare the implementation to the geometric file and conclude that when geometric file outperforms their method only when the buffer size is sufficiently large.

More recently, Nath and Gibbons [83] propose a method for maintaining a large random sample on a flash disk [16]. They introduce B-File (Bucket File) to store samples. A B-File contains many buckets. And a sample may be randomly stored in one of the buckets determined by their algorithm. They also introduce the concept of semi-random write, such that the block may be selected randomly, but pages within a block must be written sequentially from the beginning to the end. The algorithm assigns each item in the database a “deletion level”. If the item’s deletion level is higher than the minimum sample deletion level, it is selected as a sample. If the size of the sample exceeds the limit, a bucket with minimum deletion level is deleted. The levels are assigned so that the items
kept is a uniform sample of the database. The authors claim that their implementation is flash-disk friendly, and is three orders of magnitude faster than existing techniques.

2.1.2 Sampling-Based Estimation

There is a long history of research that considers how to utilize samples in the database literature [3, 48, 49, 51–54, 56, 57, 62, 63, 69–71, 73, 79, 89]. Many of these papers are closely related to my research. The first work in this area considers how to estimate the selectivity of a select query and estimate the size of a join [24, 39, 51, 53–55, 62, 63, 79]. Later on, researchers consider estimating the answer of a database query [3, 48, 49, 52, 56, 57, 69–71, 73, 89]. This section gives a thorough literature survey on these topics.

2.1.2.1 Selectivity and size of join estimation

Work on how to utilize samples started by estimating the selectivity of a selection query and the size of join. Initially, in order to use samples to estimate the size of join, researchers recommend either sampling [62, 63] from the product space of the tables involved in a join, or using some index structure to aid in sampling from joins [78, 79]. Later on, researchers develop methods to first sample from each base table. The samples from the base tables are then joined to estimate the size of join [52]. The later approach finally leads to the approximate query processing [48, 49, 57, 70, 73].

Pioneering work on using samples to produce estimates for selectivity and size of join are done by Hou el at. [62, 63]. They propose methods to estimate \( \text{Count}(E) \) using a simple random sample, where \( E \) is an arbitrary relational algebra expression [41]. This includes both selectivity and size of join estimation. Selection is handled trivially by scaling up the simple count with respect to the inverse of the sampling ratio. In order to estimate the join size, they propose a method that samples from the product space of the join tables. Intersection is treated as a special case of join. They also propose an algorithm for cluster sampling. Their estimators are somewhat limited because sampling
from the product space of base tables is not efficient. Furthermore, the variance [18] of the proposed estimator is large.

Lipton and Naughton [78, 79] introduce adaptive sampling to estimate the selectivity of a selection condition and the size of a join query. The adaptive sampling method assumes that any query result can be treated as the union of results from a set of disjoint subqueries. In order to estimate the selectivity of a selection condition or the size of join, these subqueries are sampled randomly. The size of the sampled subqueries are computed and used to estimate the selectivity of a selection condition or the size of join. Adaptive sampling assumes that the maximum size of the subqueries is known in advance. This is generally impossible. Thus, the authors use an upper bound of the maximum size of the subqueries.

Hou et at. [61] propose double sampling to estimate a count query [41] with error guarantees. Their method contains two steps. First, a small pilot sample is taken to determine the actual number of samples needed for the input error bound. Then, the actual samples are taken so as to guarantee adherence to the error bound. They compare their method to adaptive sampling and double sampling. Based on their results, double sampling requires less samples to maintain the same accuracy, compared to adaptive sampling.

In [54], Haas and Swami point out that there is no theoretical guide for choosing the pilot sample size in double sampling. Haas and Swami also find that if the upper bound used in adaptive sampling is loose, adaptive sampling may take too many samples. They propose two sequential sampling algorithms called $S_2$ and $S_3$. $S_2$ decomposes the query result into partitions. It randomly selects partitions and observes the sizes of the partitions one at a time. The algorithm terminates according to a stopping rule that depends on the random observations obtained so far. Authors find that if the partitions are divided into strata of equal size, and if each time when sampling, a partition from each stratum is selected, the performance of the sampling increases. They call this new
algorithm S3. Experimental results show that S3 is the best method for estimating the join size. Both S2 and double sampling performed well, while adaptive sampling is a bit unstable. They also find that their estimator does not perform well when one relation is skewed and the other one is not.

Haas et al. [51] compare various estimators in the literature for estimating the join size. They conclude that for a fixed number of sampling steps, estimators based on index-level sampling are better than estimators based on page-level sampling, and estimators based on page-level sampling are better than estimators based on tuple-level sampling. Furthermore, estimators based on cross-product sampling are better than estimators based on independent sampling. They suggest to use cross-product index-level sampling when an index is available and cross-product page-level sampling when an index is not available.

Haas et al. [53] also study the problem of when it is appropriate to estimate the join size instead of computing the join result directly. They find that if an absolute criterion is used to measure the precision of the estimates, the relative cost of sampling decreases when either the size of the relations increases or the number of relations in the join increases. If a relative criterion is used instead, the relative cost of sampling decreases when the size of the relations increase, and increases when the number of relations increases. They also find that the sampling is more expensive when more tuples are not involved in the join. Finally they find that when the join key is uniformly distributed in all join relations, the sampling cost is low. When the skew in the distribution of the join attribute values is high, the sampling cost is low if the join values in different relations are homogeneously distributed (i.e. high frequent values are always high in each relation, low frequent values are always low in each relation). If the join attribute values in different relations are heterogeneously distributed, the sampling cost is high.

In order to solve the problem that the sampling cost is relatively high when the join values are skewed and heterogeneously distributed, Haas and Swami [55] propose
the $TCM$ algorithm. The algorithm uses augmented frequented value statistics to decrease the cost of sampling. They also provide a single pass algorithm to estimate the augmented frequented value statistics that are needed by the $TCM$ algorithm. They show experimentally on real world data that the cost of sampling using $TCM$ algorithm is relatively cheap even when the join values are skewed and heterogeneously distributed.

Ganguly et al. [39] introduce Bifocal Sampling to overcome the problems in the previous proposed estimators. Based on how many number of tuples have the same join value, the tuples in each relation are grouped into two different groups: spare and dense. The authors then develop different estimators for dense-dense pairs, and sparse-any pairs. The dense-dense estimator only takes samples from th dense tuples of both relation. There are two spares-any estimators, each of them takes samples from sparse tuples in one relation and samples from the whole relation of the other. By separating out the dense-dense part, Bifocal Sampling achieves good accuracy. Experimental results show that Bifocal Sampling outperforms previous estimators.

Chaudhuri et al. [24] study the problem of sampling from the output of the join. Their primary purpose is making the sampling an operator within the database. They focus on how to commute the sampling operator with a single join operator [41]. They establish the conclusion that by simply joining random samples from two base relations, it is not possible to obtain a sample for the output of the join. Thus they focus on the problem of obtaining a sample for the output of the join without completely evaluating the join. They observe that because of the skew in the data, it is better to sample a tuple in one relation depending on how many tuples with the same join value in the other relation. They then classify the problem based on how many relations have indices or statistics for each join value, and propose different sampling algorithms to sample from the join output without fully evaluating the join.
2.1.2.2 Approximate query processing

In late 1990’s, research on sampling-based estimation slowly moved from estimating various statistics to approximately evaluating database queries [2–4, 44, 49, 56, 57, 69–71, 73, 89]. The two representative projects are the AQUA project [3] from Bell Labs, and the CONTROL project [56] from University of California, Berkeley. These projects (especially the CONTROL project) are closely related to my research.

AQUA is an approximate query answering system. The main technical contributions of AQUA include congressional samples for GROUP BY queries [2], join synopses [4], and concise and counting samples [44].

Congressional samples are proposed to approximately answer GROUP BY queries with high accuracy. Acharya et al. [2] find that the problem of the uniform sampling is that some small groups have few records, leading to much poorer accuracy. A congressional samples is a precomputed, biased sample such that each group has a sufficient set of representative records in the sample. Given fixed amount of space and a set of database table columns, congressional samples try to maximize estimation accuracy for all possible GROUP BY queries. The authors also provide one-pass algorithms for constructing congressional samples, and efficient methods to evaluate GROUP BY queries on congressional samples. Of course, the database table columns that are used for grouping must be known beforehand, which significantly limits the usage of congressional samples.

Join synopses [4] are precomputed samples from a small set of distinguished joins used for foreign-key [41] joins in data warehousing. The authors show that they can obtain an estimate for any possible foreign-key join by using the precomputed samples. They also provide optimal strategies to allocate these distinguished joins, assuming the workload information is available. They show experimentally that estimates based on join synopses are better than estimates based on random samples.
The concise and counting samples [44] are two sampling-based statistical summaries. Because the summaries requires much less storage than the random samples, the authors claim that the accuracy of estimates for database queries from the summaries is better than the accuracy of estimates from random samples with the same amount of storage. They also provide fast incremental maintainers algorithms for the two sampling-based statistical summaries.

Unlike the AQUA project, the UC Berkeley CONTROL project provides techniques that worked directly on random samples to answer approximate queries, such as online aggregation [57] and ripple join [49]. These techniques are more closely related to the problems I tackled in this thesis.

Hellerstein et al. [57] first propose online aggregation. Online aggregation is a system that executes analytical queries interactively. At any point of the query evaluation, users are shown the current estimate of the query with confidence bounds [18] for an input confidence. The estimates are constantly updated as time progresses. The user may terminate the query evaluation at any time when he satisfies the estimate of the query. In the system, random samples from input relations are used to provide estimates. The paper also discusses various techniques that are required for online aggregation.

In the context of online aggregation, Hellerstein et al. [57] consider the single-table GROUP BY queries. One problem with sampling-based GROUP BY queries is that the small groups may have few or even no records in the sample. Thus, the accuracies for smaller groups is low, compared with the accuracy for larger groups. The index striding technique is proposed to remedy this. They assume that the records is indexed by the grouping attributes. Then the index can be exploited to sample uniformly in a round-robin fashion (e.g. a tuple from the first group, a tuple from the second group, a tuple from the third group,..., and so on), but deliver the records within each group randomly. Sometimes, in order to increase the accuracies of the small groups, it may
be important to sample small groups more heavily than big ones. Therefore, a weighted round-robin fetching schema may be more desirable.

Haas et al. [52] and Haas [48] further discuss how to provide confidence bounds for analytical queries in online aggregation. The major difference between Haas’s work and previous research is that unlike other methods that samples from the output of the join space, the method Haas proposed only needs to sample from each base relation. The samples are joined estimate the query result.

Based on this work, Haas et al. [49] propose a family of join algorithms called the ripple join. The ripple join takes some samples from one relation, and then takes some samples from the other relation, and then takes some samples from the first relation, and so on. The new samples are then joined with the samples from the other relation. The estimate of the query is updated with each new sample set. The paper also discusses how to provide confidence intervals [18] for the estimate.

The ripple join algorithm converges slowly when the selectivity of the query is low. In order to speed up the convergency, Luo et al. [80] propose a distributed hash ripple join algorithm. The algorithm assumes that the two join relations are stored in a distributed environment where each node has some tuples. A split vector that maps the join values to the nodes is used to redistribute both relations. The algorithm augments a traditional parallel hybrid hash join algorithm [94]. It contains two phases. In the first phase, the tuples from both relations are retrieved in random order and redistributed to each node so that each node roughly receives the same number of tuples from both relations. When each node is redistributing its stored tuples, it also joins the incoming tuples simultaneously. The second phase starts when the tuples are fully redistributed. The tuples in the buckets of the two relations that are stored on disk are re-read into memory alternatively. They are then joined with existing samples to provide estimates. The authors also discussed how to extend the previous results to provide confidence
bounds for their new distributed hash ripple join. Experimental results showed that the algorithm obtained good speedup.

Jermaine et al. [70, 71] point out that the ripple joins do not work correctly when the sample size exceeds the total amount of memory. They propose the *Sort-Merge-Shrink* (SMS) join to overcome the problem. Essentially this is a combination of the ripple join and the sort-merge join [41]. There are three phases in the algorithm. The sort phase is an augmented sort phase of the sort-merge join. The two join relations are read and sorted into runs in parallel. Furthermore, a hash ripple join is evaluated on each pair of runs. The estimates provided by each hash ripple join are used to provide an estimate for the query answer. The merge phase and shrink phase run simultaneously. In the merge phase, runs are merged and joined together, which means the sorted runs are losing tuples. This requires a shrink phase to update the estimator in order to continuously provide an estimate for the query answer.

Jermaine et al. [69, 73, 89] further extend the SMS join and propose the DBO system. The DBO system is a scalable online aggregation system that utilizes randomized algorithms estimate the answer to a multi-table disk-based query. Like online aggregation, DBO constantly updates estimates from the beginning to the end of the query evaluation. Unlike online aggregation, DBO can handle not only memory-base queries but also disk-based queries. Thus DBO is scalable. In DBO, the database tables are prerandomized and stored on disk. In order to constantly provide and update the estimate of the answer to the query, DBO fundamentally redesigned some part of the query processing engine. Query plans are processed by running all the operations at the same level in the query tree at the same time. This is called a levelwise step. The operations in the same levelwise step communicate with each other to find “lucky tuples” by joining the partial results from each of the operation together. These lucky tuples are then used to estimate the query answer. The authors provide detailed discussions of the DBO-specific operations, and also
gave a thorough analysis for the statistical issues involved in DBO system to estimate the answer to a running query.

There are some other researches in approximately answering a database query, though these are less related to my researches. Ganti et al. [40] propose Icicles, a class of self-tuning sampling techniques that tune themselves to a dynamic workload. The basic idea is that the probability of a tuple being selected as a sample in Icicles is proportional to its importance to answering queries in the workload. If a tuple is used to answer many queries in the workload, the probability that it becomes a sample is much higher. The authors adapt the traditional sampling-based estimators to this biased sampling environment and show that the accuracy of approximate answers obtained by using Icicles is better than a static uniform random sample.

Similarly, Chaudhuri et al. [20] notice that using a uniform sample to estimate an aggregate is not efficient if the selectivity of the query is low or the distribution of the aggregated attribute is skewed. They propose two methods to solve this problem. One of their methods is to use weighted sampling based on the workload information. A record that is evaluated by many queries is sampled with higher probability than a record that is rarely evaluated by queries in the workload. They further introduce a technique called Outlier Indexing to find tuples with outlier values and save them into a separate tables. They demonstrate that the combination of weighted sampling and Outlier Indexing gives better estimates than either uniform sampling or weighted sampling only.

Babcock et al. [11] propose the small-grouping sampling for GROUP BY queries. The basic idea of small-grouping sampling is to partition groups into small groups and large groups. For large groups, the system uses a uniform sample to answer the query. For small groups, the system uses the original data directly to answer the query. Unlike the congressional samples, the small-grouping sampling maintain only a uniform sample. Furthermore, the small-group sampling also maintain a so-called “small group tables” for each attribute to store rows that contain rare values for that particular attribute. The
incoming query is rewritten so that the big groups are answered through the uniform sample and the small groups are answered directly by these “small group tables”.

Chaudhuri et al. [21, 22] further investigate the problem of how to utilize the workload information to precompute a stratified sample [91] in order to minimize the error in answering the workload queries. They demonstrate that when the workload is similar but not identical, the stratified sample taken by their method produces good estimates for workload queries.

2.1.3 Comparison to Survey Sampling

Sampling has been studied for a very long time outside of the database literature. Giving a thorough review of this literature is beyond the scope of this thesis. However, I give a brief comparison between the database sampling and the survey sampling, and pointed out readers to [91] for more information about survey sampling.

Although database sampling techniques are built on top of survey sampling techniques, the purpose of survey sampling is quite different. The purpose of survey sampling is to obtain some specific information of the underlining distribution [18] that is difficult to obtain directly because of large population, privacy issues, legal issues, or other reasons. The target of the survey sampling is known beforehand. The samples collected is only for that particular target. And the survey sampling is specially designed by statisticians and domain experts so that only a very limited number of samples may be needed to estimate the target accurately. This is because generally the cost of collecting samples is very expensive. For example, a study supported by University of Florida would like to study how many people think that they have been sexually harassed on campus. For many reasons, it is infeasible to ask each one on campus to answer this question. Thus, a special survey sampling may be designed. Only a very small sample (tens to hundreds) may be collected to estimate the actual number.

Database sampling is quite different. The main goal in database sampling is to quickly estimate some statistic answer for a given query. Samples may be collected
in advance, stored in the memory or hard disk, and possibly shared by many queries. Furthermore, using a special sample design for a particular query that may increase the estimation accuracy is often infeasible. Because the samples may be shared by many queries, in general, simple sampling, such as uniformly sampling or Bernoulli sampling, is preferred. Furthermore, in database sampling, we generally need to join samples from base tables and estimate query results or statistics from the join result of the samples. For example, in order to estimate the total number of blue zebras, in database sampling, we first estimate the total number of blue animals by taking a sample from a table that contains all animals with their colors. Then we estimate the total number of zebras by taking a sampling from another sample from another table that contains all animals with their types. We then join these two samples together on the unique key of each animal to estimate the total number of blue zebras. This kind of estimation is never required in survey sampling, but is very common in database sampling.

2.2 Top-\(k\) Query Processing in Relational Databases

The problem of finding top-\(k\) groups with respect to a user-specified ranking function using samples is related to work in top-\(k\) query processing. Top-\(k\) query processing has been studied in the databases and web areas, information retrieval, and many other areas. In database literature, there are mainly two different type of top-\(k\) problems:

1. Find the top-\(k\) tuples or objects from the results of a SELECT-PROJECT-JOIN query [41], where each result tuple in the query is ranked by a ranking function [36–38, 67, 84].

2. Find the top-\(k\) groups for an aggregate GROUP BY query [41] such that the groups are ranked by an aggregate ranking function [76, 95].

The first type of top-\(k\) query has a longer research history, and is not very close to the problems I tackle in the thesis. The second type of top-\(k\) query is essentially the same problem that I am working on; the only difference being that other researchers are developing techniques to exactly returning the top-\(k\) groups efficiently when all the data
is available for processing. Research on this type of top-$k$ queries is rare, and the total number of papers [76, 95] published is very small.

In the following, I will first briefly review researches related to the first type of top-$k$ query and then give a thorough survey on researches related to the second type of top-$k$ query developed.

### 2.2.1 Top-$k$ Query Processing for SELECT-PROJECT-JOIN Queries

Research on processing top-$k$ SELECT-PROJECT-JOIN queries has a long history. Fagin et al. [36–38] pioneer the research of processing top-$k$ queries. In their methods, database objects are treated as $n$ lists, one for each attribute. Each attribute is ranked separately with a score, the final score of a database object is a function that combines these scores. The ranking function is assumed to be monotonically increasing. That is, if we fix all the scores of a database object from $n - 1$ list, and only increase the score from one list, the overall rank also increases. This assumption is used in most top-$k$ query processing techniques, and many functions such as $\text{sum}$ have this property. The lists may be sorted based on their scores or random accessed. The Threshold Algorithm (TA) is an algorithm that requires both sorted access and random access. In the TA algorithm, sorted access to each of $n$ sorted list are visited in parallel. When a new object is seen in some list, a random access to each of the other lists is performed to compute the actual score of this new object. An upper bound for the overall score of unseen objects is maintained by applying the score function to the last seen scores in the lists. As soon as at least $k$ seen objects have scores no less than the upper bound, the algorithm terminates and these $k$ objects are returned. The authors also propose the Combined Algorithm (CA) and the No Random Access Algorithm (NRA). CA is an algorithm that is similar to TA but optimized to reduce the number of random access. NRA is used when only sorted accesses are available. Unlike TA or CA, NRA is not guaranteed to return exact answer. It relies on the lower and upper bounds for partially seen objects. The bounds for partially seen objects are computed by applying partially seen scores and the lowest/highest possible
values for unseen attributes. If a partially seen object’s score lower bound is larger than the score upper bounds of all other projects, this object can be reported as a top-$k$ object. Some extensions to the above algorithms [47, 66] are proposed by other researchers. Chaudhuri and Gravano [23] study the problem of mapping a top-$k$ query to a range query [41] in a relational database system. Their work supports three specific score functions: $Min$, $Euclidean$, and $Sum$. All three score functions satisfy the monotonicity property. The basic idea of their mapping algorithm is to choose a score $q$ base on histogram and other statistics in the database, and select all tuples whose scores are larger than $q$. If more than $k$ tuples are selected, the $k$ tuples with highest scores are returned. Otherwise, a score lower than $q$ is selected for another round of query processing. The process is repeated until $k$ records are returned. In the journal version of the paper, the idea is further extended by Bruno el at. [17] to use multi-dimensional histogram [30] to support the mapping.

Later on, Chang and Hwang [19] study the problem of answering a top-$k$ query when some of the predicts of the query are expensive. A predict is a boolean expression that contains selection and join conditions that include user-defined function. The expensive predicts defined in the paper contain user-defined functions, which are ad-hoc, and cannot benefit from sorting and indexing [41]. A cheap predict is an predict that does not contain any user-defined function. They assume that there is at least one cheap predict and the sorted access is available for that predict. They propose the $MinimalProbing$ algorithm (MPro). The main idea is that evaluating some of the predicts (especially expensive ones) are not necessary to determine whether an object is a top-$k$ one or not. Since evaluating some predicts are expensive, the algorithm seeks for minimizing the predicts evaluation cost. The algorithm runs in two phases. In the first phase, a priority queue is initialized based on the score upper bounds of the objects, which are computed by aggregating the scores of the cheap predicates and the maximum scores of the expensive predicates. The second phase, the object at the top of the priority queue is selected and one of the
expensive predicates of that project is probed. The score upper bound is then updated. The object is reinserted into the priority queue. If there is no more predicts for the top objected to be evaluated, it is selected as the output. The problem of determining which expensive predicate is selected as the next probing predicate is an NP-hard problem and MPro only uses heuristics to determine the order. This idea is further extended by Hwang and Chang [65] by taking the cost of sorted access into account.

The first top-$k$ query processing algorithm that supports joining multiple relations proposed in the database literature is $J^*$ by Natsev et al. [84]. $J^*$ is based on the A* search algorithm. It maps the problem to a search problem in the Cartesian space of the ranked inputs and uses guided search to find the results. A priority queue [35] is used to maintain all partially and fully completed results based on the score upper bounds of the final results. At each time, the top element of the priority queue is processed by searching and selecting the next stream to join with the partial result. The partial result is reinserted into the priority queue if it is not completed. Otherwise, this is an output candidate. The authors also proposed several other similar algorithms in the paper.

The Rank-Join algorithm [67] is another top-$k$ query processing algorithm that supports the join operation. It scans the input relations in sorted order based on the scores. When a new record is retrieved, it is immediately joined with existing results of other relations. For each join result, the algorithm computes the score of the join result. The algorithm also maintains a threshold for the join results that are not seen yet. When there are $k$ join results whose actual scores are at least as large as the threshold, the algorithm stops. These $k$ results are returned. The authors develop two hash-based implementations, HRJN and HJRN*. Schnitter and Neoklis [92] further extend the Rank-Join problem, and study the cost of processing a top-$k$ join query. They develop a general model such that the previous algorithms are special cases of their general model. They then study the performance of HRJN and point out that HRJN’s performance can be arbitrary bad. They introduce a new algorithm that works better than HRJN. They
also study the cost of inferring a tight bound of unseen objects and find in general it is NP-hard.

Most top-k query processing techniques assume the ranking function adheres to the monotonicity property. A few recent publications have studied top-k query processing using more general ranking functions. Zhang et al. [106] propose the OPT* algorithm for the problem of answering a top-k query with a ranking function that determines the score of a database object and an arbitrary boolean expression that determines whether a database object is qualified. The solution to the problem is to utilize indices and search the space for indices or scored attributes along with function optimization for continuous ranking functions. The authors propose an A* search algorithm to achieve the completeness and optimality. Xin et al. [104] study the problem of answering a top-k query with an ad-hoc ranking function that can be lower bounded. The authors propose an index-merge framework that searches over the space composed by joining index nodes. These index nodes are existing B+ Tree [41] or R-Tree [41] indices. A double-heap [35] algorithm that supports both progressive state search and progressive state generation is developed.

Researchers have also studied how to integrate top-k query processing techniques into the relational DBMS to provide ranking-aware query plan. Ilyas et al. [68] study the problem of integrating the Rank-Join algorithm [67] into the query optimizer [41]. They modify the enumeration phase of the query optimizer [41] to take the Rank-Join operator as part of the enumeration space. They also provide a cost estimation [41] method for the Rank-Join algorithm. Li et al. [77] extend the work to propose the RANKSQL system that supports ranking as a first-class operation in DBMS. The authors extend the relational algebra to allow ranking operations. They further develop query rewriting rules [45] that can be used for query optimization. Furthermore, the authors introduce an optimization framework and execution model for top-k query processing as a first-class operation in DBMS. More recently, Schnaitter el at. [93] study the problem of how many records are
required to be retrieved from each sorted base relation in order to produce top-\(k\) results and develop novel estimation methods for this problem.

2.2.2 Top-\(k\) Query Processing for GROUP BY Queries

Finding the top-\(k\) groups with aggregate ranking functions is very closely related to the problem of returning top-\(k\) groups with high probability using samples. However, the total number of papers published is rather limited.

Li et al. [76] first study this problem. In the system they propose, they assume that an upper bound of the ranking score of a group can be obtained by replacing the values of each tuple in the group with their maximum values. Many aggregation functions such as \textit{sum} and \textit{average} satisfies this property. A naive implementation to answer such a query may execute the GROUP BY query first, sort the groups on the ranking score and only returns top-\(k\) groups. The authors point out that since users are only interested in the top-\(k\) groups, such naive implementation is inefficient and unnecessary. A better idea is to determine the true top-\(k\) groups while processing the tuples. Unqualified groups are pruned right away once they are found. The authors propose a metric to evaluate how good an algorithm by counting the total number of tuples processed to determine top-\(k\) groups. And they find that the optimal solution relies on two facts: the order of processing the groups and the order of processing tuples within each group. Base on their observations, they propose three principles to process tuples: Upper-Bound, Group-Ranking and Tuple-Ranking. The Upper-Bound principle determines when a group may be pruned. Intuitively if the minimum score of all top-\(k\) groups is \(s\), when the upper-bound of a group’s score is less than \(s\), we are sure that this group is not possible to be a top-\(k\) group. Thus, a loose upper bound function may not help prune any group at all. Furthermore the upper bound function may need to be updated when more tuples are processed in order to provide tighter upper bound. In the implementation, in order to obtain a tight upper bound, the authors assume that the total number of records from each group are known in advance. The Group-Ranking principle determines how to choose
a group to process. The authors point out that a tuple from the current highest upper bound score should be selected for further processing. If $k$ groups finishes processing, these are the top-$k$ groups. The authors further propose that the processing order of tuples within a group only relies on the information within the group. They suggest to process tuples with a group in descending order sorted by the scores of the tuples in the group. Finally, the authors discuss various techniques required to integrate these principles to a relation database engine.

Soliman et al. [95] further extend the problem to the probabilistic database [5, 6, 8–10, 25, 27], and propose algorithms to find top-$k$ groups ranked by a group-wise aggregate function when the underlining tuples are not deterministic. They also assume that the ranking function may be bounded and thus is monotonic. Their approach is mainly based on searching over the query space. The search has two levels: intragroup search and intergroup search. In the intragroup search stage, ordered nonoverlapping aggregate ranges are computed to materialize the necessary parts in the aggregate distribution of each group. These are necessary information to incrementally construct the search space. In the intergroup stage, probability-guided search is performed to navigate the search space and find the query answer.

### 2.3 Related Work in Statistics Literature

In this section, I briefly review some important topics in the statistical literature that are related to the work in this thesis. In statistics literature, two areas are especially related to my research: multiple inference [13, 14, 26, 32, 58–60, 64, 82, 90, 96, 98, 99, 103], and the study of error guarantees for Monte-Carlo methods [12, 28, 81]. The following subsections discuss each of them in detail.

#### 2.3.1 Multiple Inference

Although in database literature, researchers have not considered providing simultaneous confidence bounds for database queries, the study of multiple inference has a long history in statistics. The classical reference material is Miller’s book [82]. Some newer summaries
of the field are the books of Hochberg and Tamhane [59], Westfall and Young [103], and Hsu [64].

Most results regarding multiple inference are related to hypothesis testing [18], which is the process of using observed data to make decisions about unobserved parameters of interest. In a hypothesis test, the tester defines a null hypothesis, a test statistic, and a cutting value. The null hypothesis is an assumption that an effect or result we may be interested in confirming is not true (for example, we may be interested in determining whether an HIV drug is effective; the natural null hypothesis assumes that the drug is not in fact effective). Once the null hypothesis is formulated, a test statistic is calculated using the observed data. The test statistic is used to compute the p-value, which is the probability that a test statistic would be obtained assuming the null hypothesis is true. If the p-value is less than the cutting value $p$, the null hypothesis is rejected. This results in a Type-I or false positive rate of $p$. The so-called Type-II error is the probability of not rejecting a null hypothesis that is in fact false, and describes the power of the test. Further information about hypothesis testing could be found in many statistical books, such as [75].

In multiple hypothesis testing, rather than dealing with a single null hypothesis, one instead has a large number of hypotheses that are to be evaluated using the same data. The problem in this case is that since the same data is used for each test, an error on one test may increase the chance of an error on another. The easiest way to handle such potential correlations is to adjust the p-value associated with each test so that the overall chance of erroneously rejecting a hypothesis is kept manageable. There are a number methods for doing this. The simplest (and least effective) way to adjust the p-value is the Bonferroni correction, or so called classical Bonferroni procedure [82], which simply multiplies each p-value by the number of tested hypotheses. In case the resulting p-value is larger than 1, it is treated as 1. There is also the Dunn-Sidak’s procedure [32]. Intuitively, if we would like to maintain an overall error rate $\alpha$ of rejecting at least one test
out of $n$ tests, Dunn-Sidak’s procedure uses the error rate $\alpha' = 1 - (1 - \alpha)^{1/n}$ to reject each individual test. These procedures are called “single step” procedures. In general, if there are many hypotheses to be tested, these procedures are not very powerful.

There are also “step-down” procedures such as Holm’s procedure [60]. Assume we want to test $n$ hypotheses simultaneously, and the overall type-I error (of making one or more false discoveries) is $\alpha$. In Holm’s procedure, the hypotheses are ordered by their p-values. The smallest p-value is compared to $\alpha/k$. If it is less than $\alpha/k$, the hypothesis is rejected. After it is rejected, the smallest p-value in the remaining hypotheses is compared to $\alpha/(k - 1)$. It is rejected if the p-value is smaller than $\alpha/(k - 1)$. The procedure is continued until the hypothesis with the smallest p-value cannot be rejected. At this point, all remaining hypotheses are accepted. This is called a “step-down” procedure because it starts comparing to the smallest p-value, and gradually increases the p-value to be compared. There are also “step-up” procedures, such as Hochberg’s procedure [58]. Hochberg’s procedure essentially uses the same criteria to reject hypotheses, but the procedure is backwards. It starts with the hypothesis which has the largest p-value and compared this p-value to $\alpha$. It rejects all the hypotheses if the p-value associated with it is smaller than $\alpha$. It then goes to the hypothesis associated with the largest p-value in all the remaining hypotheses, and compare the p-value to $\alpha/2$. If the p-value is less than $\alpha/2$, all the hypotheses with this p-value or smaller p-values are rejected. This procedure continues until no further comparison can be applied. Hochberg’s procedure is a “step-up” procedure because it starts with the biggest p-value in all hypotheses. Hochberg’s procedure is at least as powerful as Holm’s procedure.

One common characteristic of all of the classic statistical methods for simultaneous inference (as well as all of this discussed thus far) is that the goal is to control the so-called “family-wise error rate” (FWER); that is, they seek to control the probability of falsely rejecting any of the hypotheses. They vary in power (Type-II error) and how they
decide to associate a cutting value with each p-value associated with each hypothesis in order to maintain the FWER. In practice, all of these methods tend to be conservative.

There are several problems with controlling FWER. For example, controlling FWER is only meaningful when the overall conclusion from various individual inferences is likely to be erroneous even if only one of these individual inferences is wrong. In many cases, this is not true. For example, if we would like to know whether a new treatment is better than an old one. A treatment group and a control group are generally compared by testing various aspects. The conclusion that the new treatment is better is not necessarily wrong even if some of the null hypotheses are falsely rejected.

In recent years, statisticians have studied the limitation of controlling FWER and proposed other criteria. The first paper to address this problem is published by Benjamini and Hochberg [13]. In stead of controlling FWER, they proposed the False Discovery Rate (FDR), which is the expected fraction of rejected hypotheses that are actually valid. Benjamini and Hochberg show that if all null hypotheses are true, controlling FDR is the same as controlling FWER. But in general FDR is not as large as FWER. Thus controlling FDR increases the power of the test. A “step-down” procedure is proposed by Benjamini and Hochberg. The procedure sorts all the hypotheses based on their associated p-values. Assuming the FDR to be controlled is $q$, and there are $n$ hypotheses, the biggest p-value is compared to $q$, if it is smaller than $q$, all the hypotheses whose associated p-value are smaller than or equal to this p-value are rejected. If the hypothesis is accepted, the next smallest p-value is compared to $(n - 1)q/n$. This procedure continues until some hypotheses are rejected or every hypothesis has been compared. In general, the $i$th largest p-value is compared to $(n - i + 1)q/n$. This procedure is only true for independent test statistics. In reality, this limits the usage of the procedure. The limitation of independent test statistics is addressed by Benjermini and Yekutieli [14]. They prove that the same procedure may be used when the test statistics have positive regression dependency on
each of the test statistics corresponding to the true null hypotheses. The dependency issue is further studied by others. [90, 96].

Story [98] point out several limitations of Benjamini-Hochberg’s procedure and proposed a different approach to control the FDR. Story pointed out that on expectation, Benjamini-Hochberg’s procedure controls the FDR at $\alpha$. However, because the procedure involves estimation, the reliability for a specific case is not guaranteed, which is not good for users. Instead of fixing the FDR to be controlled, Story studies the problem how the expected error rate may be computed when all the associated p-values of the tested hypotheses are given. Unlike traditional multiple hypothesis testing, where we fix the error rate to control and determine the rejection region, this approach fixes the rejection region and then estimates its corresponding FDR. Story shows experimentally that this approach achieves over eight times in power, when comparing to the Benjamini-Hochberg’s procedure.

The two approaches seem very different. Benjamini-Hochberg’s approach fixes the FDR to be controlled first and determines the rejection region, while Story’s approach fixes the rejection region first and estimate the FDR. Story and el at. [99] approve that the two approaches are essentially equivalent. And the estimation of the FDR could be used to define an Benjamini-Hochberg style procedure.

There are many follow-up researches for FDR. Genovese and Wasserman [43] develop a framework that treats FDR control as a stochastic process and propose procedures that controls the tail probabilities of FDR, assuming the test statistics are independent. Van der Laan el at. [101] propose methods to control generalized family-wise error rate (GFWER) instead of controlling FDR. GFWER($k$) is the probability of falsely rejecting at most $k$ hypotheses. The authors propose simple p-value augmented based procedures to control GFWER and claim the procedures are more useful because most procedures of controlling FDR requires independent test statistics or only limited correlated test statistics. Many domain related methods are also proposed, especially in the areas of
Genomics and Biology. The new book by Crowder and Van der Laan [26] gives a very thorough overview on recent researches in multiple inference in general, and specially for applications in Genomics.

2.3.2 Error Guarantees for Monte-Carlo Methods

This section discusses another related topic in statistics, that is the study of error guarantees for Monte-Carlo methods.

In introduction, I briefly discussed how I solve the SI problem. One of the three methods is a Monte-Carlo resampling method. I also mentioned that I have proved an error guarantee between the empirical distribution constructed by samples and the true distribution. More specifically, for any distribution, in order to guarantee that the Euclidian distance between this two distributions is $\epsilon$, the number of samples needed is at least $1/\epsilon^2$.

Similar studies have been proposed in the statistics literature. Most of the researches are about the error guarantees when estimating the cumulative distribution function (cdf) [18] using independent and identically-distributed (i.i.d.) samples [18]. The most fundamental result is the Glivenko-Cantelli’s theorem [12]. We briefly describe it here.

Assume that $X_1, X_2, X_3, ...$ are i.i.d. random variables in $R$ with common cdf $F(x)$. The empirical cdf for $X_1, X_2, X_3, ..., X_n$ is defined by

$$F_n(x) = \frac{1}{n} \sum_{i=1}^{n} I([\!-\infty, x])\{X_i\},$$

where $I$ is a indicator function. For a fixed $x$, $F_n(x)$ is a random variable that converges to $F(x)$ with probability one. This is due to the strong law of large numbers [12]. Glivenko and Cantelli further proved that $F_n$ converges to $F$ uniformly, as shown below:

$$|F_n - F|_\infty = \sup_{x \in R}|F_n(x) - F(x)| \to 0, \text{ with probability 1}$$

Glivenko-Cantelli’s theorem ensures that the empirical cdf converges to the actual cdf uniformly. Thus, when there are sufficient number of i.i.d. samples, the distance between
the empirical cdf and the true cdf can be arbitrarily low. However, Glivenko-Cantelli’s theorem does not provide any bound that relates the total number of samples and the distance between the empirical cdf and the actual cdf. Several other results have been proposed to give an error bound that is related to the total number of samples. One of the most widely used distribution independent bound is the \textit{Dvoretzky-Kiefer-Wolfowitz inequality} \cite{81}, which was first proposed by Dvoretzky, Kiefer, and Wolfowitz, and further refined by Massart, as shown below:

\[
Pr(\sup_{x \in \mathbb{R}} |F_n(x) - F(x)| > \epsilon) \leq 2e^{-2n\epsilon^2}.
\]

The inequality shows that the probability of seeing the distance between the empirical cdf and the actual cdf is more than \(\epsilon\) decreases exponentially as the sample size increases. This inequality is generally good enough for many distributions, especially discrete distributions \cite{18}. For example, if we do not know anything about \(F\) and wish to have at least 90\% confidence such that for any point in the real line, the distance between the empirical cdf and the actual cdf is within 0.01, the total number of required samples can be computed from the following equation: \(2e^{-0.0002n} < 0.1\). By solving the equation, we have \(n \approx 14978\).

Kolmogorov’s theorem \cite{12} is another important conclusion in this field for continuous distributions \cite{18}. We define

\[
D_n = \sup_{x \in \mathbb{R}} |F_n(x) - F(x)|.
\]

If the underlining distribution is a continuous distribution, then \(\sqrt{n}D_n\) converges to the Kolmogorov distribution regardless whatever distribution the underlining distribution is. The cdf of Kolmogorov distribution is shown below:

\[
Pr(K < x) = \frac{\sqrt{(2\pi)}}{x} \sum_{i=1}^{\infty} e^{-(2i-1)^2\pi^2/(8x^2)}.
\]
This can be used to determine how many samples we need for a given distance $\epsilon$ and a given confidence $p$. For example, if we would like to have at least 95% confidence, such that for any point in the real line, the distance between the empirical cdf and the actual cdf is at most 0.01, the total number of samples required is around 18496.

When the distribution family is known in advance, various density estimation techniques [28] may be used. And distribution dependent tighter bounds are available. It is beyond the scope of this section to give a thorough survey for these bounds. We refer readers who are interested in this area to Devroye and Lugosi’s book [28], which discusses these topics in detail.
CHAPTER 3
PRELIMINARIES

In this chapter discusses a few statistical preliminaries. The remainder of the chapter is organized as follows. Section 3.1 discusses estimators and their properties. Section 3.2 discusses correlations. Section 3.3 describes confidence intervals and confidence regions.

3.1 Estimators and Their Properties

Let $M$ be a parameter of interest that we are trying to estimate. In this thesis, the “parameter of interest” is typically the final answer to a query. An estimate $\hat{M}$ of $M$ is a single number computed from a sample or other statistical summary, which serves as a guess of the value of $M$. Note that if the estimation process is repeated many times, many different estimates will be observed. These estimates are typically characterized using a random variable. The random variable whose observed value is used to estimate $M$ is called an estimator and denoted $\hat{M}$.

For example, consider a SUM query over a single database table. Specifically:

```sql
SELECT SUM (f(R))
FROM R
```

The function $f$ can encode any mathematical function over tuples from $R$, and can encode an arbitrary selection predicate. Also, a query of this form can be extended to handle a clause of the form GROUP BY $\text{att}_1$, $\text{att}_2$, ... by first identifying all of the $n$ groups induced by the GROUP BY clause, and then re-running the query $n$ times. During the $i$th run, $f(R)$ is modified to accept only tuples from the $i$th group ($f$ evaluates to zero for any tuple that is not from that group). For example, imagine that we add a clause GROUP BY gender to the above query, and gender has values male and female. We could simply define a function $f_{\text{male}}$ where $f_{\text{male}}(R) = f(R)$ if $R$.gender is male and zero otherwise, as

\[ f_{\text{male}}(R) = \begin{cases} f(R) & \text{if } R.\text{gender} = \text{male} \\ 0 & \text{otherwise} \end{cases} \]

\[ f_{\text{female}}(R) = \begin{cases} f(R) & \text{if } R.\text{gender} = \text{female} \\ 0 & \text{otherwise} \end{cases} \]

1 When $M$ is a parameter to be estimated, we will use $\hat{M}$ to denote the associated estimator; we will also make use of the standard convention of using a capital letter such as $N$ to denote a random variable.
well as a function \( f_{\text{female}} \) where \( f_{\text{female}}(R) = f(R) \) if \( R.\text{gender} \) is \( \text{female} \). Then, running the query twice (once with each function) gives an answer to the GROUP BY query.

Given such a query, the final answer can be computed as:

\[
M = \sum_{r \in R} f(r)
\] (3–1)

The simplest estimator for \( M \) may be random sampling without replacement. To estimate \( M \) using this method, denote by \( R' \) the set of samples from \( R \) and let \( n_R \) denote the sample’s size. If \( N_R \) is the number of tuples in relation \( R \), we can then express \( \widetilde{M} \) by introducing a set of Bernoulli (zero/one) random variables that indicate whether tuples from \( R \) are not/are in \( R' \). Let \( X_k \) be the variable that indicates whether the \( k^{th} \) tuple from \( R \) is in \( R' \). With this, a natural estimator for the aggregate query would be:

\[
\widetilde{M} = \frac{N_R}{n_R} \sum_{k=1}^{N_R} X_k f(k)
\] (3–2)

This estimator simply sums the elements in the sample set and scales the result according to the inverse of the sampling fraction.

The estimator \( \widetilde{M} \) is called an unbiased estimator because \( E[\widetilde{M}] = M \), where \( E[\widetilde{M}] \) is the expected value of the estimator. That is, if the estimator were used many times in succession, the average would be exactly equal to the true answer.

For an unbiased estimator, the variance is usually used as the criteria to indicate how good the estimator \( \widetilde{M} \) is, where \( \sigma^2(\widetilde{M}) = E[\widetilde{M}^2] - E^2[\widetilde{M}] \). The variance of this estimator is:

\[
\sigma^2(\widetilde{M}) = \frac{N_R}{n_R} \frac{n_R - 1}{N_R - 1} \sum_{k=1}^{N_R} \sum_{l=1,l\neq k}^{N_R} f(k) f(l) - \frac{N_R}{n_R} \sum_{k=1}^{N_R} f^2(k)
\] (3–3)

If an unbiased estimator has small variance, then there is only a small chance that its value is far from the true value for the parameter \( M \).

This very simple estimator can be extended to a join over an arbitrarily large number of tables in a straightforward way. We refer to the resulting estimator as the Haas-Hellerstein estimator [49]. This generalization is the estimator used by the UC Berkeley
Control project [56], and if one samples directly from a pre-computed join, the simpler, one-table version of the Haas-Hellerstein estimator is the estimator used by the AQUA project [3] by their join synopsis method [1].

To extend the estimator, we first assume that $T_1, T_2, \ldots, T_k$ are $k$ database tables, and that we wish to guess an answer to a query of the form:

$$\text{SELECT SUM} \left( f(T_1, T_2, \ldots, T_k) \right)$$

$$\text{FROM } T_1, T_2, \ldots, T_k$$

$$\text{WHERE } \text{pred}(T_1, T_2, \ldots, T_k)$$

We let $g(T_1, \ldots, T_k)$ be a function that returns $f(T_1, \ldots, T_k)$ if $\text{pred}$ is true, 0 otherwise. We assume the numbers of tuples in these tables are $N_1, N_2, \ldots, N_k$, and the sample sizes are $n_1, n_2, \ldots, n_k$, respectively. We define Bernoulli random variables $X_{w_1}, \ldots, X_{w_k}$ that govern whether or not the $w$th tuple from $T_1, \ldots, T_k$ are sampled, respectively. The following then serves as the Haas-Hellerstein estimator for the sum of $f$:

$$\tilde{M} = \frac{N_1 \ldots N_k}{n_1 \ldots n_k} \sum_{w_1, \ldots, w_k} X_{w_1} \ldots X_{w_k} g(w_1, \ldots, w_k) \quad (3-4)$$

### 3.2 Correlations

This dissertation considers how correlation-aware methods may be developed to take the correlations among groupwise estimates into account when processing sampling-based GROUP BY queries. In general, the covariance between two variables provides a measure of the correlation between them. The covariance for two random variables $X$ and $Y$ is defined as:

$$\text{Cov}(X, Y) = E[XY] - E[X]E[Y] \quad (3-5)$$

For uncorrelated variables, the covariance between them is 0. If $\text{Cov}(X, Y) > 0$, then $Y$ tends to increase as $X$ increases, and if $\text{Cov}(X, Y) < 0$, then $Y$ tends to decrease as $X$ increases. Figure 3-1 given below show the contours of standard bivariate normal distribution with covariance 0.5 and -0.5, respectively. From the figure, we see for a
negative covariance the distribution’s contours are a set of ellipses, the major axes of which fall into the second and fourth quadrants, respectively. This indicates that $Y$ tends to decrease as $X$ increases. For the positive covariance the contours are a set of ellipses, the major axes of which fall into the first and third quadrants. This indicates that $Y$ tends to increase as $X$ increases.

Figure 3-1. Standard Bivariate Normal(coef=0.5(left) and -0.5(right)).

In general, such multivariate distributions are specified using a covariance matrix $\Sigma$ and a mean vector $\mu$. For the joint distribution of the error of a set of estimators $\tilde{M}_i$, $\mu_i = 0$ and $\Sigma_{i,j} = \text{Cov}(\tilde{M}_i, \tilde{M}_j)$.

### 3.3 Confidence Intervals and Confidence Regions

A confidence interval (CI) for $M$ is an random interval $[l, h]$ with a user-defined probability $p$ associated with it. This is the probability that the interval that is chosen actually contains the parameter of interest. Generally, $p$ is called the confidence and $\alpha = 1 - p$ is called the Type-I error of the estimator $\tilde{M}$.

Properties such as an estimator’s variance and unbiasedness can be used to compute a CI. Define:

$$\text{err} = \tilde{M} - M \quad (3-6)$$
to be the error of some estimator $\tilde{M}$. Since the estimator is unbiased, the mean of $err$ is 0 and the variance is equal to $\sigma^2(\tilde{M})$. According to the Central Limit Theorem, the distribution of $err$ is asymptotically normal for the Haas-Hellerstein estimator considered in this thesis, as long as a large-enough database sample is used (see Haas and Hellerstein, Sections 5.2.1 and 6 for more details [49]). Since (as discussed in the previous subsection) a GROUP BY query can be encoded as a number of Haas-Hellerstein estimators, the estimate for each group in a SUM-based GROUP BY query is also normally distributed.

If the error follows a normal distribution, a CI with confidence $p$ is $[-z_p \sigma(\tilde{M}), z_p \sigma(\tilde{M})]$ where $z_p$ is a coefficient that is determined by the cumulative density function for the normal distribution and $\sigma(\tilde{M})$ is the standard deviation of $\tilde{M}$, which is the square root of the variance $\sigma^2(\tilde{M})$. Thus, the probability that $\overline{M} - M$ is in this interval is $p$. By manipulating this expression, we obtain:

$$Pr[M \in [\overline{M} - z_p \sigma(\tilde{M}), \overline{M} + z_p \sigma(\tilde{M})]] = 1 - p$$ (3-7)

Therefore:

$$[\overline{M} - z_p \sigma(\tilde{M}), \overline{M} + z_p \sigma(\tilde{M})]$$ (3-8)

is a CI for $M$ with confidence level $p$.

The concept of a CI in one dimensional spaces can be extended to multidimensional spaces. Since the area is no longer a interval, it is called a Confidence Region. A confidence region with user defined confidence $p$ is a random region in a multi-dimensional space, such that the probability of seeing the parameter of interest in the region is $p$. The parameter of interest is in this thesis is the query answer for a GROUP BY query. In this thesis, I am mainly interested in providing confidence regions for the join distribution $\tilde{M} = \langle \tilde{M}_1, \tilde{M}_2, ..., \tilde{M}_n \rangle$ of the groupwise estimators, specified using a covariance matrix $\Sigma$ and a mean vector $\mu$, such that $\mu_i = M_i$ and $\Sigma_{i,j} = \text{Cov}(\tilde{M}_i, \tilde{M}_j)$. Chapter 6 will discuss how to build various types of confidence regions in more detail.
CHAPTER 4
COVARIANCE COMPUTATION

The very first step for providing correlation-aware statistical methods for sampling-based
GROUP BY queries is to compute the covariances among the groupwise estimators. This
requires both mathematically deriving the formula of the covariance between a pair of
groupwise estimators, and developing an efficient algorithm to compute it. This chapter
describes various aspects of the covariance computation. The remainder of this chapter
is organized as follows. Section 4.1 formally analyzes the covariance between a pair
of groupwise estimators for a GROUP BY query over one or more database tables.
Section 4.2 shows how to develop a hash-based algorithm to compute the covariance
efficiently. The efficiency of the proposed algorithm is experimentally evaluated in
Section 4.3. Section 4.4 concludes this chapter.

4.1 Covariance Analysis

This section considers the problem of how to formally analyze the covariance for two
groupwise estimators using the “Haas-Hellerstein estimator” [49] for SUM queries over one
or more database tables. Note that this analysis is based on finite population sampling.
This is different from the analysis in [48, 52], which is based on large sample, infinite
population.

4.1.1 Analysis for SUM over Two Database Tables

It is natural to begin by considering a SUM query over two database tables.

Specifically:

```
SELECT SUM (f(R, S))
FROM R, S
GROUP BY group(R, S)
```

Note that the function \( f \) can encode any mathematical function over pairs from
\( R \) and \( S \) (including a COUNT query). The above query may be used to perform the
computation required for a single group in a GROUP BY query if \( f \) limits the sum to only those tuples belonging to the group in question.

Let \( f_i(R, S) \) denote the function for estimator \( \hat{M}_i \) that returns the value of \( f(R, S) \) if the tuple \((R, S)\) belongs to the \( i \)th group, and returns zero otherwise. The number of tuples of relations \( R \) and \( S \) are denoted by \( N_R \) and \( N_S \), respectively. Furthermore, the samples from \( R \) and \( S \) are denoted by \( R' \) and \( S' \), respectively, and \( n_R \) and \( n_S \) are used for their respective sizes. In order to perform the analysis, it is helpful to introduce Bernoulli (zero/one) random variables that indicate whether tuples from relations are not/are in the sample. Let \( X_k \) and \( Y_l \) be variables that indicate whether the \( k \)th and \( l \)th tuples from \( R \) and \( S \) are in \( R' \) and \( S' \), respectively. With this, the \( i \)th natural Haas-Hellerstein estimator of the aggregate is:

\[
\hat{M}_i = \frac{N_R}{n_R} \frac{N_S}{n_S} \sum_{k=1}^{N_R} \sum_{l=1}^{N_S} X_k Y_l f_i(k, l)
\]  

(4–1)

where the double sum is really a sum over the tuples of \( R' \) and \( S' \) expressed using the Bernoulli variables.

Before the estimate \( \hat{M}_i \) can be characterized, some useful properties of the random variables \( X_k \) and \( Y_l \) need to be derived. The necessary results and proofs for \( X_k \) only are presented; the results for \( Y_l \) are obtained simply by substituting \( Y_l \) for \( X_k \) and \( S \) for \( R \) in all equations. First, some notation:

\[
\alpha_R = \frac{N_R}{n_R}, \beta_R = \frac{N_R - 1}{n_R - 1}
\]  

(4–2)

For the purpose of deriving intuitive formulas, the following function is introduced:

\[
\rho_{k,k'}(t) = \begin{cases} 
1 & k = k' \land t = 0 \\
1 & k \neq k' \land t = 1 \\
0 & \text{otherwise}
\end{cases}
\]  

(4–3)
Intuitively this function allows users to turn on/off terms that have equal/not equal values for the indices $k, k'$ based on value of the variable $t$. The following properties of $\rho_{k,k'}$ function will be used extensively:

**Lemma 1.** For any function $\mathcal{F}(k, k')$, The following holds:

$$\forall k, k', \sum_{t=0}^{1} \rho_{k,k'}(t) = 1 \quad (4-4)$$

$$\sum_{k} \sum_{k'} \rho_{k,k'} \mathcal{F}(k, k') = \begin{cases} \sum_{k} \mathcal{F}(k, k) & t = 0 \\ \sum_{k} \sum_{k' \neq k} \mathcal{F}(k, k') & t = 1 \end{cases} \quad (4-5)$$

$$\sum_{t=0}^{1} \sum_{k} \sum_{k'} \rho_{k,k'} \mathcal{F}(k, k') = \sum_{k} \sum_{k'} \mathcal{F}(k, k') \quad (4-6)$$

**Proof.** To prove the first identity, it is observed that irrespective of whether $k = k'$, $\rho_{k,k'}(t)$ takes value 1 for one $t \in \{0, 1\}$ and value 0 for the other, thus the sum is always 1. The second identity follows directly from the definition of $\rho$ by observing that some terms vanish since they have 0 coefficient due to $\rho$. The third identity follows directly from the second, since the two cases in Equation 4–5 are summed up and they complete the double sum.

With these, the following result that characterizes the Bernoulli random variables are available:

**Lemma 2.** If $R'$ is a sample without replacement of $R$, then for the $k$th tuple of $R$ the following holds:

$$E[X_k] = \frac{1}{\alpha_R} \quad (4-7)$$

$$\rho_{k,k'}(t)E[X_k X_{k'}] = \rho_{k,k'}(t) \frac{1}{\alpha_R \beta_R^t} \quad (4-8)$$

**Proof.** The expected value of a zero/one random variable is the probability of the variable taking the value 1; in this case $\frac{1}{\alpha_R}$ is the probability that the $k$th tuple of $R$ is in $R'$.

When $k = k'$, since $X_k^2 = X_k$, $E[X_k X_{k'}]$ is $\frac{1}{\alpha_R}$. When $k \neq k'$, $E[X_k X_{k'}] = P[X_k =
\( \mathbb{I} \land X_{k'} = 1 = P[k \in R'] P[k' \in R' | k \in R'] \). Since the conditional probability is \( \frac{1}{\beta_R} \), then \( E[X_k X_{k'}] = \frac{1}{\alpha_R \beta_R} \). Using the definition of \( \rho_{k, k'}(t) \), the two cases can be easily encoded as in the statement of the lemma. Note that \( \rho \) has to appear on the right side as well as on the left side to ensure the right side is 0 whenever the left side is.

In the formulas derived subsequently, a particular set of terms appear that deserve special attention. They are first introduce and characterize.

For \( t_R, t_S \in \{0, 1\} \) and the \( i \), \( j \)th estimators, the following is defined:

\[
P_{t_R, t_S} = \sum_{k=1}^{N_R} \sum_{l=1}^{N_S} \sum_{k'=1}^{N_R} \sum_{l'=1}^{N_S} \rho_{k, k'}(t_R) \rho_{l, l'}(t_S) f_i(k, l) f_j(k', l')
\]  

(4–9)

Depending on values of \( t_R \) and \( t_S \), these terms can be rewritten using the second identity in Lemma 1 as sums of the form \( f_i(\cdot) f_j(\cdot) \). For example,

\[
P_{0,1} = \sum_{k=1}^{N_R} \sum_{l=1}^{N_S} \sum_{l' \neq l} f_i(k, l) f_j(k, l')
\]  

(4–10)

An extra property of these terms is given by:

**Lemma 3.**

\[
\sum_{t_R=0}^{1} \sum_{t_S=0}^{1} P_{t_R, t_S} = \sum_{k=1}^{N_R} \sum_{l=1}^{N_S} \sum_{k'=1}^{N_R} \sum_{l'=1}^{N_S} f_i(k, l) f_j(k', l')
\]  

(4–11)

**Proof.** The proof follows directly from the second identity in Lemma 1 and simple reorganization of the sums.

**Theorem 1.** For any arbitrary estimators \( \tilde{M}_i \) and \( \tilde{M}_j \) given by Equation 4–1, the following holds:

\[
E[\tilde{M}_i] = \sum_{k=1}^{N_R} \sum_{l=1}^{N_S} f_i(k, l)
\]  

(4–12)

\[
\text{Cov}(\tilde{M}_i, \tilde{M}_j) = \sum_{t_R=0}^{1} \sum_{t_S=0}^{1} \left( \frac{\alpha_R \alpha_S}{\beta_R^2 \beta_S^2} - 1 \right) P_{t_R, t_S}
\]  

(4–13)
Proof. Using linearity of expectation and the first identity in Lemma 2, the unbiasedness of the expectation of $\tilde{M}_i$ follows immediately.

To compute the covariance, the first step is to estimate $E[\tilde{M}_i\tilde{M}_j]$.

$$E[\tilde{M}_i\tilde{M}_j] = \alpha_R^2\alpha_S^2 \sum_{k=1}^{N_R} \sum_{k'=1}^{N_R} \sum_{l=1}^{N_S} \sum_{l'=1}^{N_S} E[X_{k}X_{k'}] E[Y_{l}Y_{l'}] f_i(k,l)f_j(k',l')$$

(4–14)

To rewrite this term, observe that by multiplying each term within the summation by $\sum_{t_R=0}^{1} \rho_{k,k'}(t_R)$ and $\sum_{t_S=0}^{1} \rho_{l,l'}(t_S)$ (both these multipliers are 1 by the first identity in Lemma 1 thus they do not change the identity) and regrouping the sums, the following is obtained:

$$E[\tilde{M}_i\tilde{M}_j] = \alpha_R^2\alpha_S^2 \sum_{t_R=0}^{1} \sum_{t_S=0}^{1} \sum_{k=1}^{N_R} \sum_{k'=1}^{N_R} \sum_{l=1}^{N_S} \sum_{l'=1}^{N_S} \rho_{k,k'}(t_R) E[X_{k}X_{k'}] \rho_{l,l'}(t_S) E[Y_{l}Y_{l'}] f_i(k,l)f_j(k',l')$$

(4–15)

Now, using the second identity in Lemma 2 and the definition of $P_{t_R,t_S}$ the following holds:

$$E[\tilde{M}_i\tilde{M}_j] = \sum_{t_R=0}^{1} \sum_{t_S=0}^{1} \frac{\alpha_R}{\beta_R R} \frac{\alpha_S}{\beta_S S} P_{t_R,t_S}$$

(4–16)

Using now the fact that $\text{Cov} (\tilde{M}_i, \tilde{M}_j) = E[\tilde{M}_i\tilde{M}_j] - E[\tilde{M}_i]E[\tilde{M}_j]$ and the result in Lemma 3 the required result is obtained.

Note that the above result does not require that $i$ and $j$ be different; thus, the result applies to the case when $i = j$, which gives the variance of $\tilde{M}_i$.

4.1.2 Analysis for SUM over Multiple Database Tables

The previous result is now extended to an arbitrary number of database tables. Assume that $T_1, T_2, \ldots, T_k$ are $k$ database tables, and consider the queries of the form:

SELECT SUM ($f(T_1, T_2, \ldots, T_k)$)

FROM $T_1, T_2, \ldots, T_k$

GROUP BY $\text{group}(T_1, T_2, \ldots, T_k)$
Just as before, let $f_i(T_1, ... T_k)$ be a function that returns $f(T_1, ... T_k)$ if the tuple $(T_1, ... , T_k)$ belongs to group $i$, and 0 otherwise. The numbers of tuples in these tables are $N_1, N_2, ..., N_k$, respectively. In addition, it is also assumed the sample sizes are $n_1, n_2, ..., n_k$, respectively. Furthermore, Bernoulli random variables $X_{w1}, ..., X_{wk}$ are defined to govern whether or not the $w$th tuple from $T_1, ..., T_k$ are sampled, respectively. The following then serve as the $i$th estimator for the sums of $f$:

$$\tilde{M}_i = \frac{N_1 ... N_k}{n_1 ... n_k} \sum_{w_1, ..., w_k} X_{w1} ... X_{wk} f_i(w_1, ..., w_k)$$

(4–17)

As in the previous Subsection, in order to derive the covariance of $\tilde{M}_i$ and $\tilde{M}_j$, for $u$ in 1 to $k$, the following are defined:

$$\alpha_u = \frac{N_u}{n_u}, \beta_u = \frac{N_u - 1}{n_u - 1}$$

(4–18)

$$P_{t_1, ..., t_k} = \sum_{w_1=1}^{N_1} \sum_{w'_1=1}^{N_1} ... \sum_{w_k=1}^{N_k} \sum_{w'_k=1}^{N_k} \prod_{u=1}^{k} \rho_{w_u, w'_u}(t_u) \times$$

$$\times f_i(w_1, ..., w_k) f_j(w'_1, ..., w'_k)$$

(4–19)

**Theorem 2. The covariance for multiple database tables.** The covariance of $\tilde{M}_i$ and $\tilde{M}_j$ is:

$$\text{Cov}(\tilde{M}_i, \tilde{M}_j) = \sum_{t_1=0}^{1} ... \sum_{t_k=0}^{1} (\prod_{u=1}^{k} \frac{\alpha_u}{\beta_u}) - 1)P_{t_1, ..., t_k}$$

(4–20)

**Proof.** The proof mirrors the proof of Theorem 1. $k$ terms of the form $\sum_{t_u=0}^{1} \rho_{w_u, w'_u}(t_u)$ are multiplied with terms in the multiple summation of the expansion of $E [\tilde{M}_i \tilde{M}_j]$, then the results in Lemmas 2 and 1, the definition of $P_{t_1, ..., t_k}$, the generalization of Lemma 3 and summation regrouping give the required result. \qed

### 4.1.3 An Unbiased Covariance Estimator

In order to compute the covariances using the formulas described above, it is necessary to have access to the entire database table (which is obviously not feasible if sampling is performed). As a result, it is useful to obtain an unbiased estimator for the
covariance of two estimators, given only the sample that is used to estimate the answer to the query.

Consider the following estimator:

\[
\tilde{\text{Cov}}(\tilde{M}_i, \tilde{M}_j) = \frac{1}{N_1} \sum_{w_1=1}^{N_1} \ldots \frac{1}{N_k} \sum_{w_k=1}^{N_k} \prod_{v=1}^{k} \frac{\alpha_v^{w_v}}{\beta_v^{t_v}} \times \rho_{w_1,w_1'}(t_1) \ldots \rho_{w_k,w_k'}(t_k) \times X_{w_1} X_{w_1'} \ldots X_{w_k} X_{w_k'} f_i(w_1, \ldots, w_k) f_j(w_1', \ldots, w_k')
\]

(4–21)

It is simple to prove that this estimator is unbiased for the covariance of two estimators. To show this, it is noticed that the expected value of:

\[
\sum_{w_1=1}^{N_1} \sum_{w_1'=1}^{N_1} \ldots \sum_{w_k=1}^{N_k} \sum_{w_k'=1}^{N_k} \rho_{w_1,w_1'}(t_1) \ldots \rho_{w_k,w_k'}(t_k) \times X_{w_1} X_{w_1'} \ldots X_{w_k} X_{w_k'} f_i(w_1, \ldots, w_k) f_j(w_1', \ldots, w_k')
\]

(4–22)

is:

\[
\frac{1}{\prod_{v=1}^{k} \alpha_v^{t_v}} P_{t_1,\ldots,t_k}
\]

(4–23)

Therefore, the expected value of \(\tilde{\text{Cov}}(\tilde{M}_i, \tilde{M}_j)\) is the covariance of \(\tilde{M}_i\) and \(\tilde{M}_j\).

4.2 Computational Considerations for Covariances

The previous section gives a detailed derivation of the covariance formulas and associated unbiased estimators for sampling multi-table joins over GROUP BY queries. Unfortunately, if one were to simply implement them directly using the obvious nested loops computations, it would be prohibitively expensive to compute even a single entry in the covariance matrix. This section considers how these formulas can be implemented efficiently.

For ease of exposition, this section will consider how to compute the covariance of the following two queries, which are for two groups:

\[
\text{SELECT sum}(f_1(p_1, l_1))
\]
FROM PART AS \(p_1\), LINEITEM \(l_1\)

\[
\text{SELECT } \sum(f_2(p_2, l_2)) \\
\text{FROM PART AS } p_2, \text{LINEITEM } l_2
\]

In these queries, \(f_1\) computes the total aggregate value for one group, while \(f_2\) computes the total aggregate value for a second group. \(f_1\) and \(f_2\) also compute any selection and/or join conditions. Subsequently, I assume that both functions correspond to queries that can be evaluated using a hash-join or a sort-merge-join.

Upon examining the formulas given above, the key statistics needed to estimate the covariance between the two estimators given above are:

\[
V_{TT} = \sum_{p_1 \in P} \sum_{l_1 \in L} f_1(p_1, l_1) f_2(p_1, l_1)
\]

\[
V_{TF} = \sum_{p_1 \in P} \sum_{l_1 \in L} \sum_{l_2, l_2 \neq l_1} f_1(p_1, l_1) f_2(p_1, l_2)
\]

\[
V_{FT} = \sum_{p_1 \in P} \sum_{p_2 \in P} \sum_{l_1 \in L} f_1(p_1, l_1) f_2(p_2, l_1)
\]

\[
V_{FF} = \sum_{p_1 \in P} \sum_{p_2 \in P} \sum_{l_1 \in L} \sum_{l_2, l_2 \neq l_1} f_1(p_1, l_1) f_2(p_2, l_2)
\]

In the above summations, \(P\) and \(L\) refer to the samples of PART and LINEITEM, respectively. Equivalent formulas for these statistics can be obtained by making use of the identity function \(I\):

\[
V_{TT} = \sum_{p_1 \in P} \sum_{p_2 \in P} \sum_{l_1 \in L} \sum_{l_2 \in L} I(p_1 = p_2) I(l_1 = l_2) f_1(p_1, l_1) f_2(p_2, l_2)
\]

\[
V_{TF} = \sum_{p_1 \in P} \sum_{p_2 \in P} \sum_{l_1 \in L} \sum_{l_2 \in L} I(p_1 = p_2) I(l_1 \neq l_2) f_1(p_1, l_1) f_2(p_2, l_2)
\]

\[
V_{FT} = \sum_{p_1 \in P} \sum_{p_2 \in P} \sum_{l_1 \in L} \sum_{l_2 \in L} I(p_1 \neq p_2) I(l_1 = l_2) f_1(p_1, l_1) f_2(p_2, l_2)
\]

\[
V_{FF} = \sum_{p_1 \in P} \sum_{p_2 \in P} \sum_{l_1 \in L} \sum_{l_2 \in L} I(p_1 \neq p_2) I(l_1 \neq l_2) f_1(p_1, l_1) f_2(p_2, l_2)
\]
Table 4-1. Example results of query 1

<table>
<thead>
<tr>
<th>rid(p)</th>
<th>rid(l)</th>
<th>f(p, l)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 4-2. Example results of query 2

<table>
<thead>
<tr>
<th>rid(p)</th>
<th>rid(l)</th>
<th>f(p, l)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

Note that for these two relations have four statistics, where the subscript $TT$ (for example) requires a double summation over the result sets for $f_1$ and $f_2$, considering identical source record pairs. The subscript $TF$ requires a summation over the two result sets for non-zero $f_1$ and $f_2$ values where the source records from $P$ are the same, but those from $L$ differ. $FT$ is similar. The subscript $FF$ requires a summation over the two result sets where the source records from both $P$ and $L$ differ.

In order to calculate each of these statistics, a reasonable method would be to make use of a double summation over the result set from both of the queries. First, a hash join would be used to compute the set of all non-zero $f_1$ values, since those are the only ones that may contribute to the actual value for the statistic. Then in a similar fashion, a hash join would be used to compute the set of all non-zero $f_2$ values. In order to determine whether each non-zero value can contribute to one of the statistics, for each result tuple the record IDs (RIDs) of the two tuples that were used to produce the value need to be stored. For example, for $V_{FF}$, only those pairs of record pairs where $I(p_1 \neq p_2)$ and $I(l_1 \neq l_2)$ both evaluate to 1 can contribute to the total; this can be determined if the necessary RIDs are available. Once this data has been computed and stored in an array, then a nested summation can be used to compute each statistic. For example, consider Tables 4-1 and 4-2, which give example non-zero $f_1$ and $f_2$ values along with the source
Figure 4-1. The Algorithm for computing $V_{FF}$.

record IDs for the two queries. Given such a data structure, the algorithm shown in Figure 4-1 then computes $V_{FF}$.

If $S_1$ is the set of non-zero $f_1$ values and $S_2$ is the analogous set for $f_2$, this algorithm would require $O(n|S_1||S_2|)$ time. For a larger number of relations, the complexity increases quickly. In general, over $n$ relations, the total number of statistics required is $2^n$, and the time complexity for relations $S_1$ to $S_n$ is $O(2^n \prod |S_i|)$.

By careful implementation, the time complexity in the two-relation case can be reduced to $O(|S_1| + |S_2|)$ (and, by extension, the complexity over $n$ relations can be reduce to $O(2^n \sum |S_i|)$). Continuing with the same example, the basic idea is as follows:

1. First, it is easy to calculate $V_{TT}$. A hash join is simply performed on both the RID’s of PART and LINEITEM.

2. Second, once $V_{TT}$ has been calculated, it is easy to calculate $V_{TF}$. Let

\[ V_{T*} = \sum_{p_1 \in P} \sum_{p_2 \in P} \sum_{l_1 \in L} \sum_{l_2 \in L} I(p_1 = p_2) f_1(p_1, l_1) f_2(p_2, l_2) \]  

(4-24)

Instead of checking the equality of both the pairs of RIDS from both relations, $V_{T*}$ only checks the equality of the two RIDS from PART. This can be computed efficiently using a hash join on the RID from PART. Then, it is noticed that:

\[ V_{T*} = \sum_{p_1 \in P} \sum_{p_2 \in P} \sum_{l_1 \in L} \sum_{l_2 \in L} I(p_1 = p_2) f_1(p_1, l_1) f_2(p_2, l_2) = \sum_{p_1 \in P} \sum_{p_2 \in P} \sum_{l_1 \in L} \sum_{l_2 \in L} I(p_1 = p_2) (I(l_1 = l_2) + I(l_1 \neq l_2)) f_1(p_1, l_1) f_2(p_2, l_2) \]  

(4-25)

\[ = V_{TF} + V_{TT} \]

As a result, $V_{TF}$ is $V_{TF} = V_{T*} - V_{TT}$
3. Third, $V_{FT}$ can be computed in a similar way. First $V_{sT}$ is calculated using a hash join on the RID from `LINEITEM`; then, $V_{FT} = V_{sT} - V_{TT}$.

4. Finally, it is easy to calculate $V_{ss} = V_{TT} + V_{TF} + V_{FT} + V_{FF}$. This is equivalent to calculating:

$$
\sum_{p_1 \in P} \sum_{l_1 \in L} f_1(p_1, l_1) \sum_{p_2 \in P} \sum_{l_2 \in L} f_2(p_2, l_2)
$$

This calculation only requires a single scan of both $S_1$ and $S_2$. Then,

$$
V_{FF} = V_{ss} - V_{TT} - V_{TF} - V_{FT}
$$

The time complexity is $O(|S_1| + |S_1|)$ for each step. Therefore, the overall time complexity is $O(|S_1| + |S_1|)$.

The process is illustrated with an example. $S_1$ and $S_2$ are shown in Figures 4-1 and 4-2 respectively. By executing each step, the following are obtained:

- $V_{TT} = 4$
- $V_{T*} = 39$
- $V_{sT} = 42$
- $V_{ss} = 126$

Therefore:

- $V_{TF} = 39 - 4 = 35$
- $V_{FT} = 42 - 4 = 38$
- $V_{FF} = 126 - 4 - 35 - 38 = 49$

It is not hard to extend this idea to any number of relations. For example, if the two queries both have three relations, the first step is to calculate $V_{TTT}$ using a hash join on the RIDs from all three relations. The next step is to calculate $V_{T**}, V_{T*T}$ and $V_{sTT}$. $V_{TTF}, V_{TFT}$ and $V_{FTT}$ can then be obtained. Similarly, $V_{T**}, V_{ssT}$ and $V_{sTs}$ may be calculated to obtain $V_{TFF}, V_{FFT}$ and $V_{FTF}$. Finally, $V_{sss}$ may be calculated and used to compute $V_{FFF}$. The only difficulty is that the inclusion/exclusion rule needs to be applied.
carefully to ensure that no values are double-counted. For example, if one would like to calculate $V_{T_{FF}}$, he or she should notice that $V_{T_{**}} = V_{TTT} + V_{TTF} + V_{TFT} + V_{TFF}$. In order to compute $V_{T_{FF}}$, he or she needs to subtract the other three totals from $V_{T_{**}}$.

### 4.3 Experimental Evaluation

In this section, I experimentally evaluate the efficiency of the covariance computation algorithm. The efficiency is evaluated in terms of running time. It is important to know how high is the time required to compute the whole covariance matrix, compared to the time required to execute the query over the entire database. For this purpose, I am interested in the following three times:

1. The time to complete the query over the entire database.
2. The time to complete the query over a sample set in order to produce an approximate answer.
3. The time to compute the covariance matrix.

If the algorithm is efficient, I would expect that the summation of the second and third times to be much smaller than the first time. Ideally, I would also expect that the third time is not much larger than the second one. This indicates that the cost of computing the covariance matrix is as fast as estimating the query answer using samples. However, even if the third time is larger than the second one, as long as the summation of the last two times are not as large as the first time, using sampling is still beneficial.

#### 4.3.1 Experimental setup

**a. Query and Schema Tested.** In the proposed experiments, I use the following SQL statement:

```sql
SELECT SUM(o.Profit)
FROM Order o, Employee e, Branch b
WHERE o.EmpID = e.EmpID AND e.OfficeID = b.OfficeID
GROUP BY b.BranchID
```

The three relations used in the query are:
Order(EmpID, Profit, Other)

Employee (EmpID, OfficeID, Other)

Branch(BranchID, OfficeID, Other)

The Other field in each table refers to other possible attributes that could appear. In the proposed experiments, this field is occupied by a random string so that the total size of each record is 100 bytes. I set up the query so that the join between Order and Employee is a primary-key to foreign-key join, and the join between Employee and Branch is a many-to-many join.

b. Parameters Considered. I design this set of experiments to test how different parameters may affect the three times described above. The parameters I considered are the database size, the skewness of the attributes in the join operation, the number of groups in the query, and the sampling ratio.

The database size determines the size of each relation. For simplicity, I treat the size of the largest relation Orders as being approximately the same as the database size. The sizes of relations Employee and Branches are 0.1 and 0.01 times as large as the size of relation Orders. 10 GB, 1 GB and 0.1 GB databases are tested. The default database size is 10 GB. Given the fact that the size of each record is 100 bytes, the default number of records in Orders, Employee, and Branches in a 10 GB database is 100 million, 10 million and 1 million, respectively.

The skewness of the join attributes affects the join in two ways: first, a join operation takes more time if the join attributes are skewed, and second, the correlations between different groups increase as the skewness increases. I use a zipf coefficient to control the skewness of the three attributes Employee.OfficeID, Branch.OfficeID and Order.EmpID. The default zipf coefficient is 0.

The number of groups defines the number of distinct values for the attribute Branch.BranchID, which appears in the GROUP BY clause of the query. This parameter does not affect the first two times much because the grouping operation is near the end.
of query execution. However, increasing the number of groups increases the number of concurrent estimators, and therefore increases the size of the covariance matrix. Thus, the time to compute the covariance matrix is affected. The default number of groups is 100.

The sample ratio determines the number of samples obtained from the database. Given the fact that all other parameters are fixed, increasing the sample ratio increases the sample size. Thus, both the second and third times described above should be affected. The samples are taken so that if a $p\%$ sample from Orders is obtained, a $5p\%$ sample from Employee, and a $20p\%$ sample from Branches will be obtained, respectively. By default, a $1\%$ sample from Orders is obtained. The sampling without replacement policy is used in each experiment.

c. Data Generation. Given a set of parameter values, I generate the data set relation by relation. First, records in Order are generated as follows:

1. A Profit is randomly generated uniformly between 1 and 100.
2. An EmpID is produced by a zipf distribution with a domain size 0.1 times as large as the number of records in Order, and a zipf coefficient specified by the skewness of the join attributes. Because the total number of records in Employee is 0.1 times as large as the total number of records in Order, EmpID is generated from this domain to guarantee the primary key to foreign key join between Order and Employee over EmpID is preserved.
3. A random string is generated to make the size of the record 100 bytes.

Second, records of relation Employee are generated. For each record, the following steps are performed:

1. A unique EmpID is assigned (from 1 to 0.1 times as large as the number of records in Order). EmpID is then the primary key of Employee, and guarantees the domain of EmpID in Employee and Order are the same to preserve the correctness of the primary-key to foreign-key join.
2. An OfficeID is produced by a zipf distribution with a domain size 0.01 times as large as the number of records in Order, and a zipf coefficient specified by the skewness of the join attributes. OfficeID is generated from this domain for both Employee and Branch to preserve the correctness of the many-to-many join between these two relations over OfficeID.
3. A random string is generated to make the size of the record 100 bytes.

Third, records of relation Branch are generated. For each record, the following steps are performed:

1. An OfficeID is produced by a zipf distribution with a domain size 0.01 times as large as the number of records in Order, and a zipf coefficient specified by the skewness of the join attributes. This preserves a many-to-many join between Employee and Branch over OfficeID. Furthermore, this domain is as large as the total number of records in Order.

2. A BranchID is randomly selected (uniformly distributed) from value 1 to the total number of groups.

3. A random string is generated to make the size of the record 100 bytes.

d. Hardware and Software used. All the experiments are run over a DELL desktop with an Intel core-dual 1.8G CPU and 2GB memory using Open SUSE Linux 10.1. Postgres 8.1\(^2\) is used as the backend DBMS to store all the data and perform queries (both queries over the whole database and queries over the samples). The codes are written in C++ and compiled using gcc 4.1.0. Libpqxx 2.6.7\(^3\) is used to connect PostgreSQL in C++. GSL1.9\(^4\) is used for scientific computation and random number generation.

e. Experimental Procedure. I prepare four different sets of databases and samples for the experiments. Each time, I fix three parameters using the default values and vary the fourth. In the first set of experiments, I vary database size to generate 100MB, 1GB, and 10GB databases. In the second set of experiments, I vary different data sets using zipf coefficients 0, 0.3, and 0.6. In the third set of experiments, I vary data sets with 100,

\(^{1}\) http://www.opensuse.org

\(^{2}\) http://www.postgresql.org

\(^{3}\) http://pqxx.org

\(^{4}\) http://www.gnu.org/software/gsl
1000 and 10000 groups. In the last set of experiments, I generate a database and obtain 3 different samples using sample ratios 0.5%, 1%, 2% for \textit{Order}. The two other relations follow the rule described above. For each database and sample generated, I then run the experiments five times to compute the covariance matrix, and report the average time of the five runs.

In each experiment, the time required to compute the covariance matrix is the execution time for the hash-based algorithm after the join over the sample has been computed and is sitting in main memory.

The samples used in each experiment were pre-computed and stored within the database. Since the samples were relatively small with respect to the available main memory, it is reasonable to assume that they were buffered entirely within main memory by the database system and did not need to be read from disk.

\subsection*{4.3.2 Results}

The results are shown in Table 4-3. Running times are given in seconds. Numbers are rounded to the nearest second. When the number of groups is 10000, the computation does not finish in a reasonable time, resulting in an N/A in the figure. This is because operating a 10000 by 10000 covariance matrix is not possible using the given hardware.

\subsection*{4.3.3 Discussion}

I observe that as long as the number of groups is 100, the running time of computing the covariance matrix is very fast, comparing to the running time of evaluating the query over the entire database. When the number of groups is 1000, the covariance computation time is about 18% as long as the running time of executing the query over the entire database, which is still feasible but not as attractive as the 100 group case. When the number of groups is 10000, computing the entire covariance matrix is no longer feasible. Overall, I notice that in many cases, computing the entire covariance matrix using the proposed hash-based algorithm is effective.
### Running time under database size

<table>
<thead>
<tr>
<th></th>
<th>100MDB</th>
<th>1GDB</th>
<th>10GDB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query time</td>
<td>8</td>
<td>85</td>
<td>1053</td>
</tr>
<tr>
<td>Estimate time</td>
<td>1</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>Covariance time</td>
<td>7</td>
<td>8</td>
<td>18</td>
</tr>
</tbody>
</table>

### Running time under different skewness

<table>
<thead>
<tr>
<th></th>
<th>skew0</th>
<th>skew0.3</th>
<th>skew0.6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query time</td>
<td>1053</td>
<td>1604</td>
<td>29164</td>
</tr>
<tr>
<td>Estimate time</td>
<td>12</td>
<td>16</td>
<td>27</td>
</tr>
<tr>
<td>Covariance time</td>
<td>18</td>
<td>22</td>
<td>371</td>
</tr>
</tbody>
</table>

### Running time under different number of groups

<table>
<thead>
<tr>
<th></th>
<th>100 groups</th>
<th>1K groups</th>
<th>10K groups</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query time</td>
<td>1053</td>
<td>1057</td>
<td>1086</td>
</tr>
<tr>
<td>Estimate time</td>
<td>12</td>
<td>15</td>
<td>16</td>
</tr>
<tr>
<td>Covariance time</td>
<td>18</td>
<td>172</td>
<td>N/A</td>
</tr>
</tbody>
</table>

### Running time under different sample ratio

<table>
<thead>
<tr>
<th></th>
<th>0.5%</th>
<th>1%</th>
<th>2%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query time</td>
<td>1053</td>
<td>1053</td>
<td>1053</td>
</tr>
<tr>
<td>Estimate time</td>
<td>8</td>
<td>12</td>
<td>32</td>
</tr>
<tr>
<td>Covariance time</td>
<td>13</td>
<td>18</td>
<td>22</td>
</tr>
</tbody>
</table>

Table 4-3. Running time for Covariance Computation under different parameter values

### 4.4 Conclusion

This chapter has considered the covariance computation in depth. I have formally derived the covariance formula for a pair of groupwise estimators. I have also proposed an efficient algorithm. Finally, I have experimentally evaluated the efficiency of the algorithm. The work in this chapter serves as the base of providing simultaneous confidence bounds and returning all top-\(k\) groups with high probability for GROUP BY queries when only database samples are available.
CHAPTER 5
PROVIDING CORRELATION-AWARE SIMULTANEOUS CONFIDENCE BOUNDS

This chapter discusses how to provide simultaneous confidence intervals to GROUP BY queries. A key issue of this chapter is that how these simultaneous confidence bounds can be provided in a principled fashion efficiently. In order to do this, this chapter defines an abstract problem called Sum-Inference (SI) problem, which may be used to provide simultaneous confidence bounds in a principled fashion. It turned out that solving the SI problem exactly is NP-hard. Thus, the major difficulty in this chapter is to solve the SI problem efficiently. I propose three different approximate methods to solve the SI problem. The remainder of this chapter is organized as follows. Section 5.1 formally defines the SI problem and shows how the SI problem may be used to provide simultaneous confidence bounds. Section 5.3 proposes three approximate algorithms for solving the SI problem. The efficiency and correctness of the proposed algorithms are experimentally evaluated in Section 5.4. Section 5.5 concludes this chapter.

5.1 The SI Problem and GROUP BY Queries

In order to provide simultaneous confidence bounds, the first step is to determine what information or statistics should be communicated to a user in the case of an approximate GROUP BY answer so that he or she is fully aware of the accuracy of an approximation. Thus this information is first discussed in this section. Then the abstract problem that must be solved in order to compute this information, which is call the SI Problem, is formally defined. The section concludes by describing how a solution to the SI problem can be used to compute “safe” GROUP BY bounds.

5.1.1 GROUP BY and Multiple Inference

A GROUP BY query may be viewed as a large set of individual queries asked simultaneously, one for each group. Unfortunately, a traditional confidence bound such as: “With probability $p$, the answer to group $i$ is within the range $l_i$ to $h_i$. “
has only a very narrow correct interpretation in this situation. As soon as a user looks at the results for a second group \( j \), statistically speaking s/he needs to “forget” that s/he ever saw the bounds for group \( i \) and consider group \( j \) in complete isolation! When two or more groups are considered together or compared with one another, the accuracy parameter \( p \) is meaningless, and hence dangerous. Rather than viewing confidence bounds individually, I propose to extend the traditional guarantee and instead present a guarantee taking the form:

“With probability \( p \), at least \( k \) of \( n \) groups are within the ranges \((l_1 \text{ to } h_1)\), \((l_2 \text{ to } h_2)\), ..., and \((l_n \text{ to } h_n)\), respectively.”

In the case where \( n = k \), these new bounds are exactly equivalent to the classical method from statistics for dealing with simultaneous estimates: the bounds are altered so that with high probability, each one of the estimates are within the specified ranges [82]. However, statisticians have begun to acknowledge that this is overly restrictive [13]. For example, the classic method for controlling simultaneous error is to assume that the error is additive (this is known is the Bonferroni correction or the union bound). In order to provide 100 bounds that are all correct with probability 0.99, each one would be correct with probability 0.9999. This can result in needlessly wide confidence bounds. The proposed generalization of the bound to include the parameter \( k \) allows the user to control the extent to which groupwise correlations can affect the number of incorrect estimates.

No longer is a bound derived by computing the probability that it is incorrect; rather, it is derived by computing the probability that a set of bounds are \textit{simultaneously} incorrect.

In order to make this a bit clearer, Figure 5-1 gives a primitive user interface that could be used to present such bounds to the user for a \texttt{GROUP BY} query. The scroll box at the right of the figure gives a confidence bound for each of the 100 groups that are found in the query result. Using the interface, the user chooses \( p \) and \( k \), and the bounds are computed accordingly. In addition, the user is supplied with a plot that shows how likely it is that \( n - k \) or more of the bounds computed are wrong. For example, Figure 5-1 shows
that there is a very small but non-zero chance that at least 30 of the 100 bounds given are incorrect, and there is only about a 0.003 chance that at least 20 of the 100 bounds are incorrect. Since these are relatively small values, the user can be sure that (in this case) the chance of a catastrophic error in the query result is small.

5.1.2 The SI Problem

The issue at the heart of this thesis is how such simultaneous bounds can be computed in an efficient, principled fashion. This subsection describes the *Sum-Inference Problem*, or SI problem for short, which can be used to quantify the extent to which correlation among statistical estimators for database queries can cause many estimates derived from the same synopsis to be incorrect. This problem will form the basis of the computation of safe GROUP BY bounds. The main reason for formulating the abstract SI problem instead of directly introducing and analyzing estimates for GROUP BY queries.
is the fact that there is the potential that the SI problem can be applied to other types of queries. A side benefit is that the analysis GROUP BY queries can be dissociated from the particular method used to estimate the total for each individual group.

The SI Problem models the situation where there are $n$ random variables. The random variable $N_i$ corresponds to a possible answer to the $i$th query or group. It is assume that $N_i$ is normally distributed (which is true asymptotically for a sample-based SUM GROUP BY queries due to the Central Limit Theorem). Also given as input into the SI problem are a set of $n$ valid ranges, with one range for each random variable; each range corresponds to a confidence bound on the answer a each query. The $i$th range is defined by the bounds $l_i$ and $h_i$. If the $i$th range or bound is wrong – that is, if $N_i$ happens to fall outside of the range from $l_i$ to $h_i$ – then a penalty or score $s_i$ is incurred. If the purpose is to simply count the number of incorrect ranges, then $s_i = 1$; in general, $s_i$ may take a different value. The following random variable is then defined:

$$\zeta = \sum_{i=1}^{100} I(N_i \not\in [l_i, h_i])s_i$$

In this expression, $I$ is a function that returns 1 if its boolean argument evaluates to true and 0 if it is false. If a distribution function $F$ is defined such that $F[k]$ gives the probability that the above expression evaluates to $k$, then given $F$ one can check the probability that the total penalty (or total number of incorrect confidence bounds) meets or exceeds $k$, which can be used as a solid indicator as to the accuracy of the given bounds.

Formally, the SI problem is defined as follows:

**The Sum-Inference (SI) Problem.** Given:

- a multi-dimensional normal random variable $N = \{N_1, N_2, \ldots, N_n\}$ with mean vector $\mu$ and covariance matrix $\Sigma$,
- a vector $s$ of $n$ scores,
- a vector $l$ of $n$ lower bounds, and
• a vector $h$ of $n$ upper bounds

the SI problem is the problem of inferring the distribution function $F$, where $F[k] = P(\sum_i I(N_i \not\in [l_i, h_i])s_i = k)$, i.e. $F[k]$ is the probability of observing a total penalty of $k$ due to incorrect intervals.

Unfortunately, it is easy to show that it is likely impossible to infer characteristics of the distribution of $F$ directly:

**Lemma 4. Hardness of the SI problem.** If $F[k]$ can always be computed in polynomial time, then $P = NP$.

**Proof.** Given a set of $n$ integers $S = \{e_1, e_2, \ldots, e_n\}$ and a number $k$, the subset-sum problem asks whether there is a subset of $S$ such that the summation of this subset is $k$. This is known to be NP-complete. This reduce is reduced to the SI problem.

To solve an instance of the subset-sum problem using the SI problem, a random variable $N = \langle N_1, N_2, \ldots, N_n \rangle$ is constructed, where $N_1, N_2, \ldots, N_n$ are independent normally distributed random variables with mean 0 and variance 1. A vector of scores $s = [e_1, e_2, \ldots, e_n]^T$ such that each element in $S$ is in this score vector once and only once is also constructed. Two vectors of lower bounds and upper bounds $l$ and $h$ are also constructed, where the elements are the extent of 95%, two-sided confidence intervals. The resulting SI problem is then solved.

Since each confidence interval correct with probability is 95%, there is a 5% chance that each random variable is outside the interval, thus there is a nonzero probability that any subset of incorrect predictions is obtained. This means that all subsets are possible with greater than zero probability, and so determining whether $F[k] \neq 0$ is actually a question of whether a subset with sum $k$ exists. Thus, the subset sum problem has a solution if and only if $F[k] \neq 0$ for the resulting SI problem.

Subsequent sections of this chapter will consider appropriate methods for computing approximate solutions to the SI problem.
5.2 Applying the Sum Inference Problem to GROUP BY Queries

In this subsection a simple algorithm that makes use of the SI problem to compute the information necessary to display the user interface described previously is proposed. As described, the user supplies parameters $p$ and $k$ such that with probability $p$, at least $k$ of $n$ groups are within the intervals or bounds that are computed; intervals are then chosen automatically to ensure the desired accuracy.

The algorithm makes the assumption that the selected intervals for every group should have the same accuracy. That is, it is assumed that the ratio of the width of the $i$th interval to the variance of the $i$th interval’s standard deviation is always $\rho$. This is reasonable since always using the same ratio gives all groups the same priority; without this restriction the number of acceptable confidence intervals is infinite. A binary search is then performed on the possible $\rho$ values. At each iteration, an SI problem is constructed that can be used to check if a given $\rho$ produces an actual $p$ that is too high or too low for the user’s chosen $k$ value. The algorithm is given below.

<table>
<thead>
<tr>
<th>Function GetIntvls $(k, p, n, \mu, \Sigma)$: $[l, h]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Build a score vector $s$ where $s_i = 1$ for all $i$</td>
</tr>
<tr>
<td>2. Choose an initial upper bound $\rho_u$ for $\rho$</td>
</tr>
<tr>
<td>3. Set $\rho_l = 0$</td>
</tr>
<tr>
<td>4. Set $\rho = \frac{\rho_u + \rho_l}{2}$</td>
</tr>
<tr>
<td>5. Set $l_i = \mu_i - \rho \times \Sigma_{i,i}^{1/2}$</td>
</tr>
<tr>
<td>6. Set $\mu_i = \mu_i + \rho \times \Sigma_{i,i}^{1/2}$</td>
</tr>
<tr>
<td>7. Give $\mu, \Sigma, s, l, h$ as input to an SI problem.</td>
</tr>
<tr>
<td>8. If $</td>
</tr>
<tr>
<td>9. If $p - \sum_{j=0}^{n-k} F[j] &gt; \epsilon$ then $\rho_u = \rho$; goto 4</td>
</tr>
<tr>
<td>10. If $(\sum_{j=0}^{n-k} F[j]) - p &gt; \epsilon$ then $\rho_l = \rho$; goto 4</td>
</tr>
<tr>
<td>11. Return $l$ and $h$ as the final confidence intervals.</td>
</tr>
</tbody>
</table>

There are a few considerations regarding the algorithm that warrant additional discussion. First, the procedure described above requires that a different instance of the SI problem be solved at each iteration of the algorithm. With a binary search algorithm of this type, 30 or so iterations of the algorithm may be required. This may be a concern, since the SI problem can be expensive to solve. Fortunately, it turns out that in practice,
almost all of the computations required to solve the SI problem can be re-used across iterations, rendering the cost of subsequent SI solutions negligible compared to the cost of the first one. This is due to the fact that the underlying random variables $N_1, N_2, \ldots, N_n$ do not change across iterations; only the bounds $l_i$ and $h_i$ change.

Second, the algorithm as presented leaves open the question of how to choose an initial upper bound $\rho_u$. In the actual implementation $\rho_u = 0.1$ is chosen and until the $goto$ in line 9 is used, every time that the $goto$ in line 10 is used $\rho_u$ is doubled rather than setting $\rho_l = \rho$. $\rho_u = 0.1$ is chosen because it is close to the error that a user might find acceptable in the final answer to the query; this tends to reduce the number of iterations of the binary search required. However, the initial choice of $\rho_u$ is not critical, due to the exponentially-fast convergence of the binary search and the fact that additional SI solutions (past the first one) tend to have little cost.

### 5.3 Solving the SI Problem

Given the intractability of the SI problem, developing computationally feasible methods for solving it (or approximately solving it) in a database environment is mandatory if it is to be a practical abstraction. In this section, three different methods for solving the SI problem are proposed:

1. A Monte Carlo solution that samples directly from $F$ in order to learn the distribution;
2. A second method that makes use of moment analysis and an appropriate, parametric model for $F$ in order to approximate it; and,
3. A related method that instead performs approximate moment analysis and is more computationally efficient for very large instances of the SI problem.

Each of the solutions assumes that the mean $\mu$ and covariance matrix $\Sigma$ are provided as input into the SI problem and can be accessed.

#### 5.3.1 A Solution Using Monte Carlo Re-sampling

The first solution approximates $F$ via Monte Carlo sampling. The algorithm repeatedly and directly sample from the multivariate normal or Gaussian distribution
defined by $\mu$ and $\Sigma$. For each sample from the multivariate normal, the total penalty obtained due to incorrect intervals is computed. $F[k]$ is then estimated by computing the fraction of the time that the observed penalty is exactly $k$. For example, if 1000 samples are taken and a penalty of 13 was incurred 55 times, $\hat{F}[13] = 0.055$, where $\hat{F}$ is the estimate for $F$. The observed penalty for each trial or sample is computed by totaling the penalty over each of the $n$ variables $N_1, ..., N_n$. For a given trial, if $N_i$ happens to be outside of the range defined by $l_i$ and $h_i$, then a penalty of $s_i$ is included in the total for that trial.

While this method is fairly simple, there are a few technical questions that need to be considered. First, it is required to be able to sample from the Gaussian distribution defined by $\mu$ and $\Sigma$. It turns out that there are well-known methods for this, that involve first sampling a number of standard normal random variables, using those to form a random vector, and the rotating and translating the vector using $\mu$ and $\Sigma$ and standard methods from linear algebra ([31], Section 4).

A second key question that must be answered is: How many Monte Carlo samples are required in order to obtain a sufficiently accurate approximation for $F$? In order to obtain a guideline for this number, the multivariate Gaussian distribution is sampled $N$ times\(^1\), where $N$ is chosen so that the expected Euclidean distance from $\hat{F}$ to $F$ is less than $\epsilon$ for some user-supplied $\epsilon$. In other words, enough samples are taken so that:

$$E\left[\sum_v (\hat{F}[v] - F[v])^2\right] < \epsilon^2$$

(5.2)

Computing the number of samples required so that this inequality holds requires a bit of mathematics. It is assumed that $\sum_k s_k$ is max (that is, the total possible penalty

---

\(^1\) In the remainder of this section $N$ refers to the number of Monte Carlo trials. I choose not to subscript $N$ in order to clarify this (using $N_{MC}$, for example) in order to simplify the presentation of the formulas. As in the previous section, $N_i$ still refers to the $i^{th}$ component of the multivariate normal that makes up the SI problem.
for any given trial is max) and without losing generality it is assumed that there are no negative penalties (this simplifies the exposition by allowing summations over all possible penalty value to start at 0). Let \( F[v] \) evaluated at \( v = 0, 1, ..., max \) be \( p_0, p_1, ..., p_{max} \). That is, the set of values \( p_0, p_1, ..., p_{max} \) answers the SI problem correctly. Then, random variables \( X_{1.0}, ..., X_{N,max} \) are defined, where \( X_{u,v} \) indicates whether or not the total penalty incurred in the \( u \)th trial was exactly \( v \). That is, \( X_{u,v} \) is one if \( v = \sum_k s_i I(N_i \notin [l_i, h_i]) \) for the \( u \)th of \( N \) multivariate Gaussian samples, and \( X_{u,v} \) is zero otherwise.

Note that \( E[X_{u,v}] = p_v \) for \( u = 1, ..., N \). That is, on expectation, \( X_{uv} \) is exactly the desired value \( p_v \). Thus, the following holds:

\[
E \left[ \sum_{v=0}^{max} (\tilde{F}[v] - F[v])^2 \right] = \sum_{v=0}^{max} E \left[ \left( \frac{\sum_{u=1}^{N} X_{u,v}}{N} - p_v \right)^2 \right] \\
= \sum_{v=0}^{max} E \left[ p_v^2 - 2p_v \frac{\sum_{u=1}^{N} X_{u,v}}{N} + \left( \frac{\sum_{u=1}^{N} X_{u,v}}{N} \right)^2 \right] \\
= \sum_{v=0}^{max} \left( p_v^2 - 2p_v^2 + \frac{1}{N^2} E[\sum_{u=1}^{N} X_{u,v}^2 + \sum_{u=1}^{N} \sum_{w=1, w \neq u}^{N} X_{u,v} X_{w,v}] \right) \\
= \sum_{v=0}^{max} \left( -p_v^2 + \frac{1}{N^2} (Np_v + (N-1)p_v^2) \right) \\
= \sum_{v=0}^{max} \frac{p_v - p_v^2}{N} \\
= \frac{1}{N} \left( 1 - \sum_{v=0}^{max} p_v^2 \right) \leq \epsilon^2 \\
\]

Therefore if:

\[
N \geq \frac{1 - \sum_{v=0}^{max} p_v^2}{\epsilon^2} \tag{5–4}
\]

then the expected Euclidean distance between \( F \) and \( \tilde{F} \) is less than \( \epsilon \). Note that this is guaranteed to be the case if \( N \geq \frac{1}{\epsilon^2} \), because \( 1 - \sum_{v=0}^{max} p_v^2 \) is bounded by 1. As a result, 10000 samples are enough to guarantee an expected Euclidean distance of no greater than 0.01. These methods will be considered experimentally later on in the thesis.
Reusing Monte Carlo SI Computations for GROUP BY queries

When the SI problem is applied to computing bounds for a GROUP BY query, the algorithm to solve the SI problem will be invoked repeatedly to perform a binary search. Fortunately, it is possible to re-use almost all of the computations from the first SI solution when computing subsequent solutions, because only the bounds $l_i$ and $h_i$ change across iterations, and all bound widths are controlled by the parameter $\rho$ (see Section 3.3 for details). This means that subsequent SI solutions are almost “for free”, so that the actual multivariate sampling only needs to be performed the very first time that the SI problem is solved.

To reuse computations, all of the $N$ Monte Carlo trials to build a list of $(\rho^-, trialno, i)$ triples. Each triple means that for the specified trial, an $\rho$ value greater than $\rho^-$ will cause $N_i$ to fall outside of the resulting range $l_i$ to $h_i$. These triples are constructed for every $i$, and then all of them are sorted based upon the $\rho^-$ value during the first SI solution.

Subsequently, for any given $\rho$, it becomes very efficient to check the corresponding $\tilde{F}[k]$. It just requires to scan this sorted array from front to back, and keep going as long as $\rho^-$ does not exceed $\rho$. For every value of $trialno$, the number of triples found are counted. Let $cnt$ be the number of trials values where the number of triples found is $k$. Then returning $cnt/N$ gives us $\tilde{F}[k]$ for the given value of $\rho$. This process can be made even faster by pre-computing summary statistics and storing them in the array, so that the entire lower end of the array need not be re-scanned for each $\rho$ and $k$ pair that is queried.

5.3.2 Solving the SI Problem Using Moment Analysis

Because it is distribution-free, Monte Carlo may be preferred, especially if the computational resources required are minimal. However, for an SI problem with a very large problem size – that is, with a very large number of underlying groups or variables – the Monte Carlo method may not be practical because it requires a large number of samples from the underlying Gaussian. Since the generative model may have arbitrary
pairwise correlations, each Gaussian sample requires multiplying a length $n$ vector by an $n$ by $n$ matrix, which will be expensive if $n$ is very large.

Thus, the second method for solving the SI problem is quite different. It relies on calculating exactly the first and second central moments of the distribution function $F$, and then choosing an appropriate parametric distribution to approximate $F$. Let $\zeta$ be a random variable whose distribution is precisely $F$. Then the first and second central moments of $F$ are the mean $E[\zeta]$ and variance $E^2[\zeta] - E[\zeta^2]$ of $\zeta$.

The justification for resorting to such a parametric method is straightforward. Note that the valid domain of $F$ (or range of $\zeta$) is known and easily computed. That is, in linear time we can easily upper-bound and lower-bound the values of $i$ for which $F[i]$ can possibly have non-zero probability, by computing the smallest and largest sums that over subsets of the $s_i$ values input into the SI problem – the smallest possible sum is $\min\{0, \sum_{s_i < 0} s_i\}$, and the largest possible sum is $\max\{0, \sum_{s_i > 0} s_i\}$, where each $s_i$ is the penalty value associated with the $i^{th}$ group. Given an upper bound, a lower bound, and a mean and variance, the possible shapes that a reasonably well-behaved distribution can take are severely constrained, and so choosing a parametric distribution that accepts these four parameters as a surrogate for $F$ should not incur much inaccuracy.\footnote{As evidence of this, the well-known and widely-used Chebyshev’s inequality states exactly how much just two parameters – mean and variance – constrain a distribution.}

As shown experimentally, the accuracy of this method is quite remarkable when it is used in conjunction with the Beta distribution \cite{18}, which accepts exactly this set of four parameters. The Beta distribution can take a very wide variety of shapes (see Figure 5-2 at left). To give the reader an idea of how good the Beta distribution approximation is for the application, Figure 5-2 (right) shows the empirical distribution of $F[k]$ in a typical scenario obtained using 10000 Monte Carlo samples together with the approximation using beta distribution the proposed method produces. The two distributions are virtually

\cite{18}
indistinguishable. The quality of the Beta distribution for use as a solution to the SI problem will be considered experimentally in depth in Section 5.4.

While such parametric methods may be quite common in statistics and statistical machine learning, there may be skepticism towards such methods in the part of data management practitioners. Guidelines on where to use such parametric distributions can be found in the statistics literature [74]. If there is a concern regarding the accuracy, a principled approach to checking the accuracy of such an approximation would be to compute or estimate the third moment and (perhaps) the fourth moment using techniques similar to those that will be presented shortly. These moments are the so-called skew and kurtosis of $F$, respectively. If both the skew and the kurtosis also match the parametric model chosen, then with six matching parameters the possible distributions are so constrained that no reasonable statistician would ever question the applicability of the
model! However, as shown experimentally in the subsequent section, using the four parameter method, it is possible to obtain excellent results.

Of course, all of this requires that it is possible to perform an appropriate analysis on the variable $\zeta$. Computing $E[\zeta]$ given the input to the SI problem is a relatively easy matter:

$$E[\zeta] = \sum_i \int_{\mathbb{R} - [l_i, h_i]} p_i(x) s_i dx$$

(5–5)

where $p_i(x)$ is the probability density function of the estimate associated with the $i$th estimator, which is normal with mean $\mu[i]$ and variance $\Sigma[i, i]$. This integral for each $i$ simply computes the probability that a trial over $N_i$ results in a value that incurs a penalty, and multiplies that value by the incurred penalty, in order to arrive at the expected penalty for the $i$th estimate.

The next step is to calculate $E[\zeta^2]$. This is the expected squared penalty over all $i$:

$$E[\zeta^2] = \left( \sum_i I(N_i \notin [l_i, h_i]) s_i \right)^2$$

$$= \sum_i I(N_i \notin [l_i, h_i]) s_i \left( \sum_j I(N_j \notin [l_j, h_j]) s_j \right)$$

$$= \sum_i (I(N_i \notin [l_i, h_i]) s_i)^2 + \sum_i \sum_{j \neq i} I(N_i \notin [l_i, h_i]) s_i I(N_j \notin [l_j, h_j]) s_j$$

$$= \sum_i \left( \int_{\mathbb{R} - [l_i, h_i]} p_i(x) s_i dx \right)^2 + \sum_i \sum_{j \neq i} \int_{\mathbb{R} - [l_i, h_i]} \int_{\mathbb{R} - [l_j, h_j]} p_{i,j}(x, y) s_i s_j dy$$

(5–6)

This equation has simply been broken the expected squared penalty into two sums. The first sum considers the case where the penalty for a single $N_i$ is squared, and the second sum considers the case where the penalty for two different $N_i$ variables is multiplied. In the equation, $p_i(x)$ is again the probability density function of $N_i$ and $p_{i,j}(x, y)$ is the probability density function of the bivariate normal with:

$$\mu' = [\mu[i], \mu[j]]^T; \Sigma' = \begin{bmatrix} \Sigma[i, i] & \Sigma[i, j] \\ \Sigma[i, j] & \Sigma[j, j] \end{bmatrix}$$

(5–7)
In other words, \( p_{i,j}(x, y) \) is the joint density function for \( N_i \) and \( N_j \). Evaluating this equation requires that it is possible to compute the required integral over \( p_{i,j}(x, y) \).

Fortunately there is a good deal of existing work that considers how to compute an integral over a bivariate normal. If Monte Carlo integration is used [88], then it is easily possible to reuse the SI solution for the first iteration of the GROUP BY computation of Section 3.3 for subsequent iterations (see below).

**Applying the Beta Distribution**

After calculating the first two moments, an appropriate parametric distribution can then be used to (approximately) calculate any \( F[k] \). As mentioned previously, in the actual implementation the four-parameter Beta distribution is used to serve as a surrogate for \( F \). The four parameters for the Beta are the two shape parameters \( \alpha \) and \( \beta \), as well as the lower and upper bound \( \text{min} \) and \( \text{max} \) for the range of the underlying random variable \( \zeta \).

If \( \text{min} = 0 \) and \( \text{max} = 1 \), straightforward calculations show that given \( \mu(\zeta) \) and \( \sigma^2(\zeta) \), \( \alpha \) and \( \beta \) should be chosen so that:

\[
\alpha = \frac{\mu^2(\zeta) - \mu^3(\zeta) - \mu(\zeta)\sigma^2(\zeta)}{\sigma^2(\zeta)}
\]

\[
\beta = \frac{\mu(\zeta) - 2\mu^2(\zeta) + \mu^3(\zeta) - \sigma^2(\zeta) + \mu(\zeta)\sigma^2(\zeta)}{\sigma^2(\zeta)}
\]

Generally \( \text{min} = 0 \) and \( \text{max} = 1 \) are not satisfied. However a simple linear transformation can map this to a space such that \( \text{min} = 0 \) and \( \text{max} = 1 \). The mean and variance are transferred accordingly. Assuming the mean and variance in the original space are \( \mu' \) and \( \sigma'^2 \), respectively, then the following equations hold:

\[
\mu = \frac{\mu' - \text{min}}{\text{max} - \text{min}}
\]

\[
\sigma^2 = \frac{\sigma'^2}{(\text{max} - \text{min})^2}
\]
This ensures that the first two moments of the resulting distribution $\tilde{F}$ both match
$\mu(\zeta)$ and $\sigma^2(\zeta)$, respectively. Given this fit, to provide the solution to the SI problem
$\tilde{F}[k] = \int_{k-0.5}^{k+0.5} \text{Beta}(x)dx$ is used.

**Reusing Moment-Based SI Computations for GROUP BY queries**

In the same way that the Monte Carlo solution for the first iteration of the GROUP
BY algorithm can be reused, the moment-based solution can be re-used to solve multiple
SI problems with the same underlying variables. Assuming that Monte Carlo integration
is used for each integral over $p_{i,j}(x,y)$ and $p_i(x)$, a number of samples are taken from
each $p_{i,j}(x,y)$ and $p_i(x)$, and the fraction of samples within each corresponding bound
is computed to approximate the integral. Using methods very similar to those in the
previous section for reusing the pure Monte Carlo solution, these samples can be used
to compute an efficient mapping from every possible $\rho$ to a given variance and mean for
$\zeta$. For example, consider the mean of $\zeta$. To handle this, a number of records of the form
$(\rho^-, i, \mu)$ are computed for each $i$, one corresponding to each sample from $p_i$. This record
indicates that $N_i$’s contribution to the mean of $\zeta$ for $\rho > \rho^-$ is at least $\mu$. The records
are then sorted on the $\rho^-$ values. To compute the mean of $\zeta$ for a given $\rho$, the records
are scanned from front to back, for each $\rho^-$ that does not exceed $\rho$. For each $i$, the last $\mu
observed is then used to compute $\zeta$’s mean.

**5.3.3 Solution Using Approximate Moment Analysis**

While the previous solution may be faster than the Monte Carlo solution, it is still
linear in the size of the covariance matrix $\Sigma$. In the case of a GROUP BY query, the
number of entries in $\Sigma$ is quadratic in the number of groups. Since the number of groups
may be many thousands in a realistic scenario, a sub-linear algorithm in the size of $\Sigma$ is
highly desirable.

For a very large SI problem, it is possible to increase the speed of the moment
analysis method substantially by sampling from the covariance matrix. Recall that the
first two moments of the distribution resulting from the SI problem are given as equations
(13) and (14). In order to make use of the method of moments, the second uncorrected moment is generally not as important as the variance, which is related to the second moment via the relationship \( \sigma^2(\zeta) = E[\zeta^2] - E^2[\zeta] \). To write an expression for \( \sigma^2(\zeta) \), let us define:

\[
C = \sum_i \left( \int_{\mathbb{R} - [l_i, h_i]} p_i(x) s_i dx \right)^2 - \left( \sum_i \int_{\mathbb{R} - [l_i, h_i]} p_i(x) s_i dx \right)^2 \sum_i \sum_{ji} \delta_{ij}
\]

Then, the following equation is obtained:

\[
\sigma^2(\zeta) = E[\zeta^2] - E^2[\zeta] = \sum_i \sum_{ji} \left( \int_{\mathbb{R} - [l_i, h_i]} \int_{\mathbb{R} - [l_j, h_j]} p_{i,j}(x, y) s_i s_j dxdy + C \right)
\]

The main reason to define \( C \) is that in order to calculate \( C \), only the diagonal entries of the covariance matrix are needed to be calculated, a cost that scales linearly with respect to the number of estimators. Thus computing \( C \) exactly is feasible, and the problem of calculating the variance reduces to the problem of computing the value of the double summation given above, which can be estimated effectively using sampling.

Assuming the total number of estimators is \( n \), the total number of non-diagonal entries in the double summation is \( m = n(n - 1) \). Let \( S \) be a random sample without replacement from:

\[
T = \{(i, j)|1 \leq i \leq m, 1 \leq j \leq m, i \neq j\}
\]

where \(|S|\) is the sample size and \(|T| = m\). The following formula then gives an unbiased estimator of the variance of \( \zeta \):

\[
\tilde{\sigma}^2(\zeta) = \frac{m}{|S|} \sum_{(i,j) \in S} \left( \int_{\mathbb{R} - [l_i, h_i]} \int_{\mathbb{R} - [l_j, h_j]} p_{i,j}(x, y) s_i s_j dxdy + C \right)
\]

In order to compute how many samples from the covariance matrix are needed in order to accurate estimate \( \sigma^2(\zeta) \), the formula for the variance of \( \tilde{\sigma}^2(\zeta) \) can also be derived. Define:
\[
\begin{align*}
\frac{a}{m} &= \frac{m}{|S|} \\
\frac{b}{m-1} &= \frac{m-1}{|S|} - 1 \\
\end{align*} \]

\[h(i, j) = \left( \int_{\mathbb{R} - [l_i, h_i]} \int_{\mathbb{R} - [l_j, h_j]} p_{i,j}(x, y) s_i s_j dx dy - C \right)
\]

(5–12)

If sampling without replacement is used, the variance of \(\tilde{\sigma}^2(\zeta)\) is:

\[
\sigma^2(\tilde{\sigma}^2(\zeta)) = \left( \frac{a}{b} - 1 \right) \sum_{(i,j) \in T} \sum_{(k,l) \in T} I((i, j) \neq (k, l)) h(i, j) h(k, l)
\]

\[+ (a - 1) \sum_{(i,j) \in T} h(i, j)^2 \]

(5–13)

An unbiased estimator of the variance that only based on the samples is:

\[
\tilde{\sigma}^2(\tilde{\sigma}^2(\zeta)) = ab \left( \frac{a}{b} - 1 \right) \sum_{(i,j) \in S} \sum_{(k,l) \in S} I((i, j) \neq (k, l)) h(i, j) h(k, l)
\]

\[+ a(b - 1) \sum_{(i,j) \in S} h(i, j)^2 \]

(5–14)

Given these formulas, it is easy to design an algorithm that repeatedly samples from the non-diagonal entries in the covariance matrix, and calculates the estimated value of \(\tilde{\sigma}^2(\zeta)\) based on the current sample set \(S\). The algorithm then stops sampling when the standard deviation of this estimator is less than some percentage of the estimated value, computed using an appropriate bound (such as the Central Limit Theorem or Chebyshev’s inequality).

The resulting algorithm is shown in Figure 5-3. Function GetEstimatedVariance implements the algorithm. The first two lines initialize some variables. Lines 3 to 8 calculate the constant \(C\) given in Equation 5–8. The Do...While loop in lines 9 to 18 continues sampling and updating the estimated variance using Equation 5–11 until the...
stopping condition described above is satisfied. The function $CalculateVariance$ calculates the estimated variance using Equation 5–14.

Function $GetEstimatedVariance \ (T, n)$
1. $Exp = 0, Vpart = 0, C = 0, m = n(n - 1)$
2. $SampleSize = 0, Estimate = 0, Total = 0$
3. For $i = 1$ to $M$:
   4. $A = \left( \int_{R - [l_i, h_i]} p_i(x)s_i \, dx \right)$
   5. $Exp = Exp + A$
   6. $Vpart = Vpart + A^2$
   7. $C = \frac{Vpart - Exp^2}{m}$
4. Do:
   5. Sample entry $(i, j)$ from $T$
   6. $h(i, j) = \left( \int_{R - [l_i, h_i]} \int_{R - [l_j, h_j]} p_{i,j}(x, y) s_i s_j \, dx \, dy + C \right)$
   7. $SampleSet.add(h(i, j))$
   8. $Total = Total + h(i, j)$
   9. $SampleSize ++$
   10. $Estimate = \frac{m}{SampleSize} Total$
   11. $\sigma^2 = CalculateVariance(SampleSet, N)$
   12. $\sigma = \sqrt{\sigma^2}$
4. While$(Std \times 10 > Estimate)$
5. Return $Estimate$

Function $CalculateVariance \ (S, m)$
1. $NonEqPart = 0, EqPart = 0$
2. $a = \frac{m}{|S|}, b = \frac{m - 1}{|S| - 1}$
3. For any two different elements $s, t \in S$
   4. $NonEqPart = NonEqPart + s \times t$
4. For any element $s \in S$
   5. $EqPart = EqPart + s^2$
6. Return $ab(\frac{a}{b} - 1)NonEqPart + a(b - 1)EqPart$

Figure 5-3. Algorithm to calculate the estimated variance

5.4 Experimental Evaluation

In this section, the utility of the SI-based simultaneous GROUP BY bounds is experimentally evaluated. There are two questions to be considered. First, the purpose of sampling from a database is to obtain an approximate answer quickly. However, the additional computation required to produce the simultaneous GROUP BY bounds will
require extra time, which may perhaps mitigate the benefit of sampling. Thus, the first question to address is: For each of the three methods that have been proposed, how high is the time required to obtain the SI-based simultaneous GROUP BY bounds, compared to the time required to execute the query over the entire database?

Second, all three methods that have been proposed are approximate methods, and do not solve the SI problem exactly. Therefore, the second question to address is: How well do these three methods bound the correctness of an approximate GROUP BY answer in reality?

5.4.1 Running Time Experiments

To evaluate the utility of the SI-based simultaneous GROUP BY bounds, three different times are computed:

1. The time to complete the query over the entire database.
2. The time to complete the query over a sample set in order to produce an approximate answer.
3. The time to compute simultaneous confidence bounds using the SI problem.

If sampling is useful for reducing the running time of a query, it is expected that the summation of the last two times to be much smaller than the first time. Otherwise, the query should simply be answered exactly. In the ideal case, it is also expected the third time to be much smaller than the second one, so that the simultaneous confidence bounds are produced at no extra cost compared to the time required to estimate the answer. However, even if the third time is larger than the second one, this only indicates that there is room for improvement and does not preclude the use of the proposed methods, because the total sampling time may still be less than evaluating the query exactly.

5.4.1.1 Experimental setup

In the experiments, the same SQL statement as described in Section 4.3 is used, as shown below:

```
SELECT SUM(o.Profit)
```
FROM Order o, Employee e, Branch b
WHERE o.EmpID = e.EmpID AND e.OfficeID = b.OfficeID
GROUP BY b.BranchID

The schemas of the three relations used in the query are:

Order(EmpID, Profit, Other)
Employee(EmpID, OfficeID, Other)
Branch(BranchID, OfficeID, Other)

These relations and the samples used for estimation are generated through a generation procedure controlled by four parameters: database size, skewness, number of groups, and sample ratio. The generation procedure and the experimental platform (both software and hardware) are discussed in 4.3.

The same datasets as described in Section 4.3 are prepared for the experiments. There are overall 12 different datasets. Each time, three parameters are fixed using the default values and the fourth is varied. The default value for database size, skewness, number of groups and sample ratio are 10GB, 0, 100, and 0.5%, respectively. In the first set of experiments, database size is varied to generate 100MB, 1GB, and 10GB databases. In the second set of experiments, different data sets are generated using zipf coefficients 0, 0.3, and 0.6. In the third set of experiments, data sets are generated with 100, 1000 and 10000 groups. In the last set of experiments, a database is generated and obtain 3 different samples using sample ratios 0.5%, 1%, 2% for Order. The two other relations follow the rule described above. For each database and sample generated, the experiments are run five times using each of the three proposed solutions to the SI problem, and the average times of the five runs are reported.

In each experiment, the time required to solve the SI problem includes exactly the time required to compute $F$ once the join over the sample has been computed and is sitting in main memory.
The samples used in each experiment were pre-computed and stored within the database. Since the samples were relatively small with respect to the available main memory, it is reasonable to assume that they were buffered entirely within main memory by the database system and did not need to be read from disk.

5.4.1.2 Results

The results are shown in Table 5-1. Running times are given in seconds. Numbers are rounded to the nearest second. When the number of groups is 10000, both the moment analysis method and the Monte Carlo resampling method do not finish in a reasonable time, resulting in an N/A in the figure. This is because these two methods require operations over a 10000 by 10000 covariance matrix, which is not possible using the given hardware.

In order to put the sample sizes that has been tested into some sort of context, the 95% confidence interval width for sample ratio 0.5%, 1%, and 2% averaged 43%, 20%, and 8% of the estimated answer, respectively, if all other parameters take their default value. Thus, a 2% sample produces errors that are less than 10% (which should be quite reasonable for many data exploration applications) and the error bounds seem to shrink by 1/2 or more with a doubling of the sample size.

5.4.1.3 Discussion

Several interesting results can be observed, a few significant ones are pointed here.

First, it is notice that the time to actually compute the sample-based estimate increases approximately linearly as the sample ratio increases (which is expected), while the time required to solve the SI problem is generally independent of the sample size.

Second, the running time for Postgres to complete the query over the entire database is from around 16 minutes to more than 8 hours. On the other hand, Postgres never need longer than a minute to compute an approximate query answer over the samples in the worst case. Thus, sampling itself seems to be a valuable method in terms of reducing computation time, even over multi-table joins.
Running time under database size

<table>
<thead>
<tr>
<th></th>
<th>100MDB</th>
<th>1GDB</th>
<th>10GDB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query time</td>
<td>8</td>
<td>85</td>
<td>1053</td>
</tr>
<tr>
<td>Estimate time</td>
<td>1</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>Moment analysis</td>
<td>11</td>
<td>12</td>
<td>23</td>
</tr>
<tr>
<td>Approximate moment analysis</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Monte Carlo resampling</td>
<td>66</td>
<td>111</td>
<td>221</td>
</tr>
</tbody>
</table>

Running time under different skewness

<table>
<thead>
<tr>
<th></th>
<th>skew0</th>
<th>skew0.3</th>
<th>skew0.6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query time</td>
<td>1053</td>
<td>1604</td>
<td>29164</td>
</tr>
<tr>
<td>Estimate time</td>
<td>12</td>
<td>16</td>
<td>27</td>
</tr>
<tr>
<td>Moment analysis</td>
<td>23</td>
<td>26</td>
<td>406</td>
</tr>
<tr>
<td>Approximate moment analysis</td>
<td>1</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>Monte Carlo resampling</td>
<td>221</td>
<td>245</td>
<td>435</td>
</tr>
</tbody>
</table>

Running time under different number of groups

<table>
<thead>
<tr>
<th></th>
<th>100 groups</th>
<th>1K groups</th>
<th>10K groups</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query time</td>
<td>1053</td>
<td>1057</td>
<td>1086</td>
</tr>
<tr>
<td>Estimate time</td>
<td>12</td>
<td>15</td>
<td>16</td>
</tr>
<tr>
<td>Moment analysis</td>
<td>23</td>
<td>215</td>
<td>N/A</td>
</tr>
<tr>
<td>Approximate moment analysis</td>
<td>1</td>
<td>1</td>
<td>57</td>
</tr>
<tr>
<td>Monte Carlo resampling</td>
<td>221</td>
<td>18055</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Running time under different sample ratio

<table>
<thead>
<tr>
<th></th>
<th>0.5%</th>
<th>1%</th>
<th>2%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query time</td>
<td>1053</td>
<td>1053</td>
<td>1053</td>
</tr>
<tr>
<td>Estimate time</td>
<td>8</td>
<td>12</td>
<td>32</td>
</tr>
<tr>
<td>Moment analysis</td>
<td>15</td>
<td>23</td>
<td>28</td>
</tr>
<tr>
<td>Approximate moment analysis</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>Monte Carlo resampling</td>
<td>219</td>
<td>221</td>
<td>230</td>
</tr>
</tbody>
</table>

Table 5-1. Running time under different parameter values

Third, as long as the number of groups is 100, even for the slowest method (Monte Carlo resampling), the running time to produce simultaneous GROUP BY confidence intervals in the worst case is only 20% as long as the running time to execute the query over the entire database, when the data has no skew. If the data is very skewed, this extra time is only 2% as long as the query time. This indicates that all three methods are valuable for 100 groups or less. However, when the number of groups is 1000, Monte Carlo resampling takes around 20 times as long as the running time to execute the query over the entire database, and thus is not practical to use. Both moment analysis and
approximate moment analysis still require less time than the time to execute the query
over the entire database. When the number of groups is 10000, the only method practical
is approximate moment analysis.

It is noticed that the third time for both the moment analysis and the Monte Carlo
resampling method is generally larger than the time to complete the query over the
samples. This indicates that these two methods are not ideal, though they are still useful.
However, the approximate moment analysis method was nearly ideal in terms of its
computational time. Most of the time, the time to obtain simultaneous GROUP BY
bounds was much smaller than the time to compute the query result over a sample. The
only time where this was not the case was when there were 10000 groups (the computation
required 57 seconds). However, even in this case, the method takes just 4 times as long
as it does to compute the query result over the sample, and it takes just 1/20 as long
to obtain the simultaneous bounds as it does to compute the exact result. Therefore
approximate moment analysis can be used virtually for free in most cases in terms of the
extra computation that is required.

Finally, I acknowledge that the experiments consider what is really the best possible
scenario for the application of sampling: an arbitrary, ad hoc query is issued for which
no pre-computed computation is available. This favorable scenario is the main reason for
the orders-of-magnitude speedup associated with sampling. In other scenarios, such as
an incremental scenario where a user drills down into an already-computed answer set,
sampling may look much less attractive. Still, the results do convincingly show that when
sampling is applicable, computing SI-based simultaneous bounds should not be much more
expensive than simply using the underlying sample to compute an estimate.
5.4.2 Correctness Experiments

5.4.2.1 Experimental Setup

In this section, the correctness of the three proposed methods for solving the SI problem are experimentally evaluated, including the applicability of the parametric solutions.

Testing the correctness of any confidence bound is non-trivial, and is generally done using Monte Carlo methods. For example, a traditional $p\%$ confidence bound tells us that a parameter of interest is within a specified range $p\%$ of the time. A Monte Carlo experiment to check the correctness of such a confidence bound would re-run the computation that produced the bound $N$ independent times, and see if approximately $\frac{p}{100} \times N$ out of the $N$ times the bounds contain the true answer. Under this regime, each repetition can be treated as a coin flip where a heads is observed when the bound contains the true answer, and a tails otherwise. If the confidence bound computation is correct, the total number of observed heads would follow a binomial distribution, given by $Binomial(N, \frac{p}{100})$. As long as the actual number of observed heads is within a two-sided binomial confidence interval computed using $Binomial(N, \frac{p}{100})$, it is confident to conclude that this confidence bound is correct.

It is more challenging to design a test for the three solutions to the SI problem. In the case of a GROUP BY query, for a given set of confidence bounds, the solution to the SI problem is a function $F$ such that for any given $k$, $F[k]$ is the probability of observing $k$ incorrect intervals. For example, if $k = 12$ and $F[k] = 0.25$, it is expected 25% of the time to observe 12 incorrect bounds. Thus, if $p \approx \sum_{i=0}^{k} F[i]$, then the SI solution asserts that the probability of observing $k$ or less incorrect bounds is $p$.

For a given $p$, a Monte Carlo experiment can be designed to check the correctness of an SI solution by repeating the following procedure $N$ times independently. First, a sample is obtained from the database tables and the sample is used to estimate the result to a GROUP BY query. For each group, a 95% is constructed confidence interval using the
sample, and the resulting SI problem is solved. Using $F$, then next step is to find $k$ such that the probability of observing $k$ or less incorrect bounds is $p$. Then the actual number of incorrect intervals is calculated by computing the actual the answer of the query over the entire database, and then counting the total number of intervals that do not contain the answers for the associated group. The “coin flip” associated with this SI solution is a heads if the actual number of intervals is less than or equal to $k$, and a tails otherwise. Therefore, if the method to solve the SI problem is correct, the total number of heads in $N$ independent repetitions follows a binomial distribution, given by $Binomial(N, p)$. As long as the actual number of observed heads is within a two-sided binomial confidence interval, it is confident to conclude that the method for solving the SI problem is correct.

To actually implement such an experiment, the same query described in the previous section is used:

```
SELECT SUM(o.Profit)
FROM Order o, Employee e, Branch b
WHERE o.EmpID = e.EmpID AND e.OfficeID = b.OfficeID
GROUP BY b.BranchID
```

Four different data sets are built for this experiment. In the first data set, all parameters take their default values. In the second data set, the sample ratio is changed to 0.5%. In the third data set, the skewness is changed to 0.3. In the fourth data set, the skewness is changed to 0.3, and the number of groups is changed to 20. For each of the three methods (Monte Carlo resampling, moment analysis, and approximate moment analysis), and for each of four data sets, the procedure described above is performed for $p$ in \{0.05, 0.10, ..., 0.90, 0.95\}, and $N = 100$.

### 5.4.2.2 Results

The observed results are given in Figures 5-4, 5-5, 5-6, 5-7. The Figures show the 12 results observed using the three different SI solutions and four different data sets. The $x$ axis in each figure is the $p$ tested. The $y$ axis (shown as a bar) is the total
Figure 5-4. Correctness of three methods over dataset one: All parameters use their default values. The error bar is a 95% two-sided confidence interval from the corresponding binomial distribution $Binomial(100, p)$, where each $p$ value is shown in the x axis.

The number of observed heads out of 100 trials. The error bar over each bar is the two-sided 95% confidence interval of $Binomial(100, p)$. In order to compute the interval, the inverse cumulative distribution function values of probabilities 0.025 and 0.975 from $Binomial(100, p)$ are computed and treated as lower bound and upper bound, respectively.

5.4.2.3 Discussion

If the SI solutions are unreliable in practice, it is expected to observe a number of heads is outside of the two-sided 95% confidence interval of $Binomial(100, p)$ more than 5% of the time. In actuality, the observed result shows that only 7 out of 240 results are outside of the two-sided 95% confidence interval of $Binomial(100, p)$. These results
Figure 5-5. Correctness of three methods over dataset two: Except for the fact that the sampling ratio is set to 0.5%, three other parameters use their default values. strongly support that the three methods for solving the SI problem do in fact produce correct and reliable results.

Furthermore, this low rate of error was observed for all three of the methods, including the two parametric solutions that rely on the Beta distribution – in fact, the rate of errors for the parametric, Beta-based solution was actually lower than for the non-parametric Monte Carlo method. This seems to be strong evidence for the assertion that using the four parameters used in the parametric solution (lower bound, upper bound, mean, and variance) do in fact constrain the space of possible distributions so much that a parametric assumption is quite reasonable, and seems to discount the need for a parametric solution that takes into account even higher moments such as the skew and kurtosis.
Finally, there is still the concern that while the results obtained in this set of experiments do seem to argue convincingly that the SI-based solution to the GROUP BY problem is accurate in practice, they do not necessarily argue that the SI-based solution is actually necessary in practice. It is still reasonable to ask, “How dangerous would it be to simply ignore the correlations among groups, and assume that all groups are independent?”

The first step toward answering this question is to repeat the procedure described above, while ignoring the correlations among the groups. Recall that in the Monte Carlo experiment design, first a sample is obtained from the database. Then for each group, a 95% confidence is interval constructed using the sample. If it is assumed that the intervals are independent, then the probability of seeing exactly $k$ bounds are incorrect follows a
Figure 5-7. Correctness of the three methods over data set four: 20 groups and a skew of 0.3 are used.

Figure 5-8. Correctness obtained assuming independence on data set four. The right plot shows the difference between the SI solution obtained using independence (the solid line) and the Beta approximation (the dash line).
Binomial distribution \( \text{Binomial}(m, 0.05) \), where \( m \) is the total number of groups, and 0.05 is the probability of seeing a single interval wrong. Thus, if assuming independence, the binomial distribution can be used to provide a very simple “solution” to the SI problem by using \( F[k] = \sum_{i=0}^{k} \text{Prob} \left( \text{Binomial}(m, 0.05) = i \right) \). Figure 5-8 shows the accuracy obtained using this binomial “solution” to the SI problem, using the exactly the same experimental setup for the other SI problem solutions with \( p \) in \{0.05, 0.10, ..., 0.90, 0.95\}, and \( N = 100 \) over the fourth data set (the skewness is 0.3, the number of groups is 20). The results showed that for large \( p \) values, the observed value tends to be below the 95% error bar. For example, for \( p = 0.6 \), the 95% error bar covers the range from 50 to 70, but the observed value is only around 45.

Unfortunately, these results do not clearly demonstrate the extent to which a user can be mislead when the independence assumption is used to communicate the accuracy of a GROUP BY query to the user. In Figure 5-8, if the independence assumption were valid, the total number of times that is observed that the actual count of incorrect intervals is smaller than or equal to a given \( k \) (where \( F[k] = p \)) should be approximately \( p \times 100 \). This figure shows that for larger \( p \) values, the number of times that the count of incorrect intervals does not exceed \( k \) tends to be smaller than \( p \times 100 \). Thus, for a fixed \( k \), when using the independence assumption, an estimated probability of seeing \( k \) or more incorrect intervals is generally too small. The problem is that Figure 5-8 does not show exactly the extent to which this probability can be underestimated.

In order to investigate exactly how this probability is underestimated, one of the 100 Monte Carlo trials is arbitrarily chosen from Figure 5-8, and the probability of seeing \( k \) or more incorrect groups obtained using the Beta approximation, as well as the same probability obtained using the independence assumption, is plotted. The results are reported in the right part of Figure 5-8. Only the tail portion of cumulative density function is plotted because observing a large number of incorrect groups is the most worrisome outcome for the user of a sampling-based estimate. The figure shows that the
probability of seeing 14 or more incorrect groups out of 20 is around 0.002 using the Beta approximation, which means that this can actually happen around once per 500 query executions. This is not insignificant, and not communicating this may be dangerous. However, when using the independence assumption, the computed probability is very close to 0. If a user takes this result at face value, he or she will assume that there is no chance of observing that the observed query result is largely useless.

A final issue to point out is that the data is generated using mild zipf skew in order to induce some correlations among the groups. In reality, correlations among groups can be much more significant than those induced by the zipf distribution. Thus the difference between the two probabilities can be even more significant in practice than what was shown here, making the assumption of independence even more problematic in practice.

### 5.5 Conclusion

This chapter has considered in depth the statistical issues that must be addressed if a single sample is used to answer a GROUP BY query. The problem is that since the same sample is used for each group, it is unacceptable to use classic, univariate methods to quantify error since the groupwise estimates are not independent – if one bound is wrong, then it may be likely that all of the bounds are wrong. Statistically speaking, univariate bounds are only valid in isolation, and once a user has seen one of them he or she needs to forget about that bound before looking at the next. Since this is unreasonable in practice, this chapter has considered what information should be given to the user to safely quantify the accuracy of a GROUP BY query result, and also discussed in depth how to compute such information efficiently.
CHAPTER 6
RETURNING TOP-K GROUPS WITH HIGH PROBABILITY

6.1 Introduction

This chapter discusses various algorithms that use a database sample to return the top-k groups from a GROUP BY query with high probability, where “top-k” refers to the best or most interesting groups according to some user defined ranking function. A GROUP BY “top-k” query over n relations $R_1, R_2, ..., R_n$ can be written in SQL as follows:

```
SELECT SUM (f(R_1, R_2, ..., R_n))
FROM R_1, R_2, ..., R_n
GROUP BY group(R_1, R_2, ..., R_n)
HAVING SUM g(R_1, R_2, ..., R_n) IN TOP k
```

The functions $f$ and $g$ may encode any computation, including any join and selection conditions, over each possible combination of tuples from $R_1, R_2, ..., R_n$. $group(R_1, R_2, ..., R_n)$ specifies a grouping operation on any subset of attributes in $R_1, R_2, ..., R_n$. $SUM g(R_1, R_2, ..., R_n)$ is a user-specified ranking function that can be used to determine top-k groups. We say a group is a top-k group if its associated $g()$ value is one of the $k$ largest $g()$ values. The goal of the GROUP BY top-k query evaluation is to return all top-k groups.

When all of the database records can be accessed, the easiest way to answer this query is to first evaluate the underlying GROUP BY query. The results are then sorted based on their associated $g()$ values. Only the top-k groups are retained after sorting. However, this process can be overly time-consuming because it devotes computation to groups not in the top-k. Li et al. [76] studied this problem and proposed a framework that can return top-k groups as early as possible to avoid processing all database tuples.

Unfortunately, in an approximate query processing system, it is not possible to access the entire database. Only database samples are available. We may simply evaluate the
query over samples and return \( k \) groups whose estimated \( g() \) values are in the top-\( k \). But this simple method does not correctly return all of the true top-\( k \) groups.

It is best to illustrate why this simple idea does not correctly return all the true top-\( k \) groups through an example. Let us consider the query: "What were the total sales for the top 2 regions?" If the query is for two relations: PRODUCT and SALES. This query may be written in SQL as:

```sql
SELECT SUM(p.COST), s.REGION
FROM PRODUCT p, SALES s
WHERE p.PROD = s.PROD
GROUP BY s.REGION
HAVING SUM{p.COST} IN TOP 2
```

Imagine that the answer to the query is guessed at by using random samples from the following two database tables:

<table>
<thead>
<tr>
<th>PRODUCT</th>
<th>SALES</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROD</td>
<td>COST</td>
</tr>
<tr>
<td>thingy</td>
<td>$10</td>
</tr>
<tr>
<td>gadget</td>
<td>$6</td>
</tr>
<tr>
<td>widget</td>
<td>$2</td>
</tr>
<tr>
<td>dohicky</td>
<td>$3</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The random samples are used as input into a sampling-based estimator for the answer to the query [49], which has been discussed in Section 3.1. In order to apply the simplest estimator, the samples from each of the two tables would be joined, and the result scaled
up by the inverse of each sampling fraction. Assuming now we take 50% of samples from \texttt{PRODUCT} and 100% of samples from \texttt{SALES}, if only \texttt{gadget} and \texttt{widget} were in the sample of table \texttt{PRODUCT}, the two regions which have the two highest $g()$ value estimates are \texttt{Europe} and \texttt{Africa}. However, the true top 2 regions based on the $g()$ values are \texttt{Asia} and \texttt{USA}. As a result, if we simply return $k$ groups which have $k$ highest $g()$ value estimates, we cannot guarantee they are the true top-$k$ groups which have $k$ largest true $g()$ values.

Since the actual $g()$ value of each group is unknown, in order to return all of the top-$k$ groups with high probability, the size of the return set may have to be longer than $k$.

In this chapter, I develop sampling-based methods that return all of the top-$k$ groups with high probability. A key observation for developing sampling-based methods to return all top-$k$ groups with high probability is that it is very useful to use “confidence regions”. As described in Section 3.3, a confidence region with user defined confidence $p$ is a random region in a multi-dimensional space, such that the probability that the parameter of interest lies in the region is $p$. In this chapter, the parameter of interest is the $g()$ value of each group. I develop three different methods for returning all top-$k$ groups with high probability, where top-$k$ groups are the groups which have $k$ largest $g()$ values. Each of the three methods uses a specific type of confidence region. Each of the three methods also solves an abstract problem called the “region top-$k$” problem, which will be discussed in this chapter subsequently. The region top-$k$ problem provides a framework so that different methods for finding all top-$k$ groups can be easily plugged in. The combination of an input confidence region and a solution of region top-$k$ problem gives a result that contains the top-$k$ groups with high probability.

The remainder of this chapter is organized as follows. Section 6.2 discusses the confidence region and the region top-$k$ problem. It also shows how the confidence region and the region top-$k$ problem may be used for returning all top-$k$ groups with high probability. Section 6.3 proposes three different algorithms for returning all top-$k$ groups with high probability. Each of the algorithms requires a different type of confidence region,
and a different solution for solving the region top-$k$ problem. Section 6.4 discusses various issues for building confidence regions. Section 6.5 experimentally evaluates the utility of the three proposed algorithms. Section 6.6 discusses which of the three algorithms should be implemented for a real world system. Section 6.7 concludes this chapter.

6.2 Confidence Region and the Region Top-$k$ Problem

In order to develop sampling-based methods for returning all top-$k$ groups with high probability, it is useful to build a confidence region that contains the true $g()$ values for all groups with high probability. Before we discuss how a confidence region may be used to help us return all top-$k$ groups with high probability, a few issues need to be clearly explained in advance. Given a database sample, estimating the $g()$ value for the $i$th group can be done by first evaluating the query over samples\(^1\) and then scaling up the result with respect to the inverse of the sampling ratio of each table. If we obtain many database samples and repeat this procedure many times, the distribution of the estimates can be modeled by a normally distributed random variable, denoted by $M_i$.

Define $\overrightarrow{g} = \langle g_1, g_2, ..., g_n \rangle$, where $g_i$ is the $g()$ value of the $i$th group. Then the joint distribution of these random variables, denoted by $M = \langle M_1, M_2, ..., M_n \rangle$, can be used to construct a confidence region for $\overrightarrow{g}$. When I assert that a confidence region contains $\overrightarrow{g}$ with probability $p$, I mean that there is a $n$-dimensional random region such that with probability $p$, it contains $\overrightarrow{g}$. Various methods for constructing different types of confidence regions will be discussed in Section 6.3. Since the estimator $M_i$ for the $i$th group is the $i$th dimension of $M$, throughout of this chapter, I will use $i$ as the identifier for both the dimension and the group. The term “$i$th dimension” and “$i$th group” are used interchangeably.

---

\(^1\) The aggregation function is augmented so that if a resulting tuple is not in the $i$th group, the aggregation function returns 0.
Figure 6-1. Three possible cases for the confidence region, classified by using the relationship between the region and the line \( \text{first-group} = \text{second-group} \): (a) The region is above the line. (b) The region touches the line. (c) The region is below the line.

6.2.1 How Confidence Regions may Help Return all Top-\( k \) Groups

Confidence regions may be used for helping find all top-\( k \) groups. To illustrate how a confidence region may help us find top-\( k \) groups, let us consider a simple case. Imagine that we have a query with two groups, and we are interested in the top-1 group. Regardless how randomly the possible instances of the confidence are placed, it is possible to classify these possible instances into three different cases, determined by whether the region is totally above, is totally below, or touches the line \( \text{first-group} = \text{second-group} \), The three cases are shown in Figure 6-1.

It is important to understand why I classify all possible instances of the confidence region into three cases. Without losing generality, let us first consider the case (a) in Figure 6-1. Since the entire confidence region is above the line \( \text{first-group} = \text{second-group} \), if we pick any point in the region, the value that corresponds to the first group is larger than the value that corresponds to the second group. If \( \overline{g} \) is in the confidence region, then the first group must be the top-1 group, since we already know the \( g() \) value of the first group is larger than the second for any point in the region. Because we know that \( \overline{g} \) is in the confidence region with high probability, if we see the confidence region is above the line
returning the first group as the top-1 group guarantees that we return the true top-1 group with high probability. Case (c) is very similar to case (a), but this time we should return the second group instead. The more difficult case is case (b), because even if we know that $\vec{g}$ is in the confidence region, we do not know which point inside of the confidence region corresponds to $\vec{g}$. Some points in the region may have a larger value for the first group, while others may have a larger value for the second group. Thus, we have to return both groups in order to guarantee that we can correctly return top-1 group with high probability.

6.2.2 The Region Top-$k$ Problem

One way to view the classification from the previous subsection is that I am trying to find the set of all possible top-$k$ groups for each point in the region. In case (a), since the entire region is above the line $\text{first-group}=\text{second-group}$, the set of top-1 groups of all the points in the region contains nothing but the first group. Similarly, in case (c), the set of the top-1 groups of all the points in the region contains the second group. In case (b), we notice that for some points, the top-1 group is the first group, while for others, the top-1 group is the second. Thus, the set of all top-1 groups contains both group.

If we can correctly return the set of the top-$k$ groups for all the points in the region, we can return the true top-$k$ groups with high probability. The reason is that as long as $\vec{g}$ is in the confidence region, its top-$k$ group is in the set. Thus, the top-$k$ groups are correctly returned. I abstract the problem that returns the union of the top-$k$ groups of all the points in the region, and call it the “region top-$k$” problem.

In order to formally define this problem, it is necessary to first define some notation. A point $x = [x_1, x_2, ..., x_n]^T$ in an $n$-dimensional space is a vector of $n$ values, where $x_i$ is the value of the $i$th dimension and $i$ is the dimension identifier. For example, $t = [t_1 = 1.3, t_2 = 2.1, t_3 = 2.4, t_4 = 1.9]^T$ is a point in a 4-dimensional space. Dimension $i$ is a top-$k$ dimension if $x_i$ is one of the $k$ largest values in $\{x_1, x_2, ..., x_n\}$. In the above example, dimension 3 is a top-2 dimension because 2.4 is one of the top-2 largest values in
the vector. I also define top-\(k\) \((x)\) to be the set of top-\(k\) dimensional identifiers of \(x\). In the above example, top-2 \((t)\) = \{2, 3\}.

A region \(T = \{x\}\) is a set of points in an \(n\)-dimensional space. Define \(f\) to be a function that takes a region as input and returns a set of dimension identifiers. For example, considering Figure 6-1, \(f\) may return \{1\} in case (a), \{0\} in case (c), and \{0, 1\} in case (b). Alternatively, \(f\) can be treated as an algorithm that operates on an input region and return a set of dimension identifiers. If the input region is denoted by \(T\), we can denote the return set of \(f\) by \(f(T)\). We say the algorithm, encapsulated by \(f\), is a solution to the region top-\(k\) problem if \(\forall x \in T, \text{top-}k(x) \subseteq f(T)\). For example, the \(f\) function described in this section is a solution to the region top-1 problem.

In order to see why this is the case, we need to prove that for any point in any input region \(T\), its top-1 dimensional identifier is in \(f(T)\). Since these input regions can be classified into three different cases, let us consider each of the three cases, as shown in Figure 6-1:

1. In case (a), for any \(T\), \(f(T) = \{1\}\). For any point \(x\) in \(T\), top-1\((x)\)=\{1\} \(\subseteq f(T)\).
2. In case (b), for any \(T\), \(f(T) = \{0, 1\}\). For any point \(x\) in \(T\), top-1\((x)\) is either \{0\} or \{1\}. Each of them is a subset of \(f(T)\).
3. In case (c), for any \(T\), \(f(T) = \{0\}\). For any point \(x\) in \(T\), top-1\((x)\)=\{0\} \(\subseteq f(T)\).

The solution to the region top-\(k\) problem is not unique. For example, it is also possible to design an algorithm that is always return all dimension identifiers, that is, \(f(T) = \{0, 1\}\) for any input region, and this is also a solution to the region top-1 problem because for any point in the region, its top-1 dimension is for sure a subset of \{0, 1\}. But this solution is not as tight as the previous one. In case (a) and (c), it returns more groups than needed.

6.2.3 Applying the Region Top-\(k\) Problem to the GROUP BY Top-\(k\) Query

A solution to the region top-\(k\) problem may be used to answer the GROUP BY top-\(k\) problem, as described in Figure 6-2. This algorithm takes a confidence region as the input
Figure 6-2. A framework for solving the GROUP BY top-k using a solution to the region top-k problem.

region of the region top-k problem, and returns the set of group identifiers by solving the region top-k problem.

The correctness of the algorithm is obvious. If $\mu$ is in $C$, $\text{top-k}(\mu) \subseteq f(C)$. Since the confidence region has confidence $p$, with probability $p$, $\mu$ is in $C$. Thus, with probability $p$, $f(C)$ contains top-k groups of the GROUP BY query $Q$.  

6.3 Solutions for Returning All Top-k Groups with High Probability

Given the framework described in the previous section, the key to building a solution for returning all top-k groups with high probability is to develop a solution for the region top-k problem, and a method for constructing a specific type of confidence region. This section discusses three different methods for returning all top-k groups with high probability:

1. An interval sweeping algorithm.
3. A two-stage algorithm.

Each of the three proposed methods requires one or more confidence regions as input. Each of the three proposed methods also requires an algorithm for solving the corresponding region top-k problem. I describe each of them in the following subsections.

6.3.1 A Solution Using Interval Sweeping

The first solution is a interval sweeping algorithm. This solution works for the type of axis parallel confidence region shown in Figure 6-1. A benefit of using this sort of region is that only variances, as opposed to all covariances, are needed to build it. Thus, it is computationally inexpensive.
To illustrate the algorithm, I first define some notation. Denote:

\[ l_i = \min(x_i), \quad h_i = \max(x_i), \quad \forall x \in N. \]

Intuitively, \([l_i, h_i]\) is the range of possible values for \(x_i, \forall x \in T\). The algorithm aligns all intervals \([l_1, h_1], [l_2, h_2], \ldots, [l_n, h_n]\) on a single real line. The algorithm sweeps from the rightmost point \(\max(h_i)\) to the leftmost point \(\min(l_i)\). The sweeping procedure stops immediately after there are \(k\) intervals whose lower bounds are larger or equal to the current sweeping point, and returns these \(k\) dimensions, as well as any dimension whose corresponding interval contains the stopping point. Figure 6-3 shows an example of the sweeping procedure in a 4-dimensional space for finding top-2 dimensions. Figure 6-3(a) is shows the starting point of the sweeping procedure. Figure 6-3(b) shows the stopping point of the sweeping procedure. The return set in this example is \{1, 2, 4\}.

The correctness of the algorithm. By construction, the stopping point is the lowest possible top-\(k\) value for any point in the region. The upper bound \(h_i\) for any dimension \(i\) that is not returned by the algorithm is smaller than the lowest possible top-\(k\) value, and thus can be safely removed from the solution. Therefore the set returned contains top-\(k\) dimensions of any point in the input confidence region. Since \(\mu\) is in the confidence region with probability \(p\), the interval sweeping algorithm returns all top-\(k\) groups with probability \(p\).

How to construct the confidence region. Let us denote the confidence region by \(C\). \(C\) is an \(n\)-dimensional random cube, such that each of the edges is parallel to one of the axes. The range \(C_i\) of the \(i\)th group is \((\text{est}_i \pm z_i \sigma_i)\), where \(\text{est}_i, \sigma_i\) are the observed estimate, and standard deviation of the \(i\)th group, respectively. If we denote the accuracy of the \(i\)th group by \(p_i\), \(z_i\) is the corresponding coefficient used for accuracy \(p_i\), which can be compute by using the inverse cdf of the standard normal distribution. I further enforce every group to have the same confidence \(p'\), i.e. \(\forall i \quad p_i = p'\). By using the Bonferroni
Figure 6-3. An example of the interval sweeping algorithm: (a) The starting point. (b) The ending point. The return set is \{1,2,4\}.

Inequality [82], the following holds:

\[
Pr(\forall i \mu_i \in C_i) = 1 - Pr(\exists i \mu_i \notin C_i) \geq 1 - \sum_{i=1}^{n} Pr(\mu_i \notin C_i) = 1 - n(1 - p') \quad (6-1)
\]

Thus, in order to guarantee that \(Pr(\forall i \mu_i \in C_i)\) is at least \(p\), one way is to make sure that \(1 - n(1 - p') \geq p\). This requires \(p' \geq 1 - (1 - p)/n\). Thus, choosing \(p' = 1 - (1 - p)/n\) completes the confidence region construction.

**The limitation of the interval sweeping algorithm.** The interval sweeping algorithm is efficient for the type of axis-parallel confidence region discussed above. However, this is not necessarily the only type of confidence region. When the covariance
Figure 6-4. Three possible relationships between the region and the line \( \text{first-group} = \text{second-group} \). The region is a region such that each of its edges is parallel to one of the eigenvectors. (a) The region is above the line. (b) The region touches the line. (c) The region is below the line. The dash rectangle is the bounding box.

Matrix is available, standard eigenvector/eigenvalue decomposition allows us to build a confidence region such that each of the region’s edges is parallel to one of the eigenvectors. This type of confidence region generally has smaller volume.

When using the interval sweeping algorithm for this type of confidence region, the algorithm is performed on the bounding box of the region. This may result in returning unnecessary groups. Let us consider the following case again: a region in 2-dimensional space, and \( k \) is 1. This time, the confidence region is a rectangle in 2-dimensional space, such that each of the edges is parallel to one of the eigenvectors. Three cases are shown in Figure 6-4.

In this example, the bounding box for the region (the dashed box in Figure 6-4) touches the line \( \text{first-group} = \text{second-group} \) in all three cases. Thus the interval sweeping algorithm returns both groups in any case. However, it is clear that only in case (b) need the algorithm actually return both groups.
6.3.2 A Pruning-Based Solution

In order to deal with this limitation of the interval sweeping algorithm, this subsection discusses a pruning-based algorithm. The pruning-based algorithm works best for the type of confidence region shown in Figure 6-4.

A key observation from Figure 6-4 is that it is possible to rank any given pair of dimensions/groups. For example, in case (a) in Figure 6-4, since for any point $x$ in the region, the value of the first group is always larger than the value of the second group, the first group has a higher rank. That is, if the second group is a top-$k$ dimension, the first group must be a top-$k$ dimension too. For a pair of dimensions $i$ and $j$, $i$ has a higher rank than $j$ if and only if $\forall x \in T, x_i > x_j$. Notice that we may not be able to determine the rank between every pair of dimensions. For example, in case (b) of Figure 6-4, we are not sure which dimension has a higher rank.

The pruning-based algorithm relies on the relative ranks among the dimensions to remove any dimension that we know cannot be a top-$k$ dimension. We notice that if a group is a top-$k$ dimension, any dimension whose pairwise rank is higher than this group must also be a top-$k$ dimension. However, there is at most $k$ slots for the true top-$k$ groups. If for a particular group, we find $k$ or more higher rank groups for it, this group cannot be a top-$k$ group since these $k$ higher rank groups occupy all $k$ slots already. The following theorem formalizes this observation:

**Observation.** Given dimension $i$, if there exist dimensions $j_1, j_2, ..., j_k$, such that $\forall x \in T, x_i < x_{j_1}, x_i < x_{j_2}, ..., x_i < x_{j_k}$, then $i$ is not a top-$k$ dimension for any $x \in T$.

**Proof.** By contradiction. Dimensions $j_1, j_2, ..., j_k$ all have a higher rank than dimension $i$. Thus, if $i$ is a top-$k$ dimension for a point in $T$, then there are at least $k + 1$ top-$k$ dimensions for that point in the region $T$. This is impossible. \(\square\)

Using the above observation, one can easily develop a pruning-based algorithm. The algorithm works as follows. For each dimension $i$ (from 1 to $n$), the algorithm checks how
many higher rank dimensions are available. If there are \( k \) or more higher rank dimensions, this dimension is pruned.

The algorithm is shown in Figure 6-5. The function \( \text{IsSmaller}(i, j, T) \) returns \textit{true} if \( \forall x \in T, x_i < x_j \), and \textit{false} otherwise. The correctness of the algorithm relies on the observation. Any dimension identifier that is not returned by the algorithm is not a top-\( k \) dimension for any point in the given region, and thus is not a candidate of the return set. Since \( \mu \) is in the confidence region with probability \( p \), the pruning-based algorithm returns all top-\( k \) groups with probability \( p \).

Notice that the running time of this algorithm is \( O(n^2) \), where \( n \) is the number of dimensions. The reader may worry that this is inefficient, and ask: Why not improve the efficiency of the algorithm? The reason is that in practice, the algorithm itself is costless compared to the large number of covariance computations that occupies the majority of the running time.

**How to build the confidence region.** Since the eigenvectors are independent of each other, in a transformed space that uses the eigenvectors as bases, the dimensions are independent of each other. If the same accuracy is given to each dimension, denoted by \( \rho' \), then \( \rho'^n = \rho \). Thus picking \( \rho' = \sqrt[p]{\rho} \) for each dimension is sufficient for building the confidence region.
**Function TwoStageGroupByTopk** \((p, S, Q, n)\)

1. Compute variances using sample \(S\) for query \(Q\);
2. Build a confidence region \(C\) with confidence \(p\);
3. \(Candidate = IntervalSweeping(C, n)\);
4. Compute the covariances among the groups in \(Candidate\);
5. Build a confidence region \(D\) with confidence \(p\);
6. \(ReturnSet = PruningTopK(D, n)\);
7. return \(ReturnSet\);

Figure 6-6. The Two-Stage Algorithm.

### 6.3.3 A Two-Stage Algorithm

Although the pruning-based algorithm may outperform the interval sweeping algorithm, computing the entire covariance matrix is expensive. A real-world query may contain thousands or even millions of groups, which makes it impossible to compute the entire covariance matrix. Thus I propose a two-stage algorithm that combines the benefits of the two already described algorithms. In the first stage, the algorithm only requires variances and builds a confidence region using the variances only. The interval sweeping algorithm is used to evaluate this confidence region, and the result set is treated as a set of candidate groups. In the second stage, the covariances among these groups are computed, and another confidence region is built using these covariances. Then the pruning-based algorithm is run using the new confidence region as input. The returned set is treated as the final answer. The pseudo-code of the two-stage algorithm is shown in Figure 6-6.

Notice that the first stage suggests a set of candidate groups. Thus, the correctness of the algorithm only relies on line 5, 6, and 7, and the algorithm is correct because the pruning-based algorithm is correct.

**How to build the confidence regions.** One questions still needs to be answered in order for this algorithm to be practical: How to build confidence region \(C\) and \(D\)? Building confidence region \(C\) has been dealt with using the interval sweeping algorithm. Building confidence region \(D\) is more challenging. For simplicity, I assume that for any
dimension that is not returned by the interval sweeping algorithm in the first stage, the accuracy \( p' \) remains the same when building \( D \). Given this assumption, if the candidate set returned in the first stage is denoted by \( \text{Cand} \), we have:

\[
Pr(\forall i \mu_i \in D_i) = 1 - Pr(\exists i \mu_i \notin D_i)
\]

\[
= 1 - Pr(\exists i \in \text{Cand}, \mu_i \notin D_i \cup \exists j \notin \text{Cand}, \mu_j \notin D_j)
\]

\[
\geq 1 - (Pr(\exists i \in \text{Cand}, \mu_i \notin D_i) + Pr(\exists j \notin \text{Cand}, \mu_j \notin D_j))
\]

\[
\geq 1 - Pr(\exists i \in \text{Cand}, \mu_i \notin D_i) - \sum_{j \notin \text{Cand}} Pr(\mu_j \notin D_j)
\]

\[
= 1 - Pr(\exists i \in \text{Cand}, \mu_i \notin D_i) - (1 - p') \times (n - |\text{Cand}|)
\]

\[
= Pr(\forall i \in \text{Cand}, \mu_i \in D_i) - (1 - p') \times (n - |\text{Cand}|)
\]

(6-2)

In order to guarantee \( Pr(\forall i \mu_i \in D_i) \) is at least \( p \), one way is to make sure that

\[
Pr(\forall i \in \text{Cand}, \mu_i \in D_i) - (1 - p') \times (n - |\text{Cand}|) \geq p.
\]

Thus

\[
Pr(\forall i \in \text{Cand}, \mu_i \in D_i) \geq (1 - p') \times (n - |\text{Cand}|) + p.
\]

Because the covariances of groups in \( \text{Cand} \) have been computed, we may compute the eigenvectors of the sub-covariance matrix formed by dimensions in \( \text{Cand} \), and then build the subpart of confidence regions so that any of these edges is parallel to one of the eigenvectors. Since these dimensions independent, if we assume that the same accuracy is used for each of these dimensions, then the confidence of each of these dimensions is

\[
\sqrt{|\text{Cand}|}(1 - p') \times (n - |\text{Cand}|) + p.
\]

6.4 Various Issues for Building Confidence Regions

In building various types of confidence regions, I assign equal accuracy to each group. One obvious question is whether we can be “smarter”, and make a better assignment so that the total number of groups returned is smaller.
Let us consider a query that asks for the top-1 group among two groups. Currently, we assign each group with equal accuracy $p'$ before we see the sample and construct the confidence region. If we would like to change the accuracy before we see the input sample and construct the confidence region, there is no problem. For example, we can always assign accuracies such that $p_1 = 2p_2$. However this assignment is not always better than the one that equal accuracy is assigned to each group. The second assignment may increase the size of the return set of certain samples but decrease the size of the return set of other samples.

Alternatively, users may think a better way to assign accuracies to different groups is to do this according to the samples. Let us continue the above example. Assume that the variance of every group is 1. According to the sample, the estimates for the two groups are 2 and 3, respectively. Giving the fact that 2 is smaller than 3, the first group can no longer be a top-1 group. Thus, we might let $p_2 = 0$, and assign $p_1$ accordingly. This method does give us the possibility of returning less groups. However, this method is problematic. The coefficient used for determining the width relies on the accuracy assigned to this region. This method has three steps. First, look at the estimates. Second, determine which groups can be potentially dropped. Finally, assign accuracies to different groups so that we can drop as many groups as possible. The problem is that now, the accuracy assigned to each group is no longer a constant, but a function that relies on the estimates obtained from a specific sample. Unfortunately, this adds a new source of randomness to the confidence region that may be impossible to quantify. Thus, the accuracy assignment must be independent from a particular sample. As we have discussed above, if the accuracy assignment is determined before seeing any input samples, a safe assignment is to give each dimension the same accuracy.

Another question is whether we can use alternative shapes for confidence regions. For example, researchers in statistics have studied how to build ellipsoid confidence regions for some time. Stein [97] studied this problem and pointed out that the ellipsoid type
confidence region is recommended for only one dimension or two dimension cases, and that ellipsoid confidence regions constructed using simple samples in high-dimensional spaces are problematic. Stein suggested using a Bayesian estimator in order to obtain a reasonable ellipsoid. The same suggestion has also been proposed by many other researchers [15, 34]. Due to the complexity of such an estimate, I do not consider constructing a ellipsoid type confidence region for high dimensional spaces.

One method worth considering is constructing a series of two dimensional ellipsoids, then combining them together to build a confidence region. For example, for an 100-group query, we can construct an ellipsoid for the first two groups, another ellipsoid for the third and fourth groups, so on. After doing that, we can treat these 50-pairs as 50 “combined” dimensions, and use Bonferroni’s inequality to obtain a single region. Unfortunately, it does not appear that such a confidence region would be very beneficial.

The combined ellipsoid confidence region is constructed using the Bonferroni’s inequality. It does not fully utilize all the correlations among the groups. Therefore, it is expected that this confidence region is not good for the pruning-based algorithm. Thus, the following discussion mainly compares the combined ellipsoid confidence region and the axis parallel confidence region using the interval sweeping algorithm. For the comparison, we construct a 95% axis parallel confidence region, and a 95% combined ellipsoid confidence region using the methods discussed before. Then we compute the coefficient associated to each dimension. Since the coefficient determines the width of each interval, smaller coefficient indicates better results. When using the axis parallel confidence region, the coefficient associated to each group is 3.4. When using the combined ellipsoid confidence region, if all the groups are independent, the coefficient is 3.7. If all the groups are fully positively correlated, the coefficient 3.3. This indicates that the combined ellipsoid confidence region is slightly worse than the axis parallel confidence region if the groups are independent, and slightly better if all the groups are
fully positively correlated. Overall, we do not observe significant benefits for using the combined ellipsoid confidence region.

6.5 Experimental Evaluation

In this section, I experimentally evaluate the utility of the three methods I have proposed to return top-k groups with high probability. I am interested in three questions:

1. For each of the three methods, how much time is required to obtain the result?
2. For each method, how efficient is the return set, in terms of minimizing the number of groups returned?
3. Are each of the proposed algorithms correct?

The answers to the first two questions may be different for different datasets. In general, these answers are determined by the distribution of the ranking scores of the groups, the correlations among the groups, as well as the total number of groups. Therefore, in order to answer the first two questions, I run each of the three methods under different datasets. To answer the last question, the easiest way is to use Monte Carlo methods.

6.5.1 Running Time Experiments

6.5.1.1 Experimental setup

a. Query and Schema Tested. In the experiments, the following TPC-H [100] style SQL statement is used:

```sql
SELECT c_name, sum(l_extprice*(1-l_discount))
FROM supplier s, orders o, lineitem l
WHERE o_orderkey = l_orderkey AND s_suppkey = l_suppkey
GROUP BY c_custkey, c_name
Having sum(l_extprice*(1-l_discount)) in TOP 5
```

The SQL query selects top 5 suppliers by revenues.
b. **Parameters Considered.** There are three key parameters that affect the behaviors of various algorithms: the number of groups, the distribution of the associated $g()$ values, the correlations among group-wise estimates. I discuss each of them below.

The *number of groups* in the query is an important parameter in the experiment. It determines the size of the covariance matrix, which dominates the running time for the non-sweeping algorithms. In order to control the number of groups, I control the total number of distinct suppliers $N_s$ appearing in the dataset.

The *distribution of the $g()$ values* determines the difficulty of the problem of accurately returning $k$ groups. When the larger $g()$ values are much larger than the smaller ones, the problem is easy. This is because we expect that the sample reflects the distribution. When the larger $g()$ values are much larger than the smaller ones, the estimates of the larger $g()$ values are also expected to be much larger than the smaller ones. Therefore, it is very easy to find those groups whose associated $g()$ values are much larger than others. On the other hand, when all the associated $g()$ values are very close, it is rather difficult to figure out which group is a top-$k$ group, because the estimated $g()$ values are close and it is impossible to tell which associated $g()$ value is truly larger than others.

One way to control the distribution of the associated $g()$ values is to control the number of records for each group in the *lineitem* relation. In this experimental setup, I use a zipf distribution to determine how many records for each unique supplier in the *lineitem* relation. The zipf distribution is determined by the number of suppliers $N_s$ and the coefficient $z_s$.

The *correlations among group-wise estimates* affects the accuracy of various algorithms. Intuitively, when the correlations are available, a confidence region with less volume may be constructed. Thus, the correlation-aware pruning-based algorithm may have a better chance to prune more groups. For example, if every group is perfectly, positively correlated with each other, and the variances of the groups are the same, the
confidence region built using eigenvalue eigenvector decomposition is just a line segment in an \( n \)-dimensional space.

In the experimental setup, I control the correlations among group-wise estimates by controlling how important the orderkeys are in the lineitem relation. Intuitively, if a particular order has a significant number of records in the lineitem relation, and these records are equally assigned to different suppliers, there are strong correlations among group-wise estimates. This is because if this important order is not in the sample, each group-wise estimate is underestimated. Otherwise, each group-wise estimate is overestimated. Thus, it is possible to control the correlations among the group-wise estimates by controlling the frequencies of the orders in the lineitem relation. In my experimental setup, the frequencies of the orders are determined by a zipf distribution with the domain size to the total number of orders (1500000), and the coefficient \( z_0 \).

c. Data Generation. Given the total number of suppliers \( N_s \) and the two zipf coefficients \( z_s \), and \( z_o \), I generate the experiment in the following steps:

1. Generate a TPC-H scale 1 dataset.
2. Randomly select \( N_s \) records from the supplier relation and drop all other records in the supplier relation.
3. For each record in lineitem, replace the original orderkey by taking a sample from a zipf distribution with domain size 1500000 and coefficient \( z_o \), replace the original suppkey by taking a sample from a zipf distribution with domain size \( N_s \) and coefficient \( z_s \).

I then use the entire supplier relation, along with a 30% samples from the orders relation, and a 10% sample from the lineitem relation.

d. Hardware and Software used. All the experiments are run over a DELL desktop with an Intel dual core 1.8G CPU and 2GB memory using Open SUSE Linux 11. The codes are written in C++ and compiled using gcc 4.3.2.

e. Experimental Procedure. I prepare several databases for this experiment. For \( N_s = 100 \), I prepare 4 different databases: \( z_s = 0, z_o = 0 \), \( z_s = 0, z_o = 1 \), \( z_s = 0.6, z_o = 0 \),
$z_s = 0.6, z_o = 1$. For both $N_s = 1000$ and $N_s = 10000$, I prepare two databases:

$z_s = 0, z_o = 0$ and $z_s = 1, z_o = 1$, respectively. For each database, I then run the three algorithms 5 times and average the running time.

It is useful to briefly summarize what these databases look like. When $z_s$ is 0, the $g()$ values of the groups are almost identical. When $z_s$ is 0.6, the largest $g()$ value is around 14 times as large as the smallest one. The difference between the two largest $g()$ values is around 40%. When $z_o$ is 0, the correlation coefficients among groupwise estimates are around 0.01, where correlation coefficient is defined as $\text{coef}(u, v) = \frac{\text{cov}(u, v)}{\sqrt{\text{var}(u)\text{var}(v)}}$. When $z_o$ is 1, the correlation coefficients among groupwise estimates exceed than 0.97. The standard deviation (the square root of the standard deviation) of each of the groupwise estimates is around 10% of the corresponding estimate.\(^2\)

### 6.5.1.2 Results

The results are shown in Table 6-1. Running times are given in seconds. Numbers are rounded to the nearest second. When the number of groups is 10000, some setup cannot finish in a reasonable amount of time, resulting in an N/A in the figure. This is because these two algorithm may require operations over a 10000 by 10000 covariance matrix, which is not possible using the experimental hardware.

### 6.5.1.3 Discussion

The running time of the interval sweeping algorithm is always very small, and the running time for other algorithms are short when there are 100 groups. But both of them increase when the number of groups increases. The overall running time is mainly determined by the number of covariance entries that are needed to be computed. Processing the matrix in order to do eigenvalue eigenvector decomposition also requires

---

\(^2\) The variance increases when the correlations among the group increases, when the correlation is very strong, the standard deviation is larger than 10% of the estimate.
Running time for datasets that have 100 groups

<table>
<thead>
<tr>
<th>$z_o$</th>
<th>$z_s$</th>
<th>Interval Sweeping</th>
<th>Pruning-Based</th>
<th>Two-Stage</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>4</td>
<td>21</td>
<td>23</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>4</td>
<td>18</td>
<td>20</td>
</tr>
<tr>
<td>0</td>
<td>0.6</td>
<td>4</td>
<td>22</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>0.6</td>
<td>4</td>
<td>21</td>
<td>8</td>
</tr>
</tbody>
</table>

Running time for datasets that have 1000 groups

<table>
<thead>
<tr>
<th>$z_o$</th>
<th>$z_s$</th>
<th>Interval Sweeping</th>
<th>Pruning-Based</th>
<th>Two-Stage</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>5</td>
<td>221</td>
<td>226</td>
</tr>
<tr>
<td>1</td>
<td>0.6</td>
<td>4</td>
<td>198</td>
<td>29</td>
</tr>
</tbody>
</table>

Running time for datasets that have 10000 groups

<table>
<thead>
<tr>
<th>$z_o$</th>
<th>$z_s$</th>
<th>Interval Sweeping</th>
<th>Pruning-Based</th>
<th>Two-Stage</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>5</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>1</td>
<td>0.6</td>
<td>4</td>
<td>N/A</td>
<td>209</td>
</tr>
</tbody>
</table>

Table 6-1. Running time for two extreme cases for three algorithms. The first case is that the associated $g()$ values are almost identical, and the second case is that the associated $g()$ values are highly separable. Both low correlation and high correlation combinations for 100 groups are provided, and only show the first case with low correlation, the second case with high correlation combinations for 1000 and 10000 groups are also provided.

some time, approximately 1 minute for an 1000 by 1000 matrix. The interval sweeping algorithm is the most efficient algorithm according to its running time.

6.5.2 Efficiency Experiments

I first consider the two cases: when the associated $g()$ values are highly separable, and when the associated $g()$ values are very close. For this purpose, I first use the 8 databases from the previous set of experiments. For $N_s = 100$, I prepare 4 different databases: $z_s = 0, z_o = 0, z_s = 0, z_o = 1, z_s = 0.6, z_o = 0, z_s = 0.6, z_o = 1$. For both $N_s = 1000$ and $N_s = 10000$, I prepare two databases: $z_s = 0, z_o = 0$ and $z_s = 1, z_o = 1$, respectively. I next consider the case between the two extreme cases. In order to investigate the middle case more carefully, I choose $N_s = 100$ and prepare a set of $z_o$ and $z_s$ values so that at each time the interval sweeping algorithm returns approximately 50 groups. But I vary the average correlation coefficient from 0.1 to more than 0.95. Notice that when increasing $z_o$ to increase the correlations among the groupwise estimates, the variances of the groupwise
Table 6-2. Number of returned groups for two extreme cases for three algorithms. The first case is that the associated $g()$ values are almost identical, and the second case is that the associated scores are highly separable.

<table>
<thead>
<tr>
<th>$z_o$</th>
<th>$z_s$</th>
<th>Interval Sweeping</th>
<th>Pruning-Based</th>
<th>Two-Stage</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>0</td>
<td>0.6</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>0.6</td>
<td>10</td>
<td>6</td>
<td>6</td>
</tr>
</tbody>
</table>

Table 6-3. Number of returned groups for the middle case for three algorithms. The total number of groups is 100. The parameters are chosen so that the number of groups returned by the interval sweeping algorithm is approximately 50.

<table>
<thead>
<tr>
<th>$z_o$</th>
<th>$z_s$</th>
<th>Avg Coef</th>
<th>Interval Sweeping</th>
<th>Pruning-Based</th>
<th>Two-Stage</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.7</td>
<td>0.11</td>
<td>0.11</td>
<td>49</td>
<td>100</td>
<td>61</td>
</tr>
<tr>
<td>0.8</td>
<td>0.13</td>
<td>0.35</td>
<td>48</td>
<td>100</td>
<td>55</td>
</tr>
<tr>
<td>0.9</td>
<td>0.15</td>
<td>0.65</td>
<td>51</td>
<td>97</td>
<td>56</td>
</tr>
<tr>
<td>0.92</td>
<td>0.19</td>
<td>0.75</td>
<td>47</td>
<td>90</td>
<td>50</td>
</tr>
<tr>
<td>0.94</td>
<td>0.21</td>
<td>0.83</td>
<td>49</td>
<td>68</td>
<td>49</td>
</tr>
<tr>
<td>0.96</td>
<td>0.23</td>
<td>0.88</td>
<td>50</td>
<td>44</td>
<td>44</td>
</tr>
<tr>
<td>0.97</td>
<td>0.25</td>
<td>0.93</td>
<td>48</td>
<td>38</td>
<td>38</td>
</tr>
<tr>
<td>0.98</td>
<td>0.27</td>
<td>0.97</td>
<td>49</td>
<td>25</td>
<td>25</td>
</tr>
</tbody>
</table>

6.5.2.1 Results

The results are shown in Tables 6-2, and 6-3. When the number of groups is 10000, the pruning-based algorithm and some individual cases of the two-stage algorithm cannot finish in a reasonable time, resulting in an N/A in the figure.
6.5.2.2 Discussion

Several interesting results can be observed, we point out a few significant ones here.

First, in the two extreme cases, all algorithms either return all the groups, or return very few groups, regardless how these groups are correlated. The reason is that when the associated $g()$ values are highly separable, the estimates obtained from the samples are also highly separable. The high separation makes it very easy to find the correct top-$k$ groups. On the other hand, if all the scores are so close, in general, none of the algorithms can figure out which group truly has a higher associated $g()$ values. It is possible in the extreme case, the correlation may help. Recall the example that all the groups are fully positively correlated, and the variances of these groups are all identical. In this case, we only need to return the $k$ groups whose associated $g()$ value estimates are the $k$ largest ones. However, in reality, each associated $g()$ value estimates may have different variances, and they are not fully positively correlated. Thus, it is very difficult for any of the three algorithm to work well if all the scores are very close.

Second, in the middle case, as shown in Table 6-3, the pruning-based algorithm outperforms the interval sweeping algorithm when the correlations among groupwise estimators are strong. Otherwise, the pruning-based algorithm does not help much. The same observation holds for the two-stage algorithm. The reason is that we are required to use the two-dimensional projected shape to rank a pair of groups. When we build an eigenvector parallel confidence region, we do not know how these eigenvectors expand in the $n$-dimensional space. In reality, the two-dimensional projected shape can be arbitrary, especially when the correlations are not very strong. When the correlations are very strong, few important eigenvector dominates the overall space. These eigenvectors are associated with much larger variances. The two-dimensional projected shape for any pair of groups can be more regular. Thus, strong correlations help return less groups. In reality, seeing such strong correlations may not be very frequent. Therefore, the interval sweeping algorithm is preferred unless strong correlations are observed.
Monte Carlo Experiments

To verify the correctness of each of the proposed algorithms, I design a Monte Carlo experiment. I set $N_s = 10$ and choose 5 different values for $z_o$ and $z_s$. These values are $(z_o = 0, z_s = 0)$, $(z_o = 1, z_s = 0.6)$, $(z_o = 0.7, z_s = 0.11)$, $(z_o = 0.94, z_s = 0.21)$, and $(z_o = 0.98, z_s = 0.27)$. The first two datasets cover the two extreme cases. And the last three datasets cover the middle case with low, median, and high correlations. For each dataset, I take 100 independent sample and run each of the three proposed algorithms on each of the 100 samples. I then compute the true top-5 groups and count for each of the three methods how many times the return set correctly contain all top-5 groups. The results are shown in Table 6-4.

6.5.3.1 Discussion

The results show that most of the time, the proposed algorithms correctly return all top-5 groups. In fact the number of times that correctly return all top-$k$ groups is actually larger than 95. This indicates that the confidence of the proposed methods is conservative for normal database queries. The experimental results are expected because when building confidence regions, I require that every score associated with each group to be in the confidence region with high probability. This requirement is very restrictive for most normal database queries. To understand why this is true, consider a confidence region for the point $\overrightarrow{g} = (g_1, g_2, ..., g_n)$, where $g_i$ is not in the confidence region. However some points near $\overrightarrow{g}$ are in the confidence region. The points near $\overrightarrow{g}$ have similar properties to $\overrightarrow{g}$. For example, the top-$k$ groups of such a group may be identical or highly similar to

<table>
<thead>
<tr>
<th>$z_o$</th>
<th>$z_s$</th>
<th>Interval Sweeping</th>
<th>Pruning-Based</th>
<th>Two-Stage</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>1</td>
<td>0.6</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>0.7</td>
<td>0.11</td>
<td>98</td>
<td>100</td>
<td>98</td>
</tr>
<tr>
<td>0.94</td>
<td>0.21</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>0.98</td>
<td>0.27</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>

Table 6-4. Number of times that correctly returns all top-5 groups under 100 independent tries.
the true top-$k$ groups. If we still run any of these algorithms, the set returned contains the
top-$k$ groups of each point in the input region. Since the points near $\overrightarrow{g}$ have top-$k$ groups
that are similar to the true top-$k$ group, most likely, it is likely that the algorithm still
correctly returns all top-$k$ groups, even when $\overrightarrow{g}$ is not in the confidence region.

Building a less restrictive confidence region is possible, but a less restrictive confidence
region may cause problems for unusual cases. Consider an example that we would like to
return the top-5 groups for a 100-group query. Instead of building a confidence region that
contains $\overrightarrow{g}$ with high probability, we build a confidence region that contains the top-5
associated $g()$ values with high probability. This is a less restrictive confidence region
because seeing $\overrightarrow{g}$ in the confidence region with high probability is not required. Any point
which has same dimensional values for the top-5 dimensions in $\overrightarrow{g}$ correctly contain top-5
associated $g()$ values.

If we run any of the three algorithms. Most of the time, it is likely that the
algorithms return the top-5 groups correctly, because with high probability, the points
that correctly contain the true top-5 associated $g()$ values in the confidence region are
points near $\overrightarrow{g}$. However, in rare cases, points that correctly contain the true top-5
associated $g()$ values can be very far away from $\overrightarrow{g}$. In general, the dimensional values
of rest 95 dimensions of these points that correctly contain the true top-5 associated $g()$ values can be arbitrarily large. For example, these dimensional values can be larger than
the top-1 true associated $g()$ values. If the only points in a confidence region that correctly
contain all top-5 associated $g()$ values are these points, we have to return all 100 groups
because the top-1 true associated $g()$ value can appear in any place between the largest
and the 96th largest dimensional values in these points. In reality, we never know which
input confidence region falls into this rare case. Thus, for safety, we must always return
100 groups. This makes any algorithm impractical for this type of confidence region.

Enforcing all the associated $g()$ values in the confidence region eliminates this
problem, because only $\overrightarrow{g}$ contains all associated $g()$ values, and its top-5 dimensional
values are the true top-5 associated $g()$ values. This allows the algorithm to return only top-k groups for each point in the confidence region. A side effect is that the confidence of the returned results may be conservative for normal database queries. But there is no easy way to get around it.

### 6.6 Which Algorithm should be Implemented for a Real World System?

Given these results, I suggest utilizing the interval sweeping algorithm for a real world system, due to three primary reasons. First of all, the interval sweeping algorithm is very easy to implement compared to the two other algorithms, both of which require us to compute the eigenvectors of a matrix. Theoretically speaking, the math for computing the eigenvectors of an input matrix is not difficult. But in practice, to implement a robust algorithm that can handle any special input case is difficult. When I tried to implement this functionality using some open-source libraries, several of them did not work for certain input matrices. A real-world system should be very robust. Implementing an algorithm for computing the eigenvectors can be difficult.

Second, the interval sweeping requires only a few seconds even when the number of groups is large. The other two algorithms requires significantly more time when the number of groups is large. The extra time is mainly because of the covariance computation. Based on my experience, when the matrix is more than 1000 by 1000, the computation becomes inefficient and time-consuming.

Finally, in many real cases, the interval sweeping algorithm generally performs well. Even when the other algorithms are better than the interval sweeping algorithm, the results obtained by the interval sweeping algorithm remain acceptable. Furthermore, many real-world queries may not have very strong correlations. Therefore, for a practical implementation, the interval sweeping algorithm may be preferred.

### 6.7 Conclusion

This chapter has considered the problem of returning all top-k groups with high probability when only database samples are available. I developed three novel methods to
solve this problem. Each method requires constructing a confidence region, and returning
the union of top dimensions for any point in the region. The efficiency, accuracy and
correctness of these methods are experimentally evaluated.
REFERENCES


136


[99] John D. Storey, Jonathan E. Taylor, and David Siegmund. Strong control, conservative point estimation and simultaneous conservative consistency of false


BIOGRAPHICAL SKETCH

Fei Xu was a Ph.D. student in Computer and Information Science and Engineering department at the University of Florida. Fei is a member of the Database Center. Fei is under the supervision of Christopher Jermaine. Fei’s research interest is mainly in data management. During his Ph.D. study, Fei has worked on approximate query processing and uncertain data management.

Fei Xu was born in Lanxi, China. Fei received his bachelor’s degree in 2000 and master’s degree in 2003 from Zhejiang University. Fei joined the University of Florida in August, 2004.