ON UTILIZING NEW HISTOGRAM-BASED METHODS FOR QUERY OPTIMIZATION

by

JING CHEN, B.CS.

A thesis submitted to the
Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Master of Computer Science

Ottawa-Carleton Institute for Computer Science

School of Computer Science

Carleton University

Ottawa, Ontario

January 2003

© Copyright

2003, Jing Chen

National Library
of Canada

Bibliothèque nationale
du Canada

Acquisitions and
Bibliographic Services

Acquisisitons et
services bibliographiques

395 Wellington Street
Ottawa ON K1A 0N4
Canada

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

ISBN: 0-612-83490-5

Our file Notre référence

ISBN: 0-612-83490-5

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

Canada

The undersigned hereby recommends to the
Faculty of Graduate Studies and Research
acceptance of the thesis,

**ON UTILIZING NEW HISTOGRAM-BASED METHODS
FOR QUERY OPTIMIZATION**

Submitted by

Jing Chen, B.CS.

Dr. Frank Dehne

(Director, Department of Computer Science)

Dr. B. John Oommen

(Thesis Supervisor)

CARLETON UNIVERSITY

MARCH 2003

Abstract

We study the problem of query optimization in database systems, and in particular, the problem of utilizing histogram-based methods in the estimation of query result sizes. After conducting a background survey of the literature and some related applied patents, we show how we can utilize two new histogram-based algorithms, namely the R-ACM and the T-ACM, to enhance database query optimization. First of all, we introduce a variant of the R-ACM, namely the Averaged R-ACM, applicable for sparse data cases. We then study how we can improve the feasibility of the T-ACM to a broader field of database applications by controlling the growth trend within each bucket. Finally, and most importantly, we integrate the new algorithms into the ORACLE query optimizer for computing the estimates of the query result sizes, and for selecting the “best” Query Evaluation Plan (QEP). In this context, we verify the consequences of using these techniques on the accuracy of the estimates, on the selection of the access paths, on the join methods used, and on the quality of the QEPs. All of these have been done on the benchmark TPC-H benchmark database.

Dedication

To my family

ACKNOWLEDGMENTS

First and foremost, I would like to thank my Thesis supervisor, Dr. John Oommen, for his generous support, his encouragement, and for believing in me. Throughout the work of this Thesis, his guidance, ideas, and enthusiasm gave me the opportunity to study in an academic field, and to develop and improve my skills and knowledge. His substantial experience and knowledge, and his attitude toward research guided me through this Thesis, and will guide my future study and work. For all of these, I am deeply grateful.

I also would like to thank my family for their continuous support, and encouragement. I would especially like to thank my husband, Tao Wan. Without their support, I would never have achieved this landmark.

Table of Contents

Abstract	iii
Acknowledgements	v
List of Tables	vi
List of Figures	xiii
Chapter 1: Introduction.....	1
1.1 General Introduction to Query Optimization	1
1.2 Justification of the Thesis.....	3
1.3 Objectives and Contributions of This Thesis	5
1.4 Content and Organization of the Thesis	10
Chapter 2: Related Work	12
2.1 Overview of Query Optimization.....	12
2.1.1 SQL Statement Processing	12
2.1.2 Overview of the Query Optimizer.....	15
2.2 SQL Transformation.....	17
2.2.1 Standardization	18
2.2.2 Simplification	19
2.2.3 Amelioration.....	20

2.3 Access Path Selection.....	23
2.3.1 Generate Search Space	24
2.3.2 Access Methods.....	26
2.3.3 Join Orders	29
2.3.4 Join Methods	34
2.3.4.1 Nested-loops Join	34
2.3.4.2 Sort-Merge Join	35
2.3.4.3 Hash Join	36
2.4 Statistics and Cost Models	37
2.4.1 Statistical Methods	38
2.4.1.1 Parametric Methods	39
2.4.1.2 Nonparametric Methods	40
2.4.1.3 Other Estimation Methods	40
2.4.2 Univariate and Multivariate Methods.....	44
2.4.3 Cost Models.....	44
2.5 Dynamic Runtime Optimization	45
2.5.1 Dynamic Hardware Resources Optimization	46
2.5.2 Single SQL Statement Optimization	47
2.5.3 Multiple SQL Statements Optimization	48
2.6 Patent Survey.....	50
2.7 Conclusions	55
Chapter 3: Histogram Methods	57
3.1 Introduction	57

3.1.1 Terminology	58
3.1.1.1 Frequency Distributions of Values	58
3.1.1.2 Histogram	58
3.1.1.3 Selectivity	59
3.1.2 Histograms on Estimation Problems	60
3.1.2.1 Result Sizes of Arbitrary Equality Join and Selection Queries	61
3.1.2.2 Result Sizes of Range Selection Queries	62
3.1.2.3 Set-Valued Queries	63
3.1.3 Histogram Weaknesses.....	64
3.2 Basic Histogram Methods	64
3.2.1 The Equi-width Method	65
3.2.2 The Equi-depth Method.....	66
3.2.3 Variable Kernel Method.....	67
3.3 Self-tuning Histograms.....	68
3.4 New Histogram Method: the R-ACM.....	68
3.4.1 Definition.....	69
3.4.2 Generate the R-ACM.....	71
3.4.3 Query Result Size Estimation.....	73
3.5 New Histogram Method: the T-ACM	73
3.5.1 Definition.....	73
3.5.2 Generate the T-ACM.....	76
3.5.3 Query Result Size Estimation.....	77
3.6 Histogram Comparisons	78

3.6.1 Error Controls	78
3.6.2 Estimate Errors	80
3.6.3 Required Storage Units	82
3.7 Conclusions	83
Chapter 4: Using the R-ACM And T-ACM For Choosing QEPs.....	85
4.1 Determinations for QEP	85
4.1.1 The Statistics	87
4.1.2 Selectivity	89
4.1.3 Cost.....	92
4.2 Using the R-ACM and T-ACM in the Query Optimizer	93
4.2.1 Collecting Statistics for the R-ACM and T-ACM.....	94
4.2.2 Compute Selectivity for the R-ACM and T-ACM.....	95
4.2.2.1 Equi-Selections	96
4.2.2.2 Equi-Joins	97
4.3 The R-ACM in Sparse Data Cases	98
4.3.1 Averaged R-ACM	100
4.3.2 Experimental Results.....	102
4.3.3 Result Analysis.....	106
4.4 Conclusions	108
Chapter 5: Testing With TPC-H Data Sets.....	109
5.1 Testbed Introduction.....	109
5.1.1 Test Database System.....	110
5.1.2 Test Data Sets	110

5.1.3 Sample Queries.....	112
5.2 Test Methods	115
5.2.1 Catalogues	116
5.2.2 Operators	116
5.2.3 Multi-Fractal Distribution	117
5.2.4 Special Experiment of Setup for the T-ACM.....	119
5.3 Test Results	121
5.3.1 Error Rate Growth Experiment	121
5.3.2 R-ACM vs. Equi-depth with Uniform Distributions Experiments.....	123
5.3.3 R-ACM vs. Equi-depth with Multi-Fractal Distributions Experiments	125
5.3.4 BT-ACM vs. Equi-depth with Uniform Distributions Experiments	126
5.4 QEP Selection Comparison	128
5.5 Conclusions	142
Chapter 6: Conclusions.....	144
6.1 Contributions.....	145
6.2 Future Work	147
6.3 Summary	149
Bibliography	150

LIST OF TABLES

Table 1.1 Typical Statistical Information Maintained in a DBMS Catalogue	4
Table 2.1 Transformation Rules for Quantified Expressions.....	18
Table 2.2 Transformation Rules for General Expressions	19
Table 3.1 An Example EMPLOYEE Relation.....	61
Table 3.2 Frequency Distribution of “DEPTNO” for the EMPLOYEE Relation of Table 3.1.....	61
Table 3.3 Notation Associated with the R-ACM	69
Table 3.4 Required Storage Units for the Various Histogram Methods	83
Table 4.1 Results comparing the Equi-width with the Averaged R-ACM for sparse attributes when the number of buckets is the same, and $\mu=0.05$	103
Table 4.2 Results comparing the Equi-width with the Averaged R-ACM for sparse attributes when the number of buckets is the same, and $\mu=0.1$	104
Table 4.3 Results comparing the Equi-width with the Averaged R-ACM for sparse attributes when the number of buckets is the same, and $\mu=0.15$	104
Table 4.4 Results comparing the Equi-width with the Averaged R-ACM for sparse attributes when the number of buckets is the same, and $\mu=0.2$	105
Table 4.5 Results comparing the Equi-width with the Averaged R-ACM for sparse attributes when the storage requirements are the same, and $\mu=0.05$	105

Table 4.6 Results comparing the Equi-width with the Averaged R-ACM for sparse attributes when the storage requirements are the same, and $\mu=0.1$	105
Table 4.7 Results comparing the Equi-width with the Averaged R-ACM for sparse attributes when the storage requirements are the same, and $\mu=0.15$	106
Table 4.8 Results comparing the Equi-width with the Averaged R-ACM for sparse attributes when the storage requirements are the same, and $\mu=0.2$	106
Table 5.1 The TPC-H relations with a total capacity of 500MB data.....	112
Table 5.2.a. Sample Queries for Query Types Q2, Q3, Q5, Q6 and Q7 of the TPC-H Database.....	113
Table 5.3 Results Demonstrating the Growth of the Error Rate with the Number of Joins	122
Table 5.4 Experimental Results of the R-ACM vs. Equi-depth for Uniform Distributions.....	125
Table 5.5 Experimental Results of the R-ACM vs. Equi-depth for Multi-Fractal Distributions ($p = 0.2$)	125
Table 5.6 Experimental Results of the BT-ACM vs. Equi-depth for Uniform Distributions.....	127

List of Figures

Figure 2.1 Schematic of an SQL Processing Architecture	13
Figure 2.2 Operator Tree for a Simple Query	14
Figure 2.3 (a) Left-deep Tree; (b) Bushy Tree	32
Figure 3.1 The Equi-width Histogram for the Attribute DEPTNO of Relation EMPLOYEE in Table 3.1	65
Figure 3.2 The Equi-depth Histogram for the Attribute DEPTNO of Relation EMPLOYEE in Table 3.1	65
Figure 3.3 The Variable Kernel Histogram for the Attribute DEPTNO of Relation EMPLOYEE in Table 3.1	67
Figure 3.4 The R-ACM Histogram for the Attribute DEPTNO of the Relation EMPLOYEE in Table 3.1	71
Figure 3.5 The T-ACM Histogram for the Attribute DEPTNO of the Relation EMPLOYEE in Table 3.1	75
Figure 3.6 The R-ACM Histogram with 5 Buckets for the Attribute DEPTNO of Relation EMPLOYEE in Table 3.1.....	81
Figure 4.1 Query Optimizer Architecture	87
Figure 4.2 Estimations on zero frequencies	107
Figure 4.3 Estimate errors on non-zero frequencies.....	107
Figure 5.1 The Fundamental Core Architecture of Our Implementation.....	115

Figure 5.2 Generation of a Multifractal Distribution – first three steps.....	118
Figure 5.3 The QEPs Chosen for the Sample Query.....	129
Figure 5.4 QEPs for Query Q2.....	131
Figure 5.5 QEPs for Query Q3.....	132
Figure 5.6 QEPs for Query Q5.....	133
Figure 5.7 QEPs for Query Q6.....	134
Figure 5.8 QEPs for Query Q7.....	135
Figure 5.9 QEPs for Query Q9.....	136
Figure 5.10 QEPs for Query Q10.....	137
Figure 5.11 QEPs for Query Q11.....	138
Figure 5.12 QEPs for Query Q12.....	139
Figure 5.13 QEPs for Query Q18'.....	140
Figure 5.14 QEPs for Query Q19'.....	141

Chapter 1: INTRODUCTION

1.1 General Introduction to Query Optimization

A *database system* is a system which stores large amounts of data on one or more computers. The data items are stored based on certain rules, which assure that they can be efficiently retrieved and modified. A *database management system* (DBMS) is the software that manages the database system. Queries (or requests) are sent by users or programs to the DBMS, and the DBMS analyzes the requests, processes them, and sends the results back to the requester. A consistent and robust DBMS is capable of (i) allowing multiple users to have concurrent access to a single database, (ii) restricting access to data to authorized users only, and (iii) maintaining data integrity by providing adequate recovery mechanisms after system failures. Queries are usually presented in some high-level query/data manipulation language, for example, the Structured Query Language (SQL).

Query optimization has been an active research field in the database community ever since the concept of a database was introduced. It is known that the overall performance of a database is determined by both physical and software factors. Generally, the CPU processing speed, the memory capacity, and the speed of transferring

data as dictated by the I/O controller and disks, are considered as physical factors. The architecture and design of a DBMS, and various parameters with regard to its run-time characteristics, are considered as software factors. The performance of a DBMS is, to some extent, represented by how quickly it responds to and processes queries. The goal of query optimization is to minimize the cost for processing and executing a query, and therefore, it is both critical and central to the database's performance.

Query optimization is a highly complex, critical component of any DBMS. There are two distinct categories that can be optimized. These are internal and external to the DBMS in nature.

- (i) Outside the DBMS, in which case the goal of optimizing the database's performance is achieved by optimizing the queries themselves, and
- (ii) Inside the DBMS, in which case the goal is achieved by optimizing the performance of the DBMS itself.

The former optimization can actually be seen to be as emerging from the application's point of view, and requires *prior* knowledge of the contents of the data to be accessed. It also requires substantial skills and experience to accomplish the task. As opposed to this, the latter is more flexible. But by the same token, the latter is harder to achieve because of the complexity of the architecture of the DBMS, and the distinctiveness of different DBMSs. Moreover, the trends toward larger databases and more complex queries make the task even more complicated.

1.2 Justification of the Thesis

In this Thesis, we intend to focus on accomplishing query optimization from the perspective of enhancing the design of the database query optimizer itself. Thus, as such, this study concentrates on the latter category of query optimization strategies alluded to earlier.

The main goals of the database query optimizer are that of minimizing the response time and the resource consumption. Before a query is executed in the DBMS, there are typically two additional phases involved in the processing. First, the query is evaluated and standardized by a “transformer” component, so that it can be recognized more easily and be executed more efficiently. Next, the output of the “transformer” is directed to the most important task, that of query optimization, which is performed by the query optimizer itself. In this step of processing, the query optimizer must attempt to determine an “optimal” Query Evaluation Plan (QEP) for executing the query. A QEP consists of a complete road map indicating the access order and the access paths, which essentially show how the requested data (which are stored in terms of relations in a DBMS) are physically accessed, and the cost associated with each step of the execution. Therefore, the effectiveness of a query is determined by the effectiveness of the selected QEP.

The process of selecting an optimal QEP is yet another complicated procedure. The three phases of processing which are involved in this procedure are:

- (i) Generating the QEP search space which includes all possible QEPs for the given query,

- (ii) Effectively pruning the latter search space, and
- (iii) Choosing one QEP from this space with a, hopefully, minimal cost.

These complex tasks must be completed in a timely and efficient manner to ensure the overall processing time is minimized.

An efficient way to save the time-related costs in the process of query optimization, which is also a method widely adapted by the cost-based query optimizers found in commercial DBMSs, is achieved by consulting various internal catalogues or data dictionaries. This allows the query optimizer to rapidly make fairly efficient decisions, without having to physically access the real data stored in the DBMS. The data which is stored in the catalogues, typically, contains statistical information about the real data stored in the DBMS. Table 1.1 lists some of the statistical information that is commonly stored and utilized in most of the current commercial DBMSs.

Notation	Explanation
N_R	Number of tuples in relation R
b_R	Number of disk blocks storing relation R
s_R	Size of a tuple of relation R in bytes
bf_R	Blocking factor of relation R. This is the number of tuples of relation R that fit into one disk block.
$\Delta(A,R)$	Number of distinct values of attribute A in relation R.
$\Phi(A,R)$	Selection cardinality of attribute A in relation R. Given a relation R and an attribute A of the relation, $\phi(A,R)$ is the average number of records that satisfy an equality condition on attribute A.

Table 1.1 Typical Statistical Information Maintained in a DBMS Catalogue

Since the query optimizer utilizes this information so intricately, the accuracy of the statistical information in the catalogues, and the algorithms used for determining and choosing an optimal QEP, significantly influence the effectiveness of the chosen QEP. Besides the cost considerations, there are also many other aspects that determine the effectiveness of the chosen QEP. Among them, the accuracy of the estimate of the query result size, plays the most important role. It is for this reason, that in the ORACLE DBMS, estimates of query result sizes for each step of the execution are also included in the generated QEP. Since the query result size is the size of the resulting set returned on processing a query, this resultant set could be the *ultimate* result of the query, or an intermediate result. It turns out that the size of a query result determines the choices available for the access paths, the join orders, and the join methods applicable for the query. In this context, access paths are the methods by which the real data stored in the DBMS are physically accessed when executing the query. Thus, as we shall argue later, (and as is well known in the field), the cost of a QEP is directly related to the access paths and the estimates of the query result sizes.

1.3 Objectives and Contributions of This Thesis

As discussed above, there are many aspects which affect the performance of a query optimization module. We intend to focus on algorithms that can be used for estimating query result sizes in this Thesis. Histogram-based algorithms have been widely used in this area, and even in all commercial DBMSs. Two algorithms, namely the Equi-depth and the Equi-width histograms, are the most commonly used ones. For example, the

Equi-depth algorithm is used in both the ORACLE and the SYBASE DBMS. In 1999, Dr. Oommen and his student, Dr. Thiyagarajah, introduced two new histogram-based algorithms, namely the Rectangular Attribute Cardinality Map, the R-ACM, and the Trapezoidal Attribute Cardinality Map, the T-ACM, for estimating query result sizes more accurately. They have also formally proven in [Thi99] that these new algorithms yield superior error-based performance to the traditional ones.

In particular, we would like to concentrate on studying several problems encountered in the design of the database query optimizer, and on finding solutions to them by using the underlying histogram-based algorithms to yield more accurate estimates of the query result sizes. We briefly summarize these problems below.

(i) Problems encountered in the sparse data cases

Tables or relations with sparse data attributes are common in the real-life world. For example, consider the names and telephone numbers of people or companies as listed in a telephone directory. Clearly, the names are comprised of words formed from 26 (or 52) characters, and the telephone numbers are created from the 10 digits. Although these two figures are small, the number of all the possibilities of their combination, is, indeed, exponentially large. This is because the length of names is not fixed, and clearly, any character (or digit) can appear any number of times in any given name or number. However, it is easy to see that the actual number of names (or telephone numbers) that are found in the directory is relatively extremely small compared to the number of all the possible combinations. In other words, the frequencies of a large number of potential strings and digits are zero. The study of sparse-data attributes is scanty. One thing is

known, however. The use of the traditional histogram-based algorithms in such cases, leads to very poor query result size estimates.

In this Thesis, we propose a special data structure, which combines the Equi-width and the R-ACM, to maintain the accuracy of the estimates of the query result sizes for the sparse data scenario. It is called the Averaged R-ACM. Also, we demonstrate the power of the scheme by presenting the experimental results obtained by using the underlying algorithm.

A paper about the Averaged R-ACM algorithm and the associated experimental results, has been submitted for publication [OoC03].

(ii) Problems of the T-ACM

The T-ACM possesses the feature that the shape of the sub-histogram in every bucket in the generated histogram is a trapezoid. This structure is useful in the cases in which the frequency distribution obeys such a requirement, and definitely yields estimates of the query result sizes more accurately for such scenarios. Since determining the type of a frequency distribution *a priori* remains an open problem, the utilization of the T-ACM is a problem that deserves research and investigation.

In this Thesis, we investigate this, and propose a variation of the T-ACM, namely the BT-ACM. In the new, modified generated histogram, frequencies of all the values in each bucket are assured to follow the same growth trend. As results of this, the variation among the frequencies within a bucket is strictly controlled, and the modified T-ACM can be utilized in real-life database applications. The experimental results of this enhancement and its applicability, are also presented in this Thesis.

A paper about integrating the R-ACM and the BT-ACM into the ORACLE DBMS is currently being prepared [ChO03].

(iii) Problems of the accuracy of the estimates of the query result sizes

The accuracy of the estimates of the query result sizes is one of the primary focuses of this Thesis. As discussed earlier, the accuracy of the estimates plays an important role in the performance of the query optimizer. The problem of improving the accuracy has been an active research area. In this Thesis, we would like to validate and demonstrate the superiority of the R-ACM and the T-ACM, (which have been earlier proven to yield more accurate estimates of query result sizes), in a real-life DBMS. Because of the distinctiveness of every commercial DBMS, we have opted to choose the most prominent one, the ORACLE DBMS, in our implementation. The performance of the R-ACM and the T-ACM in the ORACLE DBMS is tested and compared with that of the traditionally adapted Equi-depth algorithm.

(iv) Problems of “the” optimal QEP selection

The QEP selection is another main focus of this Thesis. As we explain later, the QEP selection involves many different aspects, and is a significantly complicated task. The quality of the chosen QEP directly affects the effectiveness of the query. As alluded to earlier, the estimates of query result sizes are related to QEP selection. We would like to verify such a relationship by utilizing different histogram-based algorithms which yield different estimates for the same query. Moreover, we would like to validate that, in a real-life DBMS, the accuracy of the estimates has a serious consequence on the selection of the access paths, and on the join methods and join orders.

Thus, in this context, apart from the theoretical results which were proven in [Thi99], in this research, we desire to obtain additional experimental results related to using these newly- introduced algorithms in a real DBMS. The subject of this Thesis is to utilize these two histogram-based algorithms, in the query optimization of a real-life DBMS. From this perspective we can say that the main objectives are summarized below:

- To incorporate the R-ACM and the T-ACM into a real database system, namely, the ORACLE database system;
- To test and demonstrate the performance of the R-ACM and the T-ACM in choosing QEPs and estimating query result sizes, by utilizing them in the ORACLE optimizer and by using real-life data;
- To compare results with those obtained by the commercial algorithm, namely, the Equi-depth algorithm, which is currently central to the ORACLE optimizer.

In this regard we intend to integrate the R-ACM and the T-ACM into the ORACLE query optimizer. We shall utilize data dictionaries to store the relevant statistical information, which is done in exactly the same way that ORACLE utilizes the statistics. Furthermore, we shall use the R-ACM and the T-ACM schemes to generate the required statistical information. In other words, in the process of query optimization, we shall require that the ORACLE query optimizer uses the statistics provided by the customized algorithms, rather than those provided by the original algorithms.

To verify our results, we shall conduct extensive experiments to demonstrate the performance of the new algorithms in estimating query result sizes and choosing QEPs. The experiments are conducted on the ORACLE 9i system with 500MB volume data set

obtained from the TPC-H database. The superiority of *both* the R-ACM and the T-ACM shall be demonstrated by the experimental results.

1.4 Content and Organization of the Thesis

This Thesis begins with an overview of the field of query optimization in Chapter 2, highlighting almost every operation performed inside the query optimizer. The aim of the Chapter is to present an overview perspective of how the query optimizer works. Since each operation is critical to the overall performance of the query optimizer, we state almost all the aspects involved, and cover numerous research results in each area. As part of the background research work, we also present a survey of the applied patents which have used technologies similar to those studied in this Thesis, namely the histogram technologies.

Chapter 3 introduces the new algorithms, namely the R-ACM and the T-ACM, which are primarily studied in this Thesis. This is done in the context of the statistical methods used in database systems.

Chapter 4 presents how the R-ACM and the T-ACM are integrated into the database query optimizer in our implementation. Also presented in this Chapter is a report of the quality of the statistics obtained for a real-life DBMS. In addition, we include a new exploration of utilizing the R-ACM in cases where the attributes are sparsely populated.

The experimental results are presented in Chapter 5, together with the test methods used and a brief comparative analysis. More importantly, the experimental

results obtained by using the R-ACM and the T-ACM are shown to be far superior to those obtained by using the Equi-depth scheme.

Chapter 6 concludes the Thesis by presenting a summary of the results, outlining the main conclusions of this work, and listing the potential areas for future research in this field.

Chapter 2: RELATED WORK

2.1 Overview of Query Optimization

In this Chapter, we will briefly discuss some of the important research results and specific techniques that have been developed in query optimization. This is intended to serve as a brief overview of this research field. Clearly, since there are thousands of papers written in this field, an exhaustive survey is impractical.

2.1.1 SQL Statement Processing

SQL (Structured Query Language) is an industry-standardized interactive programming language used to get information from, and to update a database. It provides natural-language-like syntax to construct statements for querying, defining and manipulating data in databases. Queries take the form of a command language that provides functions such as selecting, inserting, updating, retrieving the location of data, and so forth. Usually, there is a programming interface for the user to input queries. Also, queries can be embedded in general purpose programming languages. These queries, in a command language form, are called SQL statements (see [SAC79], [Cha98], [Ora02], [Orb02],

[LoT00], [JaK84], [Ioa96], [StF95], [GoG96]). The syntax and use of SQL can be found in any standard database textbook.

In a DBMS, processing an SQL statement is composed of 4 phases: parsing, optimization, code generation, and execution. The role of each of these modules is explained below.

Figure 2.1 Schematic of an SQL Processing Architecture

Parsing:

The parser performs syntax analysis and semantic analysis. Each SQL statement is sent to the parser, where the syntax of the SQL statement is investigated. According to the syntax defined in SQL, each SQL statement of the query contains at least a SELECT list, a FROM list, and optionally, a WHERE list, a GROUP BY list, a HAVING list, and an ORDER BY list. Complex queries usually contain compound SELECT lists in WHERE lists or FROM lists, or have multiple relations to be accessed in WHERE lists.

Optimization:

The optimizer uses internal cost estimation models to determine the most efficient way of producing the result requested by the query. The selected (hopefully, optimal) plan contains the access path of how to access relations in the database. A physical operator tree can visually illustrate the access path (see Figure 2.2), where the structure which shows how to process the query, yields the so-called Query Evaluation Plan (QEP). The edges in the operator tree represent the data flow among the physical operators from bottom to top. The “operators” are the physical operators to be executed in the execution engine.

Figure 2.2 Operator Tree for a Simple Query

The optimizer is entirely transparent to applications and end-users. The goal of the optimizer is to ensure good performance for each query, so that it provides excellent overall performance for the DBMS.

The query optimizer is typically a ‘cost-based’ module. In a cost-based optimization strategy, multiple execution plans are generated for a given query, and

then for each plan an estimated cost is computed based on the cost model. The query optimizer chooses the plan with the lowest estimated cost. Various cost considerations can be used and these depend on which cost model the optimizer uses. Generally, there are two major category costs: (i) hardware costs, such as I/O speed and capacity, CPU speed and utilization, and memory capacity and allocation, and (ii) statistics costs, which involve statistical information on the relations and columns of database. Usually, the statistical information is stored as special relations (called the data dictionary), in the DBMS, along with other data.

Code Generation:

The code generator receives the generated, hopefully optimal, QEP from the optimizer, and outputs the execution plan for the SQL statement. In other words, the QEP is translated so as to be ready for computer execution.

SQL Execution:

The SQL execution engine is the component that ultimately operates on the real database, and performs the execution plan associated with the SQL statement. It scans each of the physically stored relations in the query along the access paths chosen by the optimizer, and then produces the result of the query.

Our research focuses on the query optimization task, which is achieved by the optimizer. We shall explain the optimizer in detail in the following subsections.

2.1.2 Overview of the Query Optimizer

The query optimizer attempts to determine the most efficient way to execute an SQL statement. This determination is an important step in the processing of any SQL

statement. Almost without exception, an SQL statement can be executed in many different ways; for example, tables or indexes can be accessed in different orders. However, the way in which a DBMS executes a statement can greatly affect the execution time. The objective of the query optimizer is to find the most cost-efficient way to execute a query, this task itself being achieved in a reasonable amount of time.

For any SQL statement processed by a DBMS, the query optimizer usually needs to perform the following operations:

- (i) **Evaluation of expressions and conditions:** The query optimizer evaluates expressions and conditions containing constants as much as possible.
- (ii) **Statement transformation:** Based on transformation rules defined in the DBMS, the original SQL statements are transformed into standardized and effective ones. For complex statements involving, for example, correlated subqueries or views, the query optimizer might transform the original statement into a semantic equivalent join statement to yield better performance.
- (iii) **Selection of access methods:** For each relation or index accessed by the statement, the query optimizer chooses one or more of the available access paths to obtain the required data.
- (iv) **Selection of join orders:** For a join statement that joins more than two tables, the query optimizer decides on the sequence of join operations. It determines which pair of tables is joined first, and then which table is joined to the result, and so on.
- (v) **Selection of join methods:** For any join statement, the optimizer chooses the operation to be used in performing the join.

Each step is a complicated procedure involving numerous sophisticated techniques, and may have direct or indirect influence on the overall performance of the query optimizer. We shall discuss each step of these processing modules next.

2.2 SQL Transformation

SQL transformation is a procedure in which the query optimizer transforms the original SQL statement into a semantically equivalent SQL statement, which is more efficient. It is one of the important components of the query optimizer. A variety of sophisticated techniques have been developed and applied to the original statement ([JaK84], [Ioa96]).

Generally, every query can be expressed in a number of different representational forms. In other words, these different representations are semantically equivalent in a query expression language such as SQL. However, their individual complexity and efficiency can be different, and sometimes even vastly different. Therefore, it is necessary to rewrite queries sent to the query optimizer if their syntax is correct. The goal of SQL transformation is to make SQL statements more efficient by utilizing sophisticated techniques, including query caching, query folding, and semantic query optimization.

Query transformation transforms a given query expression to a semantically equivalent expression based on well-defined rules. To achieve this goal, three steps of processing on SQL statements are needed, namely, (i) standardization, which constructs the standard form of the query, (ii) simplification, which eliminates the redundancies in

the predicate, and (iii) amelioration, which constructs expressions that are evaluated to be more efficient.

2.2.1 Standardization

Transformation rules are defined as illustrated in Tables 2.1 and 2.2. The rules are quite straightforward, and are in a relational calculus representation. By applying these rules in query transformation, queries with arbitrary expression forms are transformed to standard forms as the input to a DBMS. The standardization is useful for the decomposition of queries and for data-dependent amelioration.

	Arbitrary Expressions	Standard Expressions
Q1	A AND SOME r IN rel ($B(r)$)	SOME r IN rel (A AND $B(r)$)
Q2	A OR SOME r IN rel ($B(r)$)	
	rel $\neq []$	SOME r IN rel (A OR $B(r)$)
	rel = []	A
Q3	A AND ALL r IN rel ($B(r)$)	
	rel $\neq []$	ALL r in rel (A AND $B(r)$)
	rel = []	A
Q4	A OR ALL r IN rel ($B(r)$)	ALL r IN rel (A OR $B(r)$)
Q5	SOME r_1 IN rel1 SOME r_2 IN rel2 ($A(r_1, r_2)$)	SOME r_2 IN rel2 SOME r_1 IN rel1 ($A(r_1, r_2)$)
Q6	ALL r_1 IN rel1 ALL r_2 IN rel2 ($A(r_1, r_2)$)	ALL r_2 IN rel2 ALL r_1 IN rel1 ($A(r_1, r_2)$)
Q7	SOME r IN rel ($A(r)$ OR SOME r IN rel ($B(r)$))	SOME r IN rel ($A(r)$) AND $B(r)$)
Q8	ALL r IN rel ($A(r)$ AND $B(r)$)	ALL r IN rel ($A(r)$ AND ALL r IN rel ($B(r)$))
Q9	NOT ALL r IN rel ($A(r)$)	SOME r IN rel (NOT($A(r)$))
Q10	NOT SOME r IN rel ($A(r)$)	ALL r IN rel (NOT($A(r)$))

Table 2.1 Transformation Rules for Quantified Expressions

M1: Commutative rules	
a) A OR B	B OR A
b) A AND B	B AND A
M2: Associative rules	
a) (A OR B) OR C	A OR (B OR C)
b) (A AND B) AND C	A AND (B AND C)
M3: Distributive rules	
a) A OR (B AND C)	(A OR B) AND (A OR C)
b) A AND (B OR C)	(A AND B) OR (A AND C)
M4: De Morgan's rules	
a) NOT (A AND B)	NOT (A) OR NOT(B)
b) NOT (A OR B)	NOT (A) AND NOT(B)
M5: Double negation rule	
NOT (NOT (A))	A
M6: Idempotency rules	
a) A OR A	A
b) A AND A	A
c) A OR NOT(A)	TRUE
d) A AND NOT(A)	FALSE
e) A AND (A OR B)	A
f) A OR (A AND B)	A
g) A OR FALSE	A
h) A AND TRUE	A
i) A OR TRUE	TRUE
j) A AND FALSE	FALSE

Table 2.2 Transformation Rules for General Expressions

2.2.2 Simplification

Since there are many different semantically equivalent expressions representing a single query, one of the goals of query transformation is to transform the original query to one which has the least redundancy. Redundancy is considered as a way of wasting resources and would thus cause bad performance for most cases. For example, the original query is:

```
SELECT ename
FROM emp
WHERE ename='SMITH' OR (job='MANAGER') AND NOT (job='MANAGER');
```

By means of rules M6d and M6g (Table 2.2), the predicate in this SQL statement could be simplified to:

```
ename='SMITH'.
```

Therefore, the query is transformed to:

```
SELECT ename  
FROM emp  
WHERE ename='SMITH';
```

It can easily be seen that the expression form after the transformation is much simpler than the original one, which means that the new expression could, most likely, have a better performance over the previous one.

2.2.3 Amelioration

The objective of ameliorating transformation is to transform the simplified query into one with a superior performance. The reason for this is that there could be many, non-redundant expressions and their individual performances could be quite different. Indeed, this is true in terms of the size of intermediate results, and the number of relations accessed. There are three sources of information used by the ameliorating transformation: general transformation rules guided by well-documented heuristics, knowledge about the relational data structures, and the query itself. Recent research focuses on developing new rules other than those listed in Table 2.1 and Table 2.2, and involves new areas such as Boolean value evaluation and transformation ([CKK00]). They also involve studying factors within database management systems such as integrity constraints that complement the structural schema definition [GLQ99]. A bottom-up query evaluation strategy was introduced in [GGM96], which differed from the traditional top-down

strategies. This paper focused on a single optimization technique, namely, join elimination. This new strategy minimized the *number* of elementary joins and unions, and provided a compact form of query representation, so that it could achieve superior overall performance when compared to previous strategies. The paper [GGM96] also presented two different abstract algorithms for optimization. One pre-processed a query statically before it was evaluated. The other combined query evaluation with semantic optimization using heuristics to achieve the largest possible savings.

The techniques discussed above are called heuristic query transformations, which always provide equivalent or better query performance after the transformation. While queries are becoming more complex and the data volume stored in the database system is becoming increasingly larger, many new techniques related to query transformations are developed to meet the new challenges. A materialized view rewrite technique has then been developed for large data volume database system, such as those estimated in a data warehouse system. Usually, materialized views involve precomputing, and utilizing commonly used data, which might be summaries or other aggregate queries. If an arbitrary query is transformed to reference to a materialized view, and if the materialized view is smaller than the original table or tables, or has better available access paths, the transformed SQL statement could be significantly superior, and lead to a much better performance ([GoL01], [ZCL00]). For example, if a materialized view is initially created as follows:

```
CREATE MATERIALIZED VIEW SALES_SUMMARY
AS SELECT SALES.CUST_ID, TIME.MONTH, SUM(SALES_AMOUNT) AMT
FROM SALES, TIME
WHERE SALES.TIME_ID = TIME.TIME_ID
GROUP BY SALES.CUST_ID, TIME.MONTH;
```

Then, this materialized view can be used to optimize the following query:

```
SELECT CUSTOMER.CUST_NAME, TIME.MONTH,  
       SUM(SALES.SALES_AMOUNT)  
FROM SALES, CUSTOMER, TIME  
WHERE SALES.CUST_ID = CUSTOMER.CUST_ID  
GROUP BY CUSTOMER.CUST_NAME, TIME.MONTH;
```

The transformed SQL statement would be as below:

```
SELECT CUSTOMER.CUST_NAME, SALES_SUMMARY.MONTH,  
       SALES_SUMMARY.AMT  
FROM CUSTOMER, SALES_SUMMARY  
WHERE CUSTOMER.CUST_ID = SALES_SUMMARY.CUST_ID;
```

In this example, the transformed query is expected to be executed much faster because: (i) the SALES_SUMMARY table is likely much smaller than the SALES table since it stores summary data rather than the detailed sources of the summary, and (ii) the transformed query requires one less join and no aggregation operations.

Other non-transformation-based techniques have also been developed because a complete set of transformations does not necessarily guarantee good performance, and it is therefore sometimes necessary to add redundant transformations to improve the performance of SQL statements. In [LPK94], an algorithm differing from traditional transformation-based algorithms was introduced to choose optimal QEP for joins. The theory motivating the proposed technique was that the ratio of randomly choosing a good candidate plan in the search space was not insignificant. (In transformation-based algorithms, exhaustive exploration or sampling of the search space of a class of queries provides a precise measure to hit a good plan.) However, the algorithm relied on both an accurate estimation of query evaluation cost and an efficient mechanism to generate query plans that are uniformly distributed over the search space. The problem of

efficiently generating uniformly distributed random plans for queries with acyclic graphs is also solved in this paper.

2.3 Access Path Selection

Access path selection methods are techniques that specify how to access data physically stored in a database. Additionally, other decisions are to be made while choosing an access path for join operations. Examples of these are the order in which the joins of the relations are done in the query, and the join methods to be applied to the physically accessed data. As explained earlier, a QEP actually contains an access path for the query. Since there are a number of available access paths for one query, there is a number of QEPs associated with that query.

The goal of access path selection is to select a optimal access path, or QEP, for the query. Hopefully, it is “the” optimal one. Access path selection algorithms are especially sophisticated. One of the reasons for this is that it involves many database structures and query evaluation techniques. Join-order optimization is even proven to be NP-hard [IbK84]. Another issue that arises is the cost model on which the access path selection is based. This model must incorporate a complete understanding of each of these features, or else the optimal access path will not be selected. To clarify these issues, we shall take a brief look at the components which constitute a QEP.

Database structures: These could be relations, views or indexes. As we know, the data itself is physically stored in relations. Relations and views are the most commonly and

basically accessed structures in a DBMS. An Index is accessed when the correspondent index method is an access path in the QEP.

Access methods: These are usually either full table scan, the index or the rowed schemes. We shall discuss each of them in detail presently.

Join methods: The basic methods used here are nested-loop joins, sort-merge joins and hash joins. These are the fundamental methods used to perform joins.

2.3.1 Generate Search Space

To choose an access path, the query optimizer should be able to know, first of all, what the available access paths are. This is determined by examining the relations in the SQL statement's FROM clause and/or the conditions in the WHERE clause, and by consulting the related catalogues. The query optimizer then generates a set of possible execution plans consisting of available access paths, and estimates the cost for each plan based on the cost model using the statistics for the index, columns, and relations accessible to the SQL statement. Finally, a cost-based query optimizer chooses the execution plan with the lowest estimated cost.

The available access paths for each relation are stored in catalogues of a DBMS. They can be easily obtained from these catalogues whenever they are required. Once there is a new access path created for a relation, or there is an access path deleted, the catalogues are automatically updated by the DBMS to maintain consistency and integrity.

The next step, which is to estimate the cost of each plan, is one of the most difficult decisions made by the query optimizer. There are many factors to be considered in the cost model, such as hardware considerations: processing speed of CPU, capacity of

memory and access speed of I/O channels, and statistical information involving the real instantiation of the database. We shall discuss them in more detail in Section 2.4.

All the available access paths or methods for a query constitute the search space for the query optimizer from which the desired QEP has to be chosen. A QEP can also be represented by the operator tree (see Figure 2.2). The search space can be very large for complex queries. Sometimes it is impractical for the query optimizer to scan the search space exhaustively because of the time limit placed on the query optimizer. Most commercial database systems use variations of the same query optimization algorithm, which performs an exhaustive search over the search space, and whenever possible, uses heuristics to reduce the size of that space. The question of how to efficiently generate a search space containing optimal access paths while keeping its size small and correctly pruning it remain very active research topics in query optimization.

In [ChS95], an optimized algorithm was introduced to optimize queries with both aggregates and joins for single block SQL queries. Chaudhuri and Shim observed that traditional SQL semantic transformation deferred execution of grouping after the execution of joins. In some cases, where aggregates would reduce the size of input to the join significantly, optimal QEPs are not included in the search space because the query optimizer adopts traditional SQL transformation. Additionally, in other cases, the queries contain views with aggregates. In the latter cases, the aggregate clause prevents reordering the relations within the view with relations outside. Therefore, the authors proposed a new algorithm - the pull-up transformation. In other words, they advocate doing aggregations before the joins. They showed that a rich set of access paths could be generated by the new algorithm to significantly enhance the quality of the search space.

In [WaG00], Waass and Galindo-Legaria showed how QEPs were generated and how to evaluate each QEP in the search space. QEPs from the complete search space were enumerated, after being transformed, and are counted and ranked. In addition to that, each QEP or specific QEPs could be evaluated in the implemented system. These techniques allowed user to evaluate various QEPs which were associated with a single query, instead of evaluating only one selected QEP. Thus they propose a useful tool by which one can evaluate whether the algorithms used by the query optimizer are good enough to generate high quality QEPs, and to correctly and efficiently prune the search space.

A new search space pruning algorithm was presented in [BFI91]. Bennett, Ferris and Ioannidis used genetic programming, which was a tool different from those used previously, namely, the dynamic programming algorithm, introduced in System-R. Three different strategies were implemented by the proposed algorithm: left-deep, bushy and cross-over. The new algorithm overcame the main disadvantage of the previous one, which needed to maintain and process a very large number of strategies during its execution. It made the new algorithm applicable and capable of optimizing complex and large size queries, such as queries with more than 16 relations, on which the dynamic programming algorithm failed.

2.3.2 Access Methods

Access methods deal with fundamental components that constitute an access path. The requested data of a query can be retrieved by one of the following methods:

- Full table scan: The scan reads all tuples from a relation in the SQL statement and filters out those that do not meet the selection criteria.
- Index: In this method, a tuple is retrieved by traversing the index, using the indexed column values specified by the statement.
- Rowid: Specifying the rowid is the fastest way to retrieve a tuple, because the exact physical location of the tuple in the database is specified. This is one of the access methods utilized by some commercial DBMS, such as ORACLE.

Different access paths for a particular query will cause huge differences in terms of the response time, and thus to the performance of the system to that particular query. To improve query performance, we definitely want to make sure that the query optimizer attempts to choose the most efficient access path for every query passed on to the DBMS.

Full table scan is a method which scans all tuples in the given relation(s). It fetches the tuples from the relation(s) one by one, checks if they meet the predicates in the WHERE clause, and invokes further operations, such as joins. Since a table (or relation) and a view are two major objects in a DBMS, full table scan implies full view scan when the scan is performed on a view. To simplify discussions, we use the term “relation” instead of “relation/view” in the entire context, and we use the term “view” explicitly for the actual view.

Index scan is a method in which scans are made on indexes instead of relations or views. These indexes are stored traditionally in the form of B-tree on pages separate from those containing the relation tuples, and are created on one or more columns of a relation. A relation may have any number of indexes on it. Indexes are created to improve the

search speed for a particular tuple in a relation, and it is well known that searches which are made against a B-tree are usually much faster than those made by sequential searches.

In general, index access paths are used for statements that retrieve a small subset of a relation, while full table scans are more efficient when accessing a large portion of the relation. For example, some applications like online transaction processing (OLTP), which consist of short-running SQL statements with high selectivity, are often characterized by the use of index access paths. On the other hand, decision support systems (DSS) tend to partition tables and perform full scans of the relevant partitions. The advantage of the Rowid method cannot be easily capitalized on, unless the rowid(s) of requested tuples is/are known *a priori*, and hence this method is used only in specific settings.

The structure of an index need not necessarily be the traditional B-tree. Other special structures have been developed to overcome the inefficiencies of some particular operations, such as joins. Join indexes, Bc-Trees, T-Trees, Kd-Trees, and prejoins are structures tailored especially for joins by providing fast accesses to relations. Kd-Trees can be applied to cases where overlapping partitioning of relations exists. The join indexes and the Bc-Trees are useful in situations where the relations are joined often and rarely updated.

In data warehouse systems, aggregate queries are used a lot. Therefore the structure of the index is specifically tailored for the application. In [ZMT00], a new index structure was proposed, which incorporated features from both the SB-tree and B-tree, to handle temporal aggregates with range predicates. [Wu99] studied both continuous and discrete selections using bitmaps, which is a popular index structure mostly used in data

warehouse applications. New algorithms were presented to optimize selections statically and dynamically, by using different variations of bitmaps, and previously known methods such as the *ordered binary decision diagram* (OBDD) and the *tabular method*.

In [HNP95], a new balanced search (GiST) tree was introduced. The GiST supported general data types, set-valued data types, besides the traditional number and string types that are indexed. It also supported Boolean queries. The complexity for searching a GiST is $O(\log n)$, if keys on the nodes do not overlap. GiST would be more useful to Object Oriented Databases, since they support more generalized data types and queries. Similarly in [Kor99], a domain index based on GiST was presented, and the new technique was implemented in the Informix Dynamic Server with the Universal Data Option.

2.3.3 Join Orders

To execute an SQL statement that joins more than two relations, two of the relations are joined first, and then the resulting tuple set is joined with a next relation. This process continues until all relations are joined into the result. Typically, a join between n relations is executed as a sequence of $(n-1)$ two-way joins [MaL96]. Also, there are many kinds of joins, such as equijoins, non-equijoins, etc. In [CCK00], different types of join were studied: equijoins, spatial-overlap joins, and set-containment joins. The results showed that equijoins were the easiest to optimize, while spatial-overlap and set-containment joins were the hardest ones.

Joins are always complex operations in a DBMS. That is the reason why joins are one of major operations studied in the area of query optimization. Processing joins need

much more hardware and time resources than other operations, such as selection, or projections. CPU and memory utilization and optimization for equi-joins in database systems are discussed in a detailed manner in [MBK00]. The discussion concentrated on issues of CPU and memory optimization. New algorithms, which strongly accelerated large equi-joins by tuning the memory access pattern to match the characteristics of the memory cache subsystem in the hardware, were also presented, and they were applicable on a wide variety of platforms.

Complex queries always involve join operations. Sometimes there are a large number of relations involved in the join. Consider one of extreme cases which perfectly shows the importance of join order for query performance when a number of relations have to be pair-wise joined. There are two steps to evaluating an optimal execution plan when performing joins, namely to generate all possible join orders and then to evaluate the cost (or all) of these orderings. Generating all the join orders involves considering the factorial number of relations involved in the join of the SQL statement, $(2(n-1))!/(n-1)!$. For example, for a query with 5 relations in the join, there are $8!/4! = 1680$ possible join orders. Every single join order may compose many execution plans if there are multiple available access paths for the relations involved. That means that the search space would grow even larger. After evaluating each or some of all the generated execution plans, the query optimizer chooses the one with the minimum cost. These are, therefore, two problems, similar to those in the search space discussed above. These involve the question of how to deal with the large search space in terms of storage space, CPU and memory resources, selection algorithms, etc, and the question of whether it is possible to evaluate all the QEPs in the search space exhaustively.

Since there is a time limit for cost-based optimization, it is not feasible to explore the search space exhaustively. Thus, maintaining a small search space is a critical factor to improve the performance of the query optimizer. Consider, now, an example to see the importance of join orders. Assume that there are 3 relations to be joined together. Relation A has 100 tuples, relation B has 10 tuples, and relation C has 10000 tuples. The search space for the join orders is $4!/2! = 12$. One of the possible plans is: $((A \bowtie B) \bowtie C)$, with an intermediate join result of $100 * 10 = 1,000$. Another possible plan is $(B \bowtie (A \bowtie C))$ with its intermediate join result of $10000 * 100 = 1,000,000$ ¹. As we see, there is a tremendous difference in computation depending on the QEP chosen.

As described in Section 2.1.1, joins are presented by means of a query graph, which has the query relations as nodes and the joins between relations as undirected edges which indicate the data flow from bottom to top. Generally, there are three kinds of joins: (i) tree joins, whose query graph is a tree and each relation participates only in one join; (ii) star joins, in which one relation participates in all the joins, and each of the other relations participates in one join; (iii) string joins, in which two relations participate in one join and each of the rest participates in two joins. To simplify the problem, researchers mostly study tree joins since the other two kinds of joins can be treated as extensions of tree joins.

The most widely used two kinds of strategies to traverse join trees depend on whether the trees are bushy trees or left-deep trees. If all internal nodes of a join tree have at least one leaf as a child, then the tree is called linear. Otherwise, it is called bushy. Relations in a join operation are always distinguished as outer (left) relation and inner

(right) relation. A left-deep tree is a linear join processing tree whose inner relations of all joins are base relations (see Figure 2.3). Most commercial database systems adopt the left-deep tree as their join search strategy to simplify the selection algorithms since bushy trees have larger generated search spaces and are harder for selection algorithms. However, confining the search space to left-deep strategies could harm plan quality, and at the same time not saving tremendous optimization time. Ioannidis concluded in [Ioa91] that the search space of both deep and bushy trees was easier to optimize than the space of left-deep trees alone.

Figure 2.3 (a) Left-deep Tree; (b) Bushy Tree

We shall now take a brief look at some recent research results on join order problems.

Vance and Maier studied join-order optimization with Cartesian products in a non-traditional way in [VaM96]. They presented new techniques to optimize searching bushy join trees for products of up to 15 relations. Join-order enumeration was fully separated from predicate analysis by the new techniques, so that the join orders were

¹ All of these intermediate cardinalities constitute worst-case scenarios.

generated extremely rapidly. The exorbitantly expensive plans were also rejected during the optimization processing.

In [WaP96], in contrast to transformation-based algorithms, Waas and Pellenkoft proposed a new algorithm, which was a probabilistic bottom-up join-ordering technique, to solve join-ordering problem. The experiment results showed that the NP-hard problem could be approximately solved in significantly less running time.

A formal algebraic model for rewriting magic sets² was proposed in [SHP96]. As well, it described an implementation of a cost-based optimization for rewriting magic sets in IBM's DB2 C/S V2 database system. Indeed, the technique involved optimizing join operations by rewriting magic sets. The advantage of this scheme is that by working bottom-up, we can take advantage of efficient methods for executing joins which are excessively large [BMU86]. As discussed above, since joins are associative and commutative, there a large space of $((2(N-1))!/(N-1)!)$ possible join orders. The results presented in this paper showed that the performance with using a limited search space was very suitable.

A new partition-based algorithm for set containment joins was proposed in [RPN00].

The Partitioning Set Join Algorithm (PSJ) used a replicating multi-level partitioning scheme based on a combination of set elements and signatures. PSJ was completely implemented in the Paradise object-relational database system. The results

² A "magic sets" method is a general algorithm for rewriting logical rules so that they may be implemented in a bottom-up manner in a way that the number of irrelevant facts that are generated is minimized.

showed that PSJ outperformed previously proposed set join algorithms over a wide range of data sets.

2.3.4 Join Methods

Join methods are techniques and methods used to implement joins. A cost-based query optimizer estimates the cost of each available join method and chooses one with the least cost. In general, join methods include:

- Nested-loops join
- Sort-merge join
- Hash joins

There are some other join methods developed in recent years. Mostly, their goals are to improve the join performance and reduce resources cost. Special hardware has even been designed for joins to achieve these goals for large database applications. However, the most widely-used join methods are the ones mentioned above, especially in commercial database systems.

2.3.4.1 Nested-loops Join

Nested-loops join is the simplest join method. One of the relations being joined is designated as the inner relation, and the other as the outer relation. For each tuple of the outer relation, all tuples of the inner relation are read and compared with the tuple from the outer relation. Whenever the join condition is satisfied, the two tuples are concatenated and placed in the output. In practice, a nested-loops join is implemented as

a nested-block join; that is, tuples are retrieved in units of blocks rather than in individual tuple. This reduces I/O activities for most cases.

Since each tuple of the inner relation is compared with every tuple of the outer relation, the simplest implementation of this algorithm requires $O(n \times m)$ time for the execution of the join, assuming that the two relations have n and m tuples respectively. This makes this algorithm unsuitable for joining large relations unless the join selectivity factor is high, where the join selectivity factor is the ratio of the number of tuples in the join result to the total number of tuples in the Cartesian product. However, the nested-loops join technique is very popular just because it is simple, and easy to implement. It is most efficient in cases in which the relation with the lower cardinality is chosen as the outer relation.

2.3.4.2 Sort-Merge Join

The sort-merge join is executed in two phases. First, both relations are sorted on the join attributes. Then, both relations are scanned in the order of the join attributes, and tuples satisfying the join condition are merged to form the result relation.

If the relations are presorted, this algorithm has a major advantage over the brute force approach of the nested-loops method. The advantage is that there totally are two full table scans used in this method. Further, if the join selectivity is low, the number of tuples compared is considerably lower than in the case of the nested-loops join.

The execution time of this method depends on the sorting and merging algorithms used. In general, the overall execution time is more dependent on the sorting time, which is usually $O(n \lg n)$ for each relation, where n is the cardinality of the relation. Blasgen and

Eswaran have shown in [BaE77] that this algorithm is most efficient for processing on a uniprocessor system.

2.3.4.3 Hash Join

Learning from the success of the sort-merge join method, the hash join method was developed, which also reduces the number of comparisons between the tuples. The tuples from the first relation that may join with a given tuple from the second relation are isolated. Then, tuples from the second relation are compared with a limited set of tuples from the first relation. A large variety of hash join methods have been proposed and/or implemented, such as the simple hash join method and hash-partitioned joins [MiE92]. We only discuss the simple hash join in this Thesis.

In the simple hash join, the join attribute(s) values in the first relation are hashed using a hash function. These hashed values point to a hash table in which each entry may contain entire tuples or only tuple-ids. Depending on the efficiency of the hash function, one or more entries may hash to the same bucket. Then for each tuple of the second relation participating in the join, the join attribute values are hashed using the same hash function. If the values hash to a nonempty bucket of the previous hash table, the tuple(s) in the hash table bucket are compared with the tuple of the second relation. The tuples are concatenated if the join condition is satisfied.

The hash join method is most efficient when the distinct values of the join attributes are the fewest. Its performance is not good enough in cases of uniform data distribution. Non-equi joins are difficult to implement since they require the hash function to maintain the correct order of the tuples.

2.4 Statistics and Cost Models

Statistics plays an important role in both query optimization and physical database design. It quantitatively summarizes the characteristics of a database. Consider the statistics of a relation, for example. The basic features of a relation which are regarded as pertinent statistical information are: the number of tuples in the relation, the number of pages used to physically store the data, the number of attributes in the relation, and so on. Since a relation is an object in a DBMS, these kinds of statistics are also called object-level statistics. Another kind of statistics is called system statistics, and it describes the performance characteristics of the hardware platform, such as the CPU costs and I/O costs related to every operation on a specified hardware platform. These are the most fundamental factors taken into account by cost models [MCS88].

In query optimization, it is the statistics or the statistical profiles rather than the actual objects that are utilized and manipulated. Consequently, the accuracy of the statistical profiles directly affects the cost estimation and the ultimate execution plan selection. There are numerous methods to collect statistical information, which will be discussed in the following sections. However, before we move on, we would like to emphasize the assumptions usually made about the database profiles [Chr84]. These assumptions are

- (i) **Uniformity of attribute values:** This assumption implies that there are an equal number of tuples with each value.

- (ii) **Independence of attribute values:** This assumption implies that the values of two attributes (A and B) are independent if the conditional probability of an A value given a B value is equal to the probability of obtaining the A value.
- (iii) **Uniformity of queries:** Here we assume that queries reference all attribute values with same probability.
- (iv) **Constant number of tuples per page:** This assumption means that each page contains the average number of tuples. In other words, the probability of referencing any page is $1/P$, where P is the number of pages.
- (v) **Random placement of tuples among pages:** By this we assume that the placement of tuples among pages does not affect their probability of reference. Thus, the probability of referencing any tuple is $1/N$, where N is the number of tuples.

Assumptions 1 and 2 affect the estimation of execution plan result sizes. Assumption 3 affects the size estimation of queries that reference a parameter, and various physical database design problems. Assumptions 4 and 5 affects the cost estimation for cost models.

2.4.1 Statistical Methods

We focus on object-level statistics in this section, since system statistics are more related to hardware and hardware performance, and the techniques for estimating system-related characteristics are quite different from those that we study in this Thesis.

The goal of maintaining statistical information in a DBMS is to obtain the knowledge of distribution of attributes in each relation. It enables the ability to calculate

which values of an attribute are most likely to occur, how many values could most likely occur in specified ranges, summary measures such as mean and standard deviation, and so on.

Many different techniques and methods have been developed for collecting statistical information. The methods useful for estimating the distribution shape of the attribute of relation can be divided into two basic types according to the way by which they estimate the shape, namely parametric methods and nonparametric methods. Random sampling techniques have also been applied to estimate distinct values of attributes and query result sizes, which are some pieces of major statistical information. We shall discuss these methods in the following sections.

2.4.1.1 Parametric Methods

Parametric methods estimate the distribution by assuming a parametric form and determining a few parameters. Thus the shape of the distribution is completely known before setting the parameters. That means that there is already an assumption made on the distribution of attributes of the relations. Commonly used parametric distribution models are the uniform, normal, Pearson family, and Zipf.

The uniform distribution uses a conservative, minimax assumption. It assumes equal probability over the range of possible values in the case of continuous values of attribute(s), or equal probability for all distinct values in the case of discrete values of attribute(s). It is the simplest and can be applied to all types of values of attributes.

There are two parameters to be estimated in the case of normal distribution: the mean and the standard deviation. This model is a symmetric, unimodal, “bell-shaped” distribution for continuous values of attribute(s).

The Pearson family distribution was proposed by Karl Pearson. It provides a method to include a wide range of choices of shapes while using a few parameters. Zipf’s law for continuous values of attribute(s) provides a model for highly skewed values of attribute(s). Only one parameter is to be determined - the exponent of a critical parameter - the abscissa.

2.4.1.2 Nonparametric Methods

Nonparametric methods assume little or no knowledge about the form of the distribution. This makes the estimation more difficult than the estimation with parametric methods. However, in most cases, this renders the estimation more accurate than with parametric methods. The histogram is the oldest and most commonly used nonparametric method, and it will be discussed in great detail in Chapter 3.

2.4.1.3 Other Estimation Methods

In this section, we present some recent research results of other estimation methods in solving well-known database problems.

(i) Methods for estimating number of distinct values

As important as the frequency distribution, the number of distinct values for a given attribute in a relation is another statistic which is helpful for the query optimizer for it to make cost estimations. As discussed above, probabilistic methods are useful for

estimating the distribution of attributes in a relation. Additionally, it is also useful for estimating the number of distinct attribute values in the result of project operation over a given subset of attributes.

In [HNS95], several new sampling-based estimators of the number of distinct values of an attribute in a relation were proposed. Haas and his co-authors compared these new estimators to those from the database and statistical literature empirically. Experiments of the study also indicated that a new “hybrid” estimator yielded the highest precision on average for a given sampling fraction by explicitly taking into account the level of the data skewed.

Charikar *et al* established a powerful result in [CCM00] stating that no previous random sampling algorithms could guarantee small errors across all input distributions, unless it examined a large fraction of the input data. Additionally, a new estimator was proposed to provide optimal estimation on distinct values for one attribute. The new estimator guaranteed its error without examining a large fraction of the input data. However, the drawback of this estimator is that its performance would be not optimal in the worst-case.

(ii) Methods of estimating query result sizes

As we know, sampling-based methods are mostly used to estimate result sizes by collecting and processing random samples of data from the database. Compared to parametric methods, sampling-based methods require no assumptions about the distribution of relations. Compared to nonparametric methods, they do not require storing and maintaining detailed statistics about both objects and system-related parameters.

However, there has been very little study on such methods because people think the practicality of this method is not good enough for query optimization. Here are some major reasons for such a position: (i) Many samples must be taken to attain reasonable accuracy; (ii) There are lots of I/O overhead costs because the physical I/O must be performed during sampling; (iii) The I/O overhead could be huge in cases when examining relations with a large number of tuples.

However, the results in [LiN90] showed that some sampling algorithms may have a better performance than anticipated. Lipton and Naughton studied the practicality of applying such schemes to selects and joins. They proposed an adaptive, random sampling algorithm for general query size estimation. This technique expresses the stopping condition for sampling in terms of the sum of samples, and the number of samples taken. They claimed that a good performance could be obtained even for highly skewed data distributions. However, they presented no results which showed that the algorithm was exceptionally accurate in such cases.

A similar study of sequential sampling for query size estimation was presented in [HaS92]. A sequential procedure, in which the sampling was stopped after a random number of steps based on a stopping criterion, was proposed. The stopping criterion was determined based on the observations obtained from the random samples. The performance of the procedure was asymptotic, which was different from other sequential algorithms.

More recently, new techniques have been developed to improve the accuracy of the optimizer statistics for some particular scenarios. In these schemes, special criteria, such as the correlation and the properties of transient data are processed, and even

accurate statistics are not always sufficient for optimal query execution. Consequently, additional statistics are gathered dynamically during query optimization. Though some overhead is added to the query optimization, this technique is very efficient since the sampling occurs in the same transaction as the query even if the data is not committed. Some commercial database systems, such as Oracle, have adopted this technique to its statistics sampling procedures.

(iii) Sampling for Aggregation Queries

The study of aggregate queries has been a popular research topic since the data warehouse applications have grown rapidly over the years. Such applications are usually made for decision support, which requires lots of data mining for analyzing large databases. The common features of aggregate queries are that they are composed of COUNT, SUM, AVG and other computational functions, and with GROUP BY clauses. Thus, aggregate queries are expensive and resource intensive. One of goals aimed at by recent research is to make aggregate queries less expensive and more efficient. For example, Chaudhuri *et al* [CDD01] observed the problem that uniform sampling performed poorly when the distribution of the aggregated attribute was skewed. In [CDD01], they introduced a new technique of combining outlier-indexing with weighted sampling to respond to aggregate queries with significantly reduced approximation error compared to both the uniform sampling and weighted sampling methods individually.

2.4.2 Univariate and Multivariate Methods

Methods for estimating the distribution of a relation can also be classified into two categories according to the number of attributes measured per relation. Univariate methods examine only one attribute for the observed relation; multivariate methods examine more than one attributes. All the estimation methods we discussed above can also be used to estimate either univariate or multivariate distributions. Multivariate distributions are more difficult to estimate than univariate distributions because of the increasingly complex manner in which the attributes may interact as their number grows. If the observed attributes are independent, then the multivariate distribution reduces to the product of the individual, or univariate, distributions of each attribute. In practice, DBMSs consider the distribution of a single attribute only. This corresponds to what is known as the *attribute value independence assumption* (see Section 4.4).

2.4.3 Cost Models

Cost models are basically cost formulae that estimate the number of secondary storage accesses, the CPU and memory cost, and in the case of distributed databases, the costs and delays on networking communication, which are associated with each access method and database structure. Two major factors determine whether a cost model is good for the query optimizer. First, detailed information incorporated into the cost model about costs of all access methods and database structures, or the cost formulae, determine whether the optimizer could generate an accurate cost for each type of operation. Second, since cost formulae depend on the estimated size of the operators, statistical estimation is then critical to the accuracy of results generated by cost models. The more accurate statistics

that cost models have, lead to more effective and accurate results obtained by such methods.

In [SAC79], Selinger *et al* proposed the cost model for the classic System R, including both I/O and CPU costs with weighted factors for conversion, and with statistics on tables (nrows, npages, page density) and indices (unique keys, npages) which are kept in the system catalogue, and updated infrequently. The query optimizer estimates the selectivity of the operations and then the costs of the access methods.

The effectiveness of a cost model is important to the overall performance of query optimization. Many researchers have proposed new methods and techniques to improve the performance of cost models. For example, in [ALU01], a cost model was proposed to count the number of subgoals in a physical plan, and presented a search space that was guaranteed to include an optimal rewriting. The paper also presented an algorithm to find rewritings with the minimum number of subgoals by utilizing views. The result showed that by careful variable renaming, the proposed cost model had better performance than the standard “supplementary relation” approach.

2.5 Dynamic Runtime Optimization

As we know, the workload on every database may vary differently, sometimes greatly, from hour to hour, from daytime to evening, from weekday workloads to weekend workloads, and from normal workloads to peak-time workloads. No static optimizer statistic and fixed optimizer model can cover all of the dynamic aspects of these ever-

changing systems. Dynamic adjustments to the execution strategies are mandatory for achieving good performance.

The key consideration for dynamic optimization is the appropriate management of the hardware resources, such as CPU and memory, to ensure that the hardware resources are maximally utilized by each query. Other aspects usually involve optimizing a single SQL statement, and dynamic optimizing each SQL statement in the context of all the other SQL statements that are being executed simultaneously.

2.5.1 Dynamic Hardware Resources Optimization

On a system with multiple processors, parallelism is usually a very good way to improve the response time of a query. However, setting the degree of parallelism is critical to the performance of queries. As mentioned above, the available hardware resources such as the CPU and memory change with time. When the overload of the system is high, (which means that there are lots of resource contentions), parallelizing a query or using too high a degree of parallelism can be counterproductive. When the overload of the system is low, queries should have high degree of parallelism to utilize the maximum hardware resources. Therefore, dynamically adjusting the degree of parallelism for queries would increase the performance of queries and utilize hardware resources more efficiently. Commercial DBMSs like Oracle have already adopted such technologies.

Memory is important to some particular operations like sorts and hash joins, because they are mostly processed in memory. In the best cases, sorts and hash joins can occur entirely in memory so as to avoid the much slower I/O operations, and also to avoid the usage of excessive temporary disk space. Since memory is finite on all systems,

and since it is a competed-for resource among all the processes running on the system, allocating too much extra memory for a process means wasting valuable resources, and might decrease the overall performance of *all* the currently-running processes. Thus, allocating memory according to the application's need, and also relying on the workload of the system is critical. Many tools for dynamically allocating memory and CPU resources have been developed to assist applications (including database applications) to yield a superior performance.

Since query parallelism and hardware resource management are not the primary focus of this Thesis, we will not discuss them any further.

2.5.2 Single SQL Statement Optimization

Some dynamic optimization strategies for single SQL statement use techniques that suggest optimization decisions before the execution of an SQL statement, and prior to evaluating them at its execution. Graefe *et al* studied the query optimization of queries embedded in a program ([CoG94], [GrW89]). In most database systems, the embedded queries are optimized when the programs are compiled. The resulting query evaluation plan is then stored in an "access module" at compile time, and will be activated by procedure calls at run time. However, if there are variables in the predicates, the binding cannot be completed until at run time. Therefore, the stored query evaluation plan is suboptimal for most cases. This is an issue related to the incompatibility of costs at compile-time. They introduced new techniques for choosing optimal QEPs for such queries. An optimal QEP is selected efficiently at run time, from fragments prepared by the optimizer using a decision and linking procedure included in the access module.

Experimental results demonstrated better performance of the proposed approach over both static plans and run-time optimization strategies.

Other techniques take advantage of human intelligence and the experience of human “in the loop”, to avoid erroneous decisions made by the query optimizer on its own. They allow people to manually interfere with the choice of the query evaluation plans. By defining a set of new rules as extension to query language, the query optimizer chooses query evaluation plans according to the settings in the SQL statement. Andrei and Valduriez introduced the Sybase Adaptive Server Enterprise (ASE) Abstract Plan (AP) language in [AnV01]. The proposed language is a 2-way user-optimizer communication mechanism based on a physical level relational algebra. The same techniques have been implemented in other commercial DBMS such as Oracle.

As the Internet grows rapidly, web applications grow even faster. The traditional query optimizer assumes a fixed topology for each query, while for Web query application, and topology of the data to be accessed, are unknown *a priori*. This makes query evaluation and the way to retrieve data from the Web quite different from the traditional methods. Numerous researchers have begun to probe actively in this area. For example, in [JS00] Jim and Suciu proposed a new paradigm in which pairs of servers communicate by exchanging messages and models dynamically. However, since this area too is outside the scope of this Thesis, we terminate our discussion on these topics here.

2.5.3 Multiple SQL Statements Optimization

From what has been discussed about query optimization this far, we can see that it takes the cost-based query optimizer several steps to estimate the cost for each candidate QEP,

and to make a final decision to choose the one with the estimated “lowest” cost. Sometimes, this is a costly operation in itself. Since a normal DBMS supports multiple users to access database, there could be multiple SQL statements accessing the database in any given time period, or even simultaneously. Thus, the issue of taking advantage of multiple SQL statements in query optimization has also been developed.

In [MPK00], Manegold, Pellenkoff and Kersten proposed a new architecture which enhanced a current database query optimizer so as to reuse intermediate results, especially for data mining database applications. It presented an inter-application multi-query optimizer that re-used previously computed results, and eliminated redundant work.

Similarly, in [SLM01], Stillger *et al* introduced a learning optimizer on DB2, which had a feedback loop to allow the optimizer to learn from its past mistakes. The extended optimizer can adjust cardinality and other estimates by using the actual statistical information from the execution of previous queries that have similar predicates. Thus, the mistakes made for previous queries can be “rectified”, and the query optimizer benefits for the subsequent queries. The result of [SLM01] demonstrated that the cardinality estimations and change of plans had been improved by orders of magnitude in subsequent QEPs. More importantly, in terms of execution time, the overhead to the original optimizer was less than 5%.

2.6 Patent Survey

As a part of our background study for this Thesis, we carried out a survey of the patents which are related to histogram algorithms. Since the R-ACM and the T-ACM are patented technologies, we felt that such a comparison would be advantageous. There are numerous papers and patents related to the use of histogram methods in databases. Most of these techniques have already been surveyed in this Chapter. The patent officer examining the patent submission for the R-ACM and T-ACM requested a comparison against *a few* other patents. This comparison follows.

We have studied eight patents with the similar technologies ([AMS99], [ShZ90], [JNS02], [SAR99], [WRW99], [HAI90], [FrS00], and [CKA92]), of contemporary dates. One thing is obvious from these studies: The fundamental technology involving histograms has been used in numerous applications that vary from scientific computing to medical research, and even to applications involving our everyday life, such as in supermarkets. Apart from the academic publication survey (some of which also consists of patented results) given in this Chapter, the patented technologies directly related to the R-ACM and the T-ACM is few. We give below a brief overview of these patents, and also a comparison with the R-ACM and the T-ACM.

In [AMS99], the inventors presented a new method for estimating frequency moments in a database. Their aim was to assure that the estimates are optimized, and accurate. By way of notation, the frequency moment, F_0 , for a sequence, is the number of distinct elements in the sequence. The frequency moment, F_1 , is the length of the sequence. Similarly, F_2 is a moment that can be used for error estimation in the context of

estimating query result sizes and access plan costs. In this patent, the frequency moment F_k , of a sequence containing m_i elements of type i , was estimated by the formula:

$$\sum_{i=1}^n m_i^k, \text{ where } 1 < i < n \text{ and } k \geq 0.$$

The complexity of the proposed algorithm was:

$$n^{1-1/k} \log(n+m),$$

where n is the number of possible values, and m is the length of the database sequence.

One of advantages of this new algorithm in [AMS99] is that, till its patent date, this patent reported the only randomized algorithm for estimating F_k , where $k > 1$, which guarantees a high accuracy with high probability, and yet requires a small amount of memory.

Comparing the algorithm proposed in [AMS99] with the R-ACM and the T-ACM, it is clear that the former can be used for estimating F_0 , F_1 and F_2 where only the last of these is useful in query result size estimations. Both the R-ACM and the T-ACM achieve the same task by completely different principles, for example, by minimizing the variation around the mean, or by finding the best trapezoid for the same area. With respect to the complexity and memory requirement for the algorithm, the differences are more subtle. Theoretically, m is always larger or equal to n . However, in most real-life database applications, m is much larger than n . The complexity of the R-ACM is $O(n)$ if the tolerance value, τ , is selected appropriately. Hence, if m is thousands times larger than n , which in many scenarios is the case, then $\log(n+m)$ can not be better than (n) . Furthermore, the algorithm updates a counter whenever a new item is entered in the

database, rendering it very cumbersome. However, if the update occurs at run-time, and if the sample ratio is comparably large in order to obtain high estimation accuracy, the cost could be exorbitant.

A new method for estimating the time required to process a query using a selected index was introduced in [ShZ90]. This method was a sampling method maintaining the most frequently occurring values of an index key. Based on the collected statistics, an estimate of the time required to use the index as the access path was thus computed.

However, with regarding to the determinations of choosing the optimal QEP in a database query optimizer, the cost index used in [ShZ90] is quite different from the selectivity criterion. The query optimizer chooses an “optimal” QEP based on both of these criteria, and it is not easy to affirm which one is more crucial. Since the R-ACM and T-ACM generate more accurate estimates of the selectivity, and the new method introduced in [ShZ90] can provide accurate estimates based on the *required time*, it is clear that both the algorithms are quite distinct. We believe that the query optimizer could be enhanced by utilizing *both* of these algorithms in conjunction, which, indeed, represent two different perspectives for QEP determinations. We believe that such a complementary strategy will yield even better QEP selections.

Index scanning is an area to which the R-ACM and the T-ACM have not been previously applied. However, we believe that it is feasible for them to be applied for this purpose also, since the analogous histogram algorithms, namely the Equi-depth and Equi-width algorithm, have been already used for the same.

In [JNS02], the inventors introduced a method for estimating the string-occurrence probability in a database. The proposed method computed the probabilities of

occurrence of a number of maximal substrings, where each substring in the set of maximal substrings belongs to the string itself.

This is a common problem encountered in database applications, because string attributes are as common as numeric attributes. However, the methods currently being used in [JNS02], namely, pruned count-suffix trees (PST) and histograms, are not efficient and accurate in solving this problem. Although these methods have nothing in common with the R-ACM and T-ACM, we believe that the latter can be used in this field by essentially replacing them with the Equi-depth/Equi-width counterparts. But this is currently being investigated.

Equi-depth histograms were used for load-balancing strategies in the patent introduced in [SAR99]. Here, the goal was achieved by allocating an (approximately) equal quantity of workloads to each processor. Because of the unique characteristics of the Equi-depth histograms, where each bucket contains equal number of points, the Equi-depth algorithm is quite suitable for application described in this patent. It is clear that the R-ACM can be utilized in this application in a manner similar to the way the Equi-depth algorithm is used. However, the performance of the R-ACM may not necessarily be superior to that of the Equi-depth, simply because the estimation error is not as crucial in this case.

In [WRW99], the inventors introduced a new method to ascertain the identification code of an item to be purchased, in order to prevent fraud in a retail checkout environment. The method added the color features of the item to be purchased, into the conventional product identification features, namely the UPCs and the prices. A new identification method which involved this feature was shown to be helpful in

preventing fraud by comparing the color feature of the item to be purchased to that of the expected item. The color representation was stored in the form of a histogram, and the comparison was also made histogram-based.

As in the previous case, it is clear that the R-ACM and the T-ACM can be used for the same application as the one suggested in [WRW99]. This is because all the comparisons are made on each bin of the histogram, and clearly analogous comparisons can be made even if the ACMs were used as the underlying histogram.

The patent introduced in [HAI90] was a method for reading a radiation image whose information was created and corrected. Although the image pixel information is presented in terms of histograms, the latter is different from the histograms used in databases. Such histograms are used for thresholding and contour preprocessing, and are not primarily used for the purpose of estimation. Thus, we believe that the ACM technology would not be directly applicable to this application.

The patent [FrS00] relates to the use of histograms in medical research, and it provided methods for determining the presence of a number of primary targets in a cell by using a drug. In particular, the patent proposed a method involving statistical analysis by utilizing the histogram. The latter was used to analyze the quantity of the inflection drug concentrations of the various cellular constituents. Since the multimodality of a particular distribution of inflection concentrations was not obvious, other statistical methods, namely the Fisher Distance, the parameters of the Guassian Distribution, and Monte Carlo methods, were involved to determine this multimodality.

The patent [FrS00] has specified no database application. The histogram (or the distribution) is only a tool to present the final results obtained from experiments.

The patent described in [CKA92] is related to the research involving a human's circadian cycle. It provided a method (and a device) for accurately assessing and rapidly modifying the phase and amplitude of the endogenous circadian cycle.

Histograms played a role in [CKA92] by simply representing the results diagrammatically. Clearly, by using bins, or by approximating the true distribution in a manner analogous to that which is done in the ACMs, the latter can have potential applications in these domains too. But both of these avenues have to be further investigated.

2.7 Conclusions

In this Chapter, we have briefly introduced the problems involved with the task of query optimization, and presented an overview of the area, and in particular, on the query optimizer. Because the query optimizer is critical to the overall performance of a DBMS, and because it has a highly complicated structure, we have focused on the various details of each part of the query optimizer, namely, SQL transformation, access path selection, and the use of statistics and cost models. As an extension beyond traditional query optimizers, dynamic runtime optimization strategies have also been discussed in this Chapter. Additionally, we have considered various performance-related issues, and highlighted important research results and specific techniques that have been recently developed in this area.

We have also briefly submitted a survey of patents with similar technologies, as requested by the patent examiner. The two most relevant patents have been compared with the R-ACM and T-ACM in greater detail.

As mentioned above, the focus of this Thesis is on the accuracy of query result-size estimation obtained by utilizing different algorithms or techniques. The most direct factor, which has the most significant influence on this problem, is the accuracy of the statistics. However, the latter problem, indeed, involves almost all the factors discussed in this Chapter, and it is hard to effectively separate one factor from the others.

Chapter 3: HISTOGRAM METHODS³

3.1 Introduction

As discussed in Chapter 2, statistics helps the query optimizer evaluate cost of QEPs, and the histogram is one of the basic methods used for collecting statistics in a DBMS. In this Chapter, we first give a brief introduction about the background of histograms, and the basic histogram methods currently applied in DBMS. Then, we explain two new histograms: the R-ACM and T-ACM. In this Thesis, we focus on how to integrate these two new histograms into a real-life DBMS, in particular in the Oracle database system. We also present experiment results of their performance comparing these methods with the basic histogram methods currently used in commercial database systems.

Histogram methods have been chosen by many current commercial database systems for collecting statistical information to support the query optimizer so as to estimate sizes of query results or costs of queries and QEPs. This is because of their advantages, which are: (i) they produce small errors compared to the parametric methods, (ii) they are inexpensive to construct, use and maintain, and (iii) they can be used for

³ The R-ACM and T-ACM are patented technologies. For any commercial or non-commercial applications, please contact Dr. John Oommen at School of Computer Science, Carleton University, Ottawa, Canada K1S 5B6. E-mail: oommen@scs.carleton.ca.

many diverse estimation problems. The query optimizer needs accurate statistics to make the right decision when choosing a potential QEP, and at the same time it should not consume excessive resources and much time. Histograms, therefore, won over their other competitors.

3.1.1 Terminology

3.1.1.1 Frequency Distributions of Values

The frequency distribution of values is a set of pairs indicating the number of tuples for an attribute in a relation where the value appears. The frequency distribution can also be made on a combination of multiple attributes. Multi-attribute joint frequency distributions essentially capture the correlation between the values of the participating attributes. Such distributions are expensive to calculate because there are so many possible combinations for multiple attributes. We shall concentrate on the frequency distribution of single attributes only.

3.1.1.2 Histogram

A histogram of an attribute is an approximation of the frequency distribution of its values by exhaustively scanning or sampling the attribute values. The (attribute value, frequency) pairs of the distribution are partitioned into buckets; the frequency counts and the bucket boundaries are stored as a distribution table. The distribution table can be used to obtain upper and lower selectivity estimates. By default, on being requested the

frequency of each value in a bucket is approximated by the average of the frequencies of all values in the bucket, which is also known as uniform frequency assumption.

- (i) **Construction:** Histograms can be constructed in different ways. The first is called the *serial* histogram, in which the frequencies of the attribute values associated with each bucket are either all greater or all less than the frequencies of the attribute values associated with any other bucket. A subclass of serial histograms is the *end-biased* histograms. Some numbers of the two ends of the frequency distribution are explicitly and accurately maintained in separate buckets, and the remaining frequencies are all approximated together in a single bucket. The frequency of each intermediate value is approximated to be uniformly distributed, which is also known as the *continuous-value assumption*.
- (ii) **Maintenance:** To utilize “optimal” histograms, not only is it important to easily build histograms with good or optimal algorithms, but also it is equally important that the histograms can be easily maintained. The latter issue has two implications involving the required storage space, and the freshness (or validity) of the data which the histograms are built on. If the frequency distribution of an attribute is not up-to-date, it does not reflect the most current status of that attribute. The histograms built for it, thus, have no real meanings.

3.1.1.3 Selectivity

Selectivity is the ratio of the number of tuples in the result set to the number of tuples in the relation(s) accessed in a query. Since the query optimizer does not physically access the data resident in the database, selectivity is one of the most important factors to predicate the cost

estimation. For example, if there are 100 tuples in a relation, and there are 10 tuples which meet the conditions in the query's predicates, then the selectivity is: $10 / 100 = 0.1$. The higher the ratio is, the larger the number of tuples in the relation(s) that are assumed to be chosen for the query.

Selectivity is also a critical factor used to choose join orders, join methods and access methods for a QEP (see Section 2.3.2). It is generally assumed that the more accurate the selectivity that a histogram can produce, the more optimal will be the QEP that is generated.

3.1.2 Histograms on Estimation Problems

We now discuss the estimation problems that histograms can be used for. These problems are various and occur in different kinds of queries. We, therefore, discuss this issue in the content of different categories of queries.

First of all, consider an example relation similar to the one widely used by the Oracle database system, involving the so-called EMPLOYEE relation. Values of three attributes of this relation are shown in Table 3.1, and the frequency distribution of the attribute "DEPTNO" is shown in Table 3.2.

EMPNO	SAL	DEPTNO
7782	2450	10
7839	5000	10
7934	1300	20
7369	800	20
7876	1100	20
7902	3000	20
7788	3000	20
7566	2975	20
7499	1600	30
7698	2850	40
7654	1250	50
7900	950	60
7844	1500	90
7521	1250	100

Table 3.1 An Example EMPLOYEE Relation

DEPTNO	Frequency
10	2
20	6
30	1
40	1
50	1
60	1
70	0
80	0
90	1
100	1

Table 3.2 Frequency Distribution of “DEPTNO” for the EMPLOYEE Relation of Table 3.1

3.1.2.1 Result Sizes of Arbitrary Equality Join and Selection Queries

Assume now, that we have a histogram constructed on the attribute “DEPTNO” with 3 buckets for 3 different “DEPTNO” values, for example, for the sets {10,20,30}, {40,50,60}, and {70,80,90,100}. The example of selection query is:

```
SELECT EMPNO, SAL FROM EMP WHERE DEPTNO = 10;
```

So, the approximate selectivity for the predicate is: $(9 / 3) / 14 = 3 / 14 = 0.21$, while the actual selectivity for the predicate in this query is: $2 / 14 = 0.14$.

If a histogram method is applied, the closer the approximate estimation is, the less estimation error it will cause.

The same technique can be also applied to other kinds of queries, namely the frequency distributions for attribute values in results of equality join queries.

In [IoP95], trade-off techniques between the theoretical and practical utilization of histograms were discussed. One of the conclusions made in this paper was that the end-biased histograms could be much more efficiently constructed without sacrificing much with respect to the estimate errors. This conclusion should shed some light on our new histograms, especially the R-ACM.

Another approach distinct from the histogram methods, that is useful for estimation problems is the one presented in [GTK01]. In [GTK01], Probabilistic Relational Models, which extended graphical statistical models such as Bayesian Networks to relational domains, were used for selectivity estimation. This new approach provided good results on select and foreign-key join operations.

3.1.2.2 Result Sizes of Range Selection Queries

Assume that we have the same histogram as in Section 3.1.2.1. An example range selection query is like:

```
SELECT EMPNO, SAL FROM EMP WHERE DEPTNO > 10;
```

So, the approximate selectivity is: $1 - 0.21 = 0.79$, while the true selectivity for the predicate in this query is: $(14 - 2) / 14 = 0.86$.

In [KMS00], a polynomial-time, dynamic programming algorithm was introduced to compute optimal histograms for hierarchical range queries. The reason why the new algorithm was proposed, was because traditional histograms, which are good for equality queries, are sub-optimal for hierarchical range queries. Furthermore, hierarchies are an inherent part of data warehouses, since any hierarchy on a dimension attribute defines a set of hierarchical ranges used for formulating data warehouse queries. The algorithm

proposed in [KMS00] was the first one that provably computed good histograms for non-equality queries.

In [GKM00], different statistical algorithms were proposed to optimize approximations for range queries. In this paper, two sets of results for range aggregate queries were presented. First of all, it presented algorithms which took polynomial time for constructing histograms that were provable optimal, and secondly, the paper presented fast algorithms for picking wavelet statistics. Since neither of these is directly related to the major topic of this thesis, we shall not discuss these aspects in any greater detail.

3.1.2.3 Set-Valued Queries

For the fast growing database applications, especially the Internet-related and data warehouse applications, answering queries approximately and quickly, has become really important. Such techniques can reduce query response time when the precise answer is not necessary, or when early feedback is helpful. This is a new research area since most of previous work done in this area used sampling-based techniques and handled aggregate queries ignoring queries that returned relations as answers. In [IoP99], the scope of approximate query answering to general queries was extended to those that returned set values. A numerical measure to quantify the quality of an approximate answer to set-valued queries was developed, and histogram-based techniques were also proposed for providing approximate answers to general, as well as, aggregate queries.

3.1.3 Histogram Weaknesses

Although histogram-based methods are favored by many DBMSs, they have some inherent weaknesses as illustrated below:

- *Uniform frequency assumption:* Similar to the uniform distribution assumption in database, this assumption will cause large errors if most of the attribute values occur with vastly different frequencies in a bucket. This is typical in most database systems.
- *Continuous values assumption:* This assumption permits only the storage of the boundary attribute values, and assumes that all possible intermediate values are present, although not stored. This assumption will also cause large errors if the frequency distribution of the attribute is highly skewed and not continuous.

In the following sections, we shall compare different histogram methods, and explain how they implement their approaches to minimize the effects of these weaknesses.

3.2 Basic Histogram Methods

There are various histogram methods. By making a uniform distribution assumption over an attribute, a histogram, called the trivial histogram, can be constructed with a single bucket generating the same approximate frequency for all the attribute values. Clearly, such histograms are of no help in accurate estimations. On the contrary, histograms can yield more accurate selectivity estimates if every value of the attribute could have a correspondent bucket. However, this is impractical for most cases, because it is expensive

to store all the boundaries and frequency counts for each value when an attribute has large number of distinct values ([PIH96]).

To explain the various histogram methods, we use the relation shown in Table 3.1 and the frequency distribution (Table 3.2) as an example in the next sections.

3.2.1 The Equi-width Method

The use of the Equi-Width histogram in databases was predominantly advocated by Piatetsky-Shapiro and Connell ([PsC84]). As its name implies, the Equi-width histogram has, as approximately as possible, the same width for each bucket, while the frequencies or the heights are variable. The frequency in each bucket is calculated by taking the *uniform frequency assumption*. For example, we construct Equi-width histogram for the example of Table 3.1 based on attribute “DEPTNO” with 5 buckets (see Figure 3.1). Wherever there are two sets of line representations shown in one figure, the bold line represents the actual data distribution, and the dotted line represents the corresponding histogram. This same rule will apply to all the figures shown in this Chapter.

Figure 3.1 The Equi-width Histogram for the Attribute DEPTNO of Relation EMPLOYEE in Table 3.1

Figure 3.2 The Equi-depth Histogram for the Attribute DEPTNO of Relation EMPLOYEE in Table 3.1

The drawback of the Equi-width method is obvious since it utilizes the *uniform frequency assumption*. It will not yield good performance if the attribute values are skewed, and if the bucket boundaries are not determined appropriately. Moreover, the statistics literature offers little help in solving the problem without some knowledge of the distribution [MCS88]. The latter paper also shows that the Equi-width method is asymptotically inferior to other methods based on various suitable error measures.

3.2.2 The Equi-depth Method

The Equi-depth histogram is the most widely-used one in the commercial DBMSs. Basically, this technique consists of dividing the given attribute values into buckets such that each of them has, *as approximately as possible*, the same number of frequencies. The frequency of a required attribute value is approximated by dividing the population in the bucket by the number of attribute values placed in that bucket. Also, its method of creating histograms is considered to have a superior error control over the Equi-width method. This is achieved by varying the width, and thus the Equi-depth histogram has the same height or frequency for each bucket. Figure 3.2 shows the Equi-depth histogram constructed on attribute “DEPTNO” with 5 buckets for the relation EMPLOYEE of Table 3.1. In Equi-depth histograms, bucket overlapping is permitted, which means some values of the attribute could occur⁴ in more than one bucket. For example, in Figure 3.2, value 20 is contained in the first, second and the third buckets.

⁴ Certain DBMSs, like Ingres, do not permit this. They build an *approximate* Equi-depth histogram, in which each attribute value is found in only one bucket.

Since all buckets have the same height, the error rate can be easily controlled by increasing the number of buckets. Similar to the Equi-width method, choosing appropriate bucket boundaries is a method by which we can actually reduce the error rate. However, the analysis based on some distribution knowledge has established asymptotically optimal choices (up to order of n , where n is the number of tuples) for the number of buckets. Kooi in his Ph.D. Thesis [Koo80] argued that the worst case and average case errors in both the equality and range selection queries are usually smaller in the Equi-depth method than in the Equi-width method.

3.2.3 Variable Kernel Method

In this method, the width of each bucket is varied according to a criterion other than equal frequency. Figure 3.3 shows an example of histograms constructed with a variable kernel method. The advantage of this method is that it can more accurately reflect the real distribution of the attribute. However, it is difficult to analyze the error control and the number of buckets. Few results have been reported in this regard.

Figure 3.3 The Variable Kernel Histogram for the Attribute DEPTNO of Relation EMPLOYEE in Table 3.1

All above methods can also be used for multivariate analysis. Each multivariate technique shares the same general advantages and disadvantages as its univariate counterpart. The multivariate techniques would become very complicated because the observed attributes may not be independent of each other. As explained earlier, to simplify our discussions, we only focus our descriptions on univariate techniques. We note, however, that the results should hold for multivariate techniques too.

3.3 Self-tuning Histograms

For traditional histograms, the cost of building and maintaining the histogram is always dependent on the data size. This makes the cost very expensive for large tables. A novel histogram technique was proposed in [AbC99]. It built histograms not by examining the data but by using feedback information about the execution of the queries on the database. Therefore, the initial setup of the histograms was inexpensive. Moreover, the histograms were refined based on the selectivity estimation error during runtime. The proposed technique had good performance on multi-dimensional histograms. However, ST-histograms were less accurate than the MHIST-2 histograms in situations involving highly skewed data.

3.4 New Histogram Method: the R-ACM

In 1999, Dr. Oommen and Dr. Thiagarajah from Carleton University proposed two new histogram-based methods, the Rectangular Attribute Cardinality Map (R-ACM) and the Trapezoidal Attribute Cardinality Map (T-ACM), which improved the effectiveness of

histograms on estimating query result sizes. In Thiagarajah's Ph.D. Thesis [Thi99], the mathematical foundation of both the R-ACM and the T-ACM were provided, and formally proved. Although we will not discuss the theoretical effects of these in this Thesis, we shall focus on the *implementation* of the R-ACM and T-ACM in the next sections, and analyze their effectiveness on real-life query result size estimation.

3.4.1 Definition

The R-ACM is a modified form of the histogram. It achieves an excellent error control by disallowing large frequency differences within a single bucket, which is the essence of the R-ACM. The R-ACM has a variable bucket width, and a variable number of tuples associated with each bucket. Therefore, the bucket boundaries and the height for each bucket are to be determined. Based on a rule that aims at minimizing the estimation error, the sector widths can be generated over given frequency distributions of attribute values. Another highlight of the R-ACM histogram is that it is catalogue based, which means that it can be precomputed, and thus will not incur much I/O overhead during run time.

To make our explanation clearer, we present in Table 3.3 the notation that will be used throughout this Thesis.

Symbol	Explanation
τ	Allowable tolerance for an R-ACM.
s	Number of buckets in the R-ACM.
X	An attribute in relation R , or the value set for attribute X .
X_i	The i^{th} value in attribute value set X .

Table 3.3 Notation Associated with the R-ACM

The R-ACM can be used as either an univariate method or a multivariate method. To keep the consistency of this Thesis, we introduce the univariate usage only, while the results should hold for multivariate usages as well.

Definition 3.1 A univariate R-ACM: Let $V = \{v_i: 1 \leq i \leq |V|\}$, where $v_i < v_j$ when $i < j$, be the set of values of an attribute X in relation R . Let the value set V be subdivided into s number of bucket widths according to the range partitioning rule described below. Then the R-ACM of attribute X is an integer array in which the j^{th} index maps the number of tuples in the j^{th} value range of the set V for all j , $1 < j \leq s$.

Rule 3.1 Range Partitioning Rule: Given a desired tolerance value τ for the R-ACM, the sector widths, l_j , $1 \leq j \leq s$, of the R-ACM should be chosen such that for any attribute value X_i , its frequency x_i does not differ from the running mean of the frequency by more than the tolerance value τ , where the running mean is the mean of the frequency values examined so far in the current bucket.

For example, consider the frequency distribution shown in Table 3.2. Using a tolerance value $\tau = 2$, the values of the attribute “DEPTNO” will be partitioned into three buckets, $\{10\}$, $\{20\}$, $\{30,40,50,60,70,80,90,100\}$ with bucket widths of 1, 1 and 8 respectively (see Figure 3.4).

Figure 3.4 The R-ACM Histogram for the Attribute DEPTNO of the Relation EMPLOYEE in Table 3.1

3.4.2 Generate the R-ACM

The formal algorithm which uses *Rule 3.1* is given in Algorithm 3.1 below.

Algorithm 3.1 Generate_R-ACM

```

Input: tolerance  $\tau$ , frequency distribution of X as A[0..L-1]
Output: R-ACM
begin
    Initialize_ACM; /* set all entries in ACM to zero. */
    current_mean := A[1]; j := 0;
    ACM[j] := A[1];
    For i := 1 to L-1 do /* for every attribute value */
        if abs(A[i] - current_mean) <  $\tau$ 
            ACM[j] := ACM[j] + A[i];
            /*compute the running mean */
            current_mean := (current_mean * i + A[i]) / (i + 1);
        else begin
            lj := i - 1; /* set the bucket width */
            j ++;
            current_mean := A[i];
            ACM[j] := A[i];
        end;
    end;
end;
End Algorithm Generate_R-ACM;

```

From the given algorithm, we can see that the R-ACM histogram buckets are generated based on *Rule 3.1*. The inputs to the algorithm are the tolerance value, τ , and the actual frequency distribution of attribute X. The frequency distribution is assumed to be available in an integer array A, which has a total of L entries for each of the L distinct values of X. This corresponds to the statistical information which can be obtained from the data catalogues or data dictionaries in a DBMS. The tolerance value τ is the critical factor to control the errors. From *Rule 3.1*, we can see that τ controls the frequency differences within a bucket. Although we may want to have smaller tolerance value τ to achieve more accurate estimations, in practice, trade-offs should be made to reduce the construction and maintenance cost of the R-ACM. We shall discuss how we make such trade-offs for our experiments in the next Chapter. The output of the algorithm is the R-ACM for the given attribute.

The bucket boundaries and the frequency for each bucket are stored in the R-ACM structure. The R-ACM does not actually sample or scan the database to approximate the data distribution. Instead, it uses the statistical information as its input. The advantage of the algorithm is that it saves I/O costs for both storing the required information about the generated R-ACM, and for accessing the real data from the secondary storage. This would improve the run time performance of the algorithm. However, as in any static structure, one drawback is that the resulting structure depends on the “freshness” of the given information related to the frequency distribution. If the given information is stale, then the generated R-ACM is just as “stale”, since the information based on which it is built does not reflect the most current data distribution.

3.4.3 Query Result Size Estimation

Result 3.1 *For a univariate R-ACM, the maximum likelihood estimate of the number of tuples for a given value X_i of attribute X is given by,*

$$\hat{x}_{ML} = \frac{n}{l}$$

where n is the number of tuples in the bucket containing the value X_i , and l is the width of that bucket.

From the above result, it is clear that there is no major difference in the computation involved between the R-ACM and the basic histogram methods used in the selectivity estimation. In fact, however, the R-ACM improves the accuracy in estimation by its better error controls. It disallows the huge frequency differences within a bucket, so that the distribution within that bucket is much closer to its mean, and thus satisfies the *uniform assumption*. Observe that this is the easiest way to compute the selectivity, which requires the least cost for its computation and utilization.

The R-ACM has numerous other theoretical properties. They can be found in [Thi99], but omitted here in the intent of briefly.

3.5 New Histogram Method: the T-ACM

3.5.1 Definition

Similar to the R-ACM, the Trapezoidal Attribute Cardinality Map (T-ACM) is another non-parametric histogram-based estimation technique. The T-ACM generalizes the R-

ACM from a “step” representation to a “linear” representation. It has equal bucket widths and variable linear-sloped bucket heights.

The T-ACM is a modified form of the Equi-width histogram where each histogram partition is a trapezoid instead of a rectangle in the R-ACM. In fact, the T-ACM is obtained by replacing each of the rectangular sectors of the Equi-width histogram by a trapezoid. The beginning and the ending frequency values of each bucket is chosen so that the area of the resulting trapezoid will be equal to the area of the “rectangle” of the histogram it is replacing.

Definition 3.2 A univariate T-ACM: Let $V = \{v_i: 1 \leq i \leq |V|\}$, where $v_i < v_j$ when $i < j$, be the set of values of an attribute X in relation R . Let the value set V be subdivided into s equal-width buckets, with the width of each bucket being l . We approximate each Equi-width bucket by a trapezoid in which the j^{th} trapezoid is obtained by connecting the starting value, a_j , to the terminal value, b_j , where the quantities a_j and b_j satisfy:

- (a) The starting value a_1 is a user-defined parameter.
- (b) For all $j > 1$, the starting value of the j^{th} trapezoid, a_j , is the terminal value of the $(j-1)^{\text{th}}$ trapezoid, b_{j-1} .
- (c) The area of the j^{th} trapezoid exactly equals the area of the j^{th} Equi-width bucket from which the exact computation of the quantity, b_j , is possible.

Then the T-ACM of attribute X with initial attribute value X_1 and width l is the set $\{(a_j, b_j) \mid 1 \leq j \leq s\}$.

For example, consider the frequency distribution shown in Table 3.2, and the Equi-width histogram shown in Figure 3.1. The corresponding T-ACM is shown as

below. The red line represents the real data distribution, and the blue line represents the T-ACM histograms.

As we can see from the figure, each bucket of a T-ACM stores the *modeled* frequency values of the boundaries, which naturally leads to a fairly accurate value to the total number of frequencies in that bucket. Another important aspect is, of course, is that the shape of the T-ACM. Though the bucket number and widths are the same in these two methods, the shape of the T-ACM reflects the actual frequency distribution much better than that of the Equi-width. This is where the T-ACM has an improved performance.

Figure 3.5 The T-ACM Histogram for the Attribute DEPTNO of the Relation EMPLOYEE in Table 3.1

3.5.2 Generate the T-ACM

Algorithm 3.2 Generate_T-ACM

Input: No. of buckets – s, frequency distribution of attribute X as A[0..L-1]. Or, the input can also be an Equi-width histogram.

Output: T-ACM

begin

Initialize_ACM; /* set all entries in ACM to zero */

$ACM[1].a := \frac{\sum_{i=0}^{L-1} A[i]}{L}$; /* set a_1 to average frequency */

for j := 0 to s do /* for every bucket */

for i := 0 to l do /* for every attribute value */

ACM[j].n := ACM[j].n + A[j * l + i];

if (j > 1) then ACM[j].a := ACM[j-1].b;

ACM[j].b := 2 * ACM[j].n / l – ACM[j].a;

end for;

end

End Algorithm Generate_T-ACM;

The algorithm is fairly simple. The inputs are the number of buckets and the frequency distribution. Such information can be easily obtained from the statistics in the data dictionaries, since it is the same information to feed the Equi-width algorithm. The output of the T-ACM is generated by processing using the starting and end points being set for each bucket. Since the resultant histogram has a trapezoidal shape for each bucket, instead of a rectangle, we need to decide the start point and end point for each trapezoid.

- The starting point: for the first bucket, the starting frequency value, namely a_1 in the algorithm, is determined by the average frequency of given frequency distribution. For the remaining buckets, the starting point is the same as the end point of the previous bucket.

- The end point: is determined by the rule that each trapezoid has the same total number of frequencies as with the Equi-width methods. Recalling from the geometry of the trapezoidal bucket, we know that the total number of frequencies, n_j , for each bucket is equal to,

$$n_j = \left(\frac{a+b}{2} \right) * l,$$

where a , b are the start and end points of that bucket, and l is the number of distinct values in that bucket. Since the total number of frequencies of the bucket can be obtained from the frequency distribution, and since l is given as an input, and the start point is already determined, the end point is easily computed.

There are many other ways to determine the starting and ending points. The problem of obtaining an optimal starting point is still open and is currently being investigated. The scheme presented in *Algorithm 3.2* is only one of many possible choices, which is easily implemented.

3.5.3 Query Result Size Estimation

Result 3.2 *For a univariate T-ACM, the maximum likelihood estimate of the number of tuples for a given value X_α of attribute X in the k^{th} T-ACM bucket is given by,*

$$\hat{x}_{ML} = a_k + \frac{2(n_k - a_k l)}{l(l-1)} * z_\alpha$$

where n_k is the number of tuples in the k^{th} T-ACM bucket, a_k is the frequency of the first attribute value in the k^{th} bucket, l is the number of distinct attribute values (or width) of the T-ACM buckets, and X_α is the z_α^{th} value in the T-ACM bucket.

As we see, the selectivity estimation with the T-ACM is more complex than the estimation using the R-ACM. It is more expensive to construct and maintain the T-ACM as well. The T-ACM has numerous other analytic properties. They are found in [Thi99] and omitted here.

3.6 Histogram Comparisons

In this section, we compare the two new histogram methods, the R-ACM and T-ACM, with two traditional Equi-width and Equi-depth histogram methods, in terms of the following different aspects: (i) error controls, (ii) estimate errors, and (iii) required storage space.

3.6.1 Error Controls

One of the factors to determine if a histogram method is good is whether the method has a good error control. Frequency differences within a single bucket determine the effectiveness of the histogram. Different histogram methods have different error controls.

- The Equi-width method: Since each bucket has an equal width, the height is then the only determination for all the buckets. It is obvious that as the number of buckets increases, the accuracy of the estimation also increases. For uniform distributions, the different number of buckets will not necessarily affect the final estimation results. For skewed distributions, however, it is true that as the number of buckets increases, the frequency differences within a bucket decreases. However, without *a priori* knowledge of the frequency distribution, it is difficult to determine the number of

buckets, and even more difficult to determine a fixed *number* of buckets which is good for all the different kinds of distributions.

- The Equi-depth method: This method has an equal height and variable widths for each bucket. Therefore, it has more error controls than the Equi-width method. However, it does not completely solve the problem of having the frequency differences within a bucket. It could be possible that these are cases where the frequencies are widely different within a bucket by using the Equi-depth method. Similarly, determining the approximate height and width of each bucket for a particular distribution has to be investigated case by case. There is no definite solution to all cases.
- The R-ACM: As introduced by its definition, there is a tolerance value τ , which indicates the frequency difference within a bucket. This is how the R-ACM can be used to control the estimation errors. By defining the value of τ , the frequency difference is then controlled to be within τ . The smaller the value of τ , the smaller the frequency difference that a bucket has. This is how the R-ACM improves the traditional histogram methods.
- The T-ACM: Although the T-ACM is similar to the Equi-width method, which has the same width for each bucket, it has trapezoids for the shapes within the buckets, instead of rectangles. The starting-point and endpoint for each trapezoid, control the asymptotic distribution, and its resemblance to the real distribution. We discuss the T-ACM as it is generated by *Algorithm 3.2* in this Thesis. In [Thi99], Thiagarajah has proven that the T-ACM improves over the Equi-width method because the

trapezoidal rule for numerical integration provides a more accurate estimation of the area under a curve than the left-end or right-end histograms.

3.6.2 Estimate Errors

According to the optimal histogram model proposed by Ioannidis in [Ioa95], we take a look at the effectiveness on of different histograms discussed above. This model can be applied to result sizes of arbitrary equality join and selection queries, frequency distributions of attribute values in results of equality join queries, and result sizes of range selection queries. The optimal histogram with β buckets on an attribute is the one that minimizes:

$$\sum_{i=1}^{\beta} n_i V_i$$

where n_i is the number of attributes, placed in bucket b_i , and V_i is the variance of the frequencies of the attribute values placed in bucket b_i .

Once again, we take the example used in Section 3.1, and the results of the different methods. To make these methods comparable, we use the same number of buckets, which is 5. In the case of the R-ACM, we set the tolerance value, τ , to 1, so that we have a 5-bucket histogram as {10}, {20}, {30, 40, 50, 60}, {70, 80}, {90, 100} (see Figure 3.6). By applying the optimal model, and referencing the generated histograms (see Figure 3.1, 3.2, 3.4), we have the following results:

Figure 3.6 The R-ACM Histogram with 5 Buckets for the Attribute DEPTNO of Relation EMPLOYEE in Table 3.1

-The Equi-width method:

$$\sum_{i=1}^{\beta} n_i V_i = 2 * 4 = 8$$

- The Equi-depth method:

$$\sum_{i=1}^{\beta} n_i V_i = 1 * 2 + 1 * 2 + 1 * 4 = 8$$

- The R-ACM:

$$\sum_{i=1}^{\beta} n_i V_i = 0$$

- The T-ACM: The rationale for the T-ACM is that

$$\sum_{i=1}^{\beta} n_i V_i = 1 * 2 + 1 * 2 + 1 * 4 = 8$$

From this simple example, we can see that the R-ACM is the most effective histogram. As a matter of fact, this is no surprise, because the latter histogram perfectly

and accurately mimics the distribution. In this case, the Equi-width and Equi-depth methods have similar degrees of effectiveness, because none of them exactly mimic the real distribution. For the T-ACM, the trapezoidal model does not seem appropriate, because it does not consider the cases that the shape of a histogram is trapezoid. In spite of this, we can conclude that the effectiveness of the T-ACM is at least as good as that of the Equi-width method.

While these arguments relate to a particular example, they are verified theoretically in [Thi99], and experimentally in [Thi99] and in this thesis.

3.6.3 Required Storage Units

As discussed in Section 3.1, the required storage space for histogram methods is one of critical aspects for their maintenance. During run time, utilizing a histogram method has to access those parameters stored in the computer system. We assume that each parameter requires an equal amount of storage, and so all the parameters have the same I/O cost and the same CPU cost for processing. The storage unit here refers to a unit from either the hard drive or RAM, but not to secondary memory. In terms of run time I/O overhead, we will take into account the storage requirement for each method.

Assume that the number of buckets is s . For each method, the following parameters should be stored, namely, the number of buckets, the width of each bucket (or the number of distinct values placed in each bucket), the height of each bucket, the starting value, or the minimum value of the distribution for each bucket, and the end value, or the maximum value of the distribution or each bucket (optional). Specifically

for the T-ACM, an extra parameter is required, which is the total number of tuples placed in each bucket. Table 3.2 shows the minimum required storage units for each method.

	Equi-width	Equi-depth	R-ACM	T-ACM
Number of total buckets	0	1	0	1
Width of each bucket	1	s	s	1
Height of each bucket	s	1	s	0
Total number of tuples in each bucket	0	0	0	s
Starting value of the distribution	1	1	1	s^5
Minimum required storage units	$s + 2$	$3 + s$	$2s + 1$	$2s + 2$

Table 3.4 Required Storage Units for the Various Histogram Methods

The T-ACM requires most storage space among all these methods since the bucket shape in this method is trapezoidal and it needs extra parameter, namely the total number of tuples in each bucket, for the estimation (see Result 3.2). The R-ACM requires more storage units when the number of buckets is larger than 1. This does not mean, however, that the R-ACM requires a greater I/O cost than others. In practice, we can make trade-offs between the required storage and the tolerance value τ since the number of buckets s is determined by the tolerance value, τ . We shall explain how the trade-off is made for our experiments in the next Chapter.

3.7 Conclusions

In this Chapter, we have explained, in detail, the oldest statistical techniques used in databases, namely the histogram methods, which are widely used in the estimation process by commercial database systems. We have discussed different estimation

⁵ Special for T-ACM, s here represents “the number of all starting values for each bucket”.

problems to which histograms have been applied, and listed the weaknesses of this technique for solving these problems. Two relatively new histogram methods, namely, the R-ACM and the T-ACM, have been presented. Additionally, comparisons between these two new methods and the other two traditional histogram methods, namely the Equi-width and the Equi-depth methods, have also been included.

The R-ACM and T-ACM are the primary focus of this Thesis. We have alluded to the details of their superior theoretical performance in this Chapter, and shall present their practical performance by incorporating them into a commercial database system in the following chapters.

Chapter 4: USING THE R-ACM AND T-ACM FOR CHOOSING QEPs

In this Chapter, we discuss in particular how the R-ACM and T-ACM can be used in a real DBMS to improve query effectiveness. In Section 4.1, we explain in great detail about how the database query optimizer utilizes the statistical information, and chooses “the” optimal QEP among all available access paths. In Sections 4.2, 4.3 and 4.4, we present how the statistical information, which is collected and maintained for the R-ACM and the T-ACM, can be utilized by the query optimizer. This Chapter is the primary focus of this thesis. In next Chapter, we will present the methods and results of the experiments conducted in this thesis, which incorporate the R-ACM and the T-ACM into the ORACLE database system, and compare the query size estimates obtained from these two algorithms with those obtained from the traditional algorithm, the Equi-depth, which is currently used in ORACLE, as well as in most other commercial database systems.

4.1 Determinations for QEP

As discussed in Chapter 2, determining an optimal QEP is a procedure which constitutes the following processing operations:

- (i) The query optimizer must check all available access paths for the given query.
- (ii) The query optimizer must search in the space of QEPs and evaluate the costs for all candidate QEPs by using the statistical information and cost models provided by the DBMS.
- (iii) The query optimizer must choose the one with the least cost as the “optimal” QEP for the given query.

Certainly, this is one of major responsibilities of the query optimizer.

In this Section, we explain, in particular, how the potential QEPs for the ORACLE database system are determined. We do this since we have chosen the ORACLE database system as the platform on which the experiments are to be carried out. The architecture of the ORACLE cost-based query optimizer [Ora02] is presented in Figure 4.1. As the reader can observe, the query optimizer consists of three major components, namely, the query transformer, the estimator and plan generator. The query transformer standardizes and transforms the SQL statement, and then sends the result to the estimator (see Section 2.2). Based on the cost model and statistical information obtained from the data catalogues or dictionaries, the estimator generates three different types of measures, namely the selectivity, the cardinality and the cost. Finally, the plan generator selects a single QEP with the least cost among all the generated QEPs.

Figure 4.1 Query Optimizer Architecture

4.1.1 The Statistics

The statistics used by the query optimizer contains information about the data uniqueness feature and the data distribution, which is stored in the DBMS in terms of data dictionaries or catalogues, where, to make our explanation simpler, we shall use the term catalogue (or catalogues) only in this context. The statistical information forms the foundational basis using which the query optimizer evaluates candidate QEPs. As discussed in Section 2.4, in the whole process of query optimization, no real data access occurs. In other words, it is the statistical information stored in the catalogues that is accessed by the query optimizer, instead of the real data being physically accessed and processed. The query optimizer does this, clearly, in order to save the time required in the process of query optimization. This is because, as we know, there is usually a bound on the time allowed for the query optimization, and thus, a fast response from the query

optimizer is desirable. Clearly, this makes the quality of the statistics even more important for the query optimizer, so that it can utilize the more accurate estimates in the decision making process.

The statistics can be obtained by either scanning the database thoroughly, or sampling the database. The former method, certainly, takes more time if the database is very large, and may not be affordable if it is performed at run-time. However, the latter method can only provide estimates on the real data, and unfortunately, the estimates are not too accurate. Most commercial DBMSs provide both methods for collecting the statistics, and have solutions for making trade-offs between saving time for collecting statistics, and for obtaining accurate information.

Since the data in the DBMS is not unchanged, the statistics must be updated as the data changes. If this is not done, the statistics available is useless for the query optimizer, simply because it does not provide accurate information about the real physical data in the database. The effectiveness of the query optimizer relies on the accuracy of the statistics.

The statistics is used by the query optimizer, whether or not the histogram is utilized in the DBMS. Additionally, the maintenance of the catalogues is automatically performed by the DBMS. If no statistics are presented, the query optimizer, or more specifically, the *estimator*, rather uses internal defaults which are based on the predicate type.

4.1.2 Selectivity

From Figure 4.1, we notice that the selectivity is one of the outputs of the *estimator*. For the *estimator*, the statistics is the input, and the selectivity is the output. As discussed in the previous section, the *estimator* relies on the statistics. Therefore, the selectivity relies largely on the statistics too.

Selectivity is the fraction of rows to be selected from a row set. It is associated with a query predicate. A predicate acts as a filter that filters a certain number of rows which do not meet the specified condition. Therefore, the selectivity of a predicate indicates the number of rows which pass the *predicate* filter. A selectivity of 0 means that no rows will be selected from a row set, and a selectivity of 1 means that all the rows will be selected.

In a DBMS, there are many data structures used apart from the usual “simple” basic data structures, namely relations and views. Many of them are designed to improve the performance of the basic ones. For example, an index is supposed to accelerate the speed of data retrieval from a relation. As a matter of fact, it is commonly known that access by index is faster than that by scanning the relation thoroughly. This sounds both correct and reasonable since we can, typically, extract the meaning of the index by its name. However, it is not always the case, as can be seen by the following example.

Suppose we have a simple relation TAB with two attributes A and B.

Name	Type
A	NUMBER(6)
B	NUMBER(6)

Attribute A contains unique values from 1 to 10000. Attribute B contains 10 distinct values. While the value '5' occurs 9991 times, the values '1', '2', '3', '4', '9996', '9997', '9998', '9999', and '10000' occur only once. In addition to the basic TABLE SCAN access method over this relation, there is another access method, namely, an index created on attribute B.

Consider two test queries as below:

(1) SELECT * FROM tab WHERE b=5;

(2) SELECT * FROM tab WHERE b=3;

These are two similar queries. The only difference between them is the attribute value bound found in the predicate. What access method should the queries choose? Since there is an index already created on attribute B, which also exactly matches the condition of using indexes, the answer might be that the both queries should use the INDEX SCAN for faster access. While this sounds reasonable, it is not necessarily true in this case.

The reason for this is the following. The QEPs for these two queries are not the same as what is generally expected. With an INDEX present, the query optimizer prefers an INDEX RANGE SCAN for query (2), but a FULL TABLE SCAN for query (1). The execution plans which are generated by the ORACLE optimizer are shown as follows.

Query (1) requires a Full Table Scan, and the QEP is as follows.

Execution Plan		
0		SELECT STATEMENT Optimizer=ALL_ROWS (Cost=10 Card=9991 Bytes=99910)
1	0	TABLE ACCESS (FULL) OF 'TAB' (Cost=10 Card=9991 Bytes=99910)

Query (2), on the other hand, does an Index Range Scan, yielding the following QEP.

Execution Plan		
0	SELECT STATEMENT Optimizer=ALL_ROWS (Cost=6 Card=500 Bytes=5000)	
1	0	TABLE ACCESS (BY ROWID) OF 'TAB' (Cost=6 Card=500 Bytes=5000)
2	1	INDEX (RANGE SCAN) OF 'TAB_B' (NON-UNIQUE)

We should not be surprised by the result, because the reason why the query optimizer makes such these decisions is due to the big difference between the selectivities of these two queries. For Query (1), the selectivity is: $9991/10000 = 99.91\%$, while the selectivity for Query (2) is: $1/10000 = 0.01\%$. For such a large selectivity of Query (1), if the query optimizer chose the INDEX SCAN method, Query (1) would have spent time on scanning both the index and the relation, almost thoroughly. Clearly, for Query (1) the time spent on choosing the INDEX SCAN method is almost double the time spent by scanning only the relation itself. Hence, it is obvious that the decision that the query optimizer made is absolutely right. An analogous reasoning applies to Query (2), and if we follow the parallel reasoning, we will find that the optimizer made the accurate choice for Query (2) also.

From the above example, we see how selectivity affects the choice of the QEP. We would like to emphasize that the selectivity not only affects the choice of the access methods, but also for the join orders and the join methods used for the join operations. A conclusion, therefore, can be made that, the accuracy of the selectivity for a predicate, directly affects the choice of the QEP, and also the effectiveness of a DBMS.

4.1.3 Cost

Cost is another output of the *estimator* (See Figure 4.1). The cost represents the units of resources used for answering the query. Generally, three categories of resources are taken into account, namely disk I/O accesses, CPU usage, and memory usage. The cost calculated by the *estimator*, therefore, represents the estimated cost associated with a QEP for the query.

The cost model plays an important role in the estimation, since it specifies the cost associated with each access method and each join method. During the table scan, multiple blocks are read from the hard drive in a single I/O operation. Hence, the cost of a table scan depends on the number of blocks to be scanned and the number of multi-blocks read. The cost of an index scan depends on the levels in the B-tree, the number of index leaf blocks to be scanned, and the number of rows to be fetched using the *rowid* in the index keys. For the join methods, the join cost represents the combination of the individual access costs of the two relations being joined. The combinations are different, and depend on the algorithm of each join method. In other words, the cost of a join method is typically proportional to the asymptotic running time of the corresponding algorithm. For example, recall that in a nested loop join, for every row in the outer relation, the inner relation is accessed to find all the matching rows to be joined. Therefore, the cost is estimated as follows:

$$\text{cost} = \text{outer access cost} + (\text{inner access cost} * \text{outer cardinality}).$$

Even though the cost model could be the same, the physical parameters associated with the I/O operation, CPU and memory consumption are quite different, depending on

the underlying operating system and the hardware platform. The actual values of these parameters should be specifically examined, and configured on the specific DBMS installation.

4.2 Using the R-ACM and T-ACM in the Query Optimizer

As discussed in Chapter 3, the R-ACM and T-ACM yield superior performance on the estimations of the query result sizes. If both of these algorithms can be integrated into a real database query optimizer, the performance of the DBMS could thus be improved by utilizing the superior estimation algorithms, which is the primary focus of this Thesis. We want to show that the performance of the query optimizer is indeed superior by incorporation of the ACM technology.

In most DBMSs currently being used, the assumption (see Section 2.4), termed as the “*uniformity of attribute values*”, is implicitly adopted. Unfortunately, most real-world applications do not lend themselves to such an assumption. One of the critical effects of such an assumption, results in the inaccuracy of the resulting query result size estimates. This, in turn, decreases the overall performance of the DBMS.

As discussed above, to solve this problem, histograms are utilized in skew data cases. More specifically, the Equi-depth histograms are used by the query optimizers of most DBMSs. Since the query optimizer is the core of a DBMS, utilizing the R-ACM and T-ACM in the query optimizer involves at least the following aspects.

4.2.1 Collecting Statistics for the R-ACM and T-ACM

As discussed in Chapter 3, one of the advantages of the R-ACM and the T-ACM is that they are catalogue based. This saves CPU and I/O cost and time, at run-time, and also saves the time spent on query optimization. As a result of it, the query optimizer can also then respond to the query faster.

With respect to the utilization of the R-ACM and T-ACM, special catalogues are created in the DBMS. For the R-ACM, the related pieces of statistical information which are required for the corresponding histograms are:

- (i) **relation name:** Name of the relation, on which the histogram is constructed.
- (ii) **attribute name:** Name of the attribute, on which the histogram related statistical information is collected.
- (iii) **bucket number:** Serial number of the bucket of the constructed histogram.
- (iv) **endpoint value of bucket:** Endpoint value of each bucket. The attribute is divided into several divisions by a set of endpoint values separating the different buckets.
- (v) **tuple number:** Number of tuples placed in the corresponding bucket.

For the T-ACM, the related statistical information to be collected are:

- (i) **relation name:** Name of the relation, on which the histogram is constructed.
- (ii) **attribute name:** Name of the attribute, on which the histogram related statistical information is collected.
- (iii) **bucket number:** Serial number of the bucket of the constructed histogram.

- (iv) **starting point value of bucket:** The starting point value of each bucket. For the T-ACM, only the starting point of each bucket should be stored since the buckets are contiguously arranged, and the starting point of the next point is exactly the end point of the previous bucket.
- (v) **tuple number:** Number of tuples placed in the corresponding bucket.

These pieces of information are collected based on the underlying relation and attribute by invoking the corresponding algorithms of the R-ACM or T-ACM. To ensure the accuracy of the collected statistical information, a thorough scan over the relation is required for the first pass. Furthermore, once any changes are made on the underlying attribute, these catalogues are to be updated. Since the cost of collecting and constructing such statistical information is not trivial for most applications, especially for online applications or 24X7 mission critical applications, the frequency of updating the catalogues is very much case dependent. Certainly, there are always more than just one ways to solve this updating problem in most situations.

4.2.2 Compute Selectivity for the R-ACM and T-ACM

In Section 4.1.2, the criticalness of the selectivity in choosing a QEP was alluded to. To make the integration successful, computing the selectivity accurately for the R-ACM and T-ACM histograms is the key issue, since, as discussed above, the selectivity is clearly linked to the predicate. In most DBMSs, if no specific selectivity is assigned to the predicate, a default internal selectivity is used instead. Clearly, the default selectivity does not reflect the algorithm that the query optimizer uses in the estimates, and it thus cannot yield accurate estimations either. Consequently, additional selectivity

computations on primary operations are mandatory. The formulae presented in this Section are especially for the R-ACM and T-ACM histograms. They are used in our experiments, and might be different from those used in any real DBMS.

In this thesis, we investigate the performance of the R-ACM and T-ACM on some particular types of queries, namely, the equi-selections and equi-joins. Their utility on other types of queries is for more tedious, and will not be undertaken here.

4.2.2.1 Equi-Selections

We refer to “equi-selections” here as operations that are based on equality selections on any particular attribute. For example, an equi-selection predicate takes on the form:

SELECT ... FROM ... WHERE *deptno* = 10;

Since the equality operation is the basic fundamental operation, and the selectivity for all other relational operations, for instance greater than, less than, and not equal to, can be inferred from the equality operation, in this thesis, we discuss the selectivity for *equality* operations only.

We assume that the predicate is “att = value”, and that the R-ACM or T-ACM histogram is created for this attribute *att* and stored in the catalogues as discussed earlier. Then, for the R-ACM histogram, the formula used to compute the selectivity, σ , of the predicate is:

$$\sigma = \frac{\text{number of tuples in the bucket in which } value \text{ falls}}{\text{range of the bucket in which } value \text{ is placed} * \text{number of total tuples in the relation}}$$

For the T-ACM histograms, the formula is the analogous, and as described in Section 3.5.3 as:

$$\sigma = (a_k + \frac{2(n_k - a_k^l)}{l(l-1)} * z_\alpha) / \text{number of total tuples in the relation}$$

The computations are achieved by having the query optimizer consult the catalogues. If the catalogues contain the most up-to-date statistical information about the underlying attribute and relation, the outcome of the computation is the most accurate for the relation, and can be utilized appropriately.

4.2.2.2 Equi-Joins

As in the case of the equi-selections, we refer to “equi-join” as the operation of joining relations on equality predicates. Equi-joins are the most commonly used join operations. We will not discuss other join operations in this thesis.

The selectivity of a join is defined as the selectivity of the most selective join attribute adjusted by the proportion of non-null values in each join attribute. In other words, the join selectivity is related to the number of non-null values of the participating attributes in the join operation.

We assume that the equi-join predicate is “ $r1.a1 = r2.a2$ ”, which represents a join operation performed on attribute $a1$ of relation $r1$, and attribute $a2$ of relation $r2$. While NDV1 is the number of distinct values in $a1$, NV1 is the number of tuples with null values in $a1$. Similarly, NDV2 is the number of distinct values in $a2$, and NV2 is the number of tuples with null values in $a2$. Then, if no values are specified for the join predicate, the formula to compute the selectivity is obtained as below.

Let $|r1|$ be the cardinality of $r1$, and $|r2|$ be the cardinality of $r2$. Then,

$$\sigma = \frac{1}{\max(NDV1, NDV2)} * \frac{|r1| - NV1}{|r1|} * \frac{|r2| - NV2}{|r2|}$$

If there is value specified for the join predicate, the selectivity is computed in a slightly different way.

Let $\phi1$ be the number of tuples which satisfies the predicate in $r1$, and $\phi2$ be the number of tuples which satisfies the predicate in $r2$. Then, in this case,

$$\sigma = \phi1 * \phi2.$$

It is important to note that in most DBMSs, the information about null values and non-null values, such as DNVs and DVs in the formula, are stored in the general public statistical catalogues. They can be obtained just as easily as the special statistical catalogue described in Section 4.2.1.

Equi-selections and equi-joins are typically considered as the primary operations in database applications. For our purposes, computing the selectivity for these two operations is the basic computation required in the implementation of the R-ACM and T-ACM when integrated into the query optimizer.

4.3 The R-ACM in Sparse Data Cases

In addition to the known results for the R-ACM, we have also explored its applicability for cases when the frequency distribution of an attribute is sparse. Sparse data cases are the extreme cases of skew data, where the frequencies of most attribute values are zero, while only a few of them are non-zero. The attribute density function is thus spiky. The sparse data case represents a typical application. Unfortunately, this is an area in which

little research has been done - there seems to be no related published literature for such sparse data cases. It should be highlighted that the problem is not only of pathological importance, but it is often the predominant scenario in real life and databases. For example, it is normal that, when looking up in the yellow pages, names of people or companies we are looking for, form only a very small number of combinations out of all the possibilities of the characters and numbers.

It is generally assumed that the value distribution of an attribute is continuous even if the frequency distribution is highly skewed. The term “value distribution of attribution” (“value distribution” in short) is used as often as the term “frequency distribution of attribution” (“frequency distribution” in short), which is the distribution of the values of the attributes. This present study brings us to another perspective of the problem. Typically, we have always considered the frequency distributions to be the Y-axis of the distribution chart. One thing which is almost completely ignored is the X-axis of the chart, the value distributions, because the assumption made on the value distributions is that it is continuous. Unfortunately, the assumption has many implications especially for the sparse data cases. We list the implications below.

(i) Accuracy of selectivity

Consider the method used for estimating in a histogram. The estimated frequency for a given attribute value directly depends on the frequencies of attribute values which are placed within the same bucket, whether it is computed by using the averaged value, or by any other method. Hence, if either the Equi-width or the Equi-depth algorithm is applied to relations belongs to the sparse data case, the resultant histogram would not able to yield accurate selectivity estimates. The reason for this is

because, in the generated histogram, there must be certain (large) number of attribute values, whose frequencies are zero, which are placed in a single bucket and classed together with other attribute values whose frequencies are *not zero*. The large number of zero values will dramatically decrease the accuracy of the estimates when the difference between the non-zero values and the value ‘zero’ is very large.

(ii) Large number of buckets for generated histograms

If the R-ACM is applied for the sparse data cases with a proper choice of the tolerance value, τ , the resultant histogram should be able to *perfectly* mimic the frequency distribution. The only problem is that, the generated R-ACM histogram has the feature that it will possess a large number of buckets for any reasonable value for the tolerance value, τ . Since there is a large number of zeros in the frequency distribution, a new bucket is probably going to be generated for every non-zero frequency present. Though accurate, this is not a helpful result, since excessive memory would be required due to the large number of buckets involved, and furthermore, excessive CPU, memory and I/O cost would also be involved at run-time to access and process the generated histogram.

4.3.1 Averaged R-ACM

In order to solve the above problems, we propose the structure which we call the “Averaged R-ACM”, which is constructed out of the *averaged* values of a hypothetical *Equi-width* structure. Our goal, of course, is to maintain accurate estimates, while at the same time keeping the number of buckets to be small. The approach is constituted by two steps of processing.

(i) Value Distribution Compression

This procedure aims at compressing the scale of the value distribution. One of advantages of the Equi-width is that the resultant histogram definitely has smaller number of buckets, and each bucket has equal width. So, by applying an Equi-width philosophy initially, the resultant distribution has less attribute values on the X-axis. For example, let us assume that there are 1,000 attribute values in the original value distribution. After being compressed by the value-averaging operation with a bucket width of 4, the resultant value distribution has only 250 attribute values. Note that this is NOT a true Equi-width histogram, because these values are not used for estimation subsequently.

(ii) Averaged R-ACM Computation

The compressed value distribution is used as the input to the next phase, where, the module creating the R-ACM histogram is invoked, because of *its* advantage in generating accurate estimates. Since the compressed value distribution already has significantly less attribute values, the resultant histogram would, therefore, not have an excessively large number of buckets, and thus, accurate estimates can be obtained on the given frequency distribution.

The formal algorithm for the Averaged R-ACM is given below.

Algorithm 4.1 Generate_Averaged_R-ACM

Input: tolerance τ , bucket_width s , frequency distribution of X as $A[0..L-1]$
Output: Averaged R-ACM
begin
 /*compress the given distribution by using the Equi-width algorithm. */
 $B[0..((L-1)/s)] = \text{Equi-width}(A[0..(L-1)]);$
 $\text{ACM} = \text{Generate_R-ACM}(\tau, B[0..((L-1)/s)]);$ /*apply Algorithm 3.1*/
end;

End Algorithm Generate_Averaged_R-ACM;

This approach basically solves the problems discussed above, and also has the following advantages:

(i) Ease of Maintenance

Since a single additional step is added prior to the R-ACM processing, the basic requirement for such a process is to choose one with the minimal maintenance cost. Clearly, the Equi-width algorithm satisfies the requirement. This can easily be concluded by referring to the storage requirement analysis (See Table 3.3).

(ii) Ease of Mapping

One of potential problems of adding an additional step in the process is that the attribute values are not easily “traced” back after the compression operation. While other potential solutions are possible, not all of them lend themselves to be easily traceable back the original attribute values. In our approach, where we utilize Value Distribution Compression, we do not modify any attribute values in the first step because of the characteristics of the compression phase. Hence, the issue of tracking attribute values before the compression and after the compression, is easily achieved.

We shall now present experiment results which show the power of the modified scheme for the sparse data scenarios.

4.3.2 Experimental Results

The aim of the experiments conducted here is to compare the performance of two algorithms, namely, the averaged R-ACM and the Equi-width, in the sparse data cases. To render this possible, we built a sample arbitrary attribute, which had 1,000 attribute values ranging from 1 to 1000. With regard to its frequency distribution, μ , the ratio of

non-zero frequencies and zero frequencies was varied between 0.05 and 0.2. Furthermore, all the non-zero frequencies were randomly generated within the range 5 to 10.

In the Value Distribution Compression phase, we chose the bucket width to be 4. In other words, after this procedure, the value distribution range was compressed to 250, from the original 1000.

A particular consideration on the storage requirement was taken into account in the experiments. As discussed in Chapter 3, the minimum storage requirement for the Equi-width algorithm is $(s+2)$, while the requirement for the R-ACM is $(2s+1)$, where s is the number of total buckets. To make our results more meaningful and fair, we have carried out many sets of experiments with both these storage considerations.

In each set of experiments, 1,000 different data sets were randomly generated. For each individual data set, 1,000 attribute values were randomly chosen to test the estimates made by the averaged R-ACM and Equi-width algorithms.

The results are listed below for μ values 0.05 to 0.2.

Actual Freq. Value	$\tau = 2$				$\tau = 3$			
	Estimated Values		Estimated Errors		Estimated Values		Estimated Errors	
	Equi-width	Avg. R-ACM	Equi-width	Avg. R-ACM	Equi-width	Avg. R-ACM	Equi-width	Avg. R-ACM
0	1.46248	1.26245	1.462	1.262	1.3901	1.37611	1.390	1.376
5	1.46680	2.64997	0.722	0.557	1.4369	2.20547	0.717	0.605
6	1.47896	2.60832	0.760	0.599	1.4139	2.4256	0.766	0.628
7	1.45585	2.97761	0.794	0.608	1.3744	2.74592	0.804	0.641
8	1.46361	6.44158	0.819	0.293	1.3973	2.79775	0.825	0.672
9	1.42539	7.53137	0.843	0.249	1.3910	2.9277	0.845	0.684
10	1.44671	7.89697	0.855	0.272	1.3883	3.18718	0.861	0.694

Table 4.1 Results comparing the Equi-width with the Averaged R-ACM for sparse attributes when the number of buckets is the same, and $\mu=0.05$

Actual Freq. Value	$\tau = 2$				$\tau = 3$			
	Estimated Values		Estimated Errors		Estimated Values		Estimated Errors	
	Equi- width	Avg. R-ACM	Equi- width	Avg. R-ACM	Equi- width	Averaged R-ACM	Equi- width	Avg. R-ACM
0	2.87645	2.48951	2.876	2.490	2.9813	2.66883	2.981	2.669
5	2.89027	4.54437	0.487	0.280	2.9866	3.96841	0.472	0.314
6	2.85284	4.79433	0.545	0.318	2.9804	4.4368	0.529	0.336
7	2.87154	5.18153	0.597	0.321	2.9763	4.72082	0.582	0.366
8	2.93701	7.52727	0.639	0.211	2.9863	5.08637	0.629	0.391
9	2.81161	8.34245	0.689	0.195	2.9946	5.29096	0.668	0.424
10	2.89255	8.92224	0.711	0.193	2.9882	5.66285	0.701	0.449

Table 4.2 Results comparing the Equi-width with the Averaged R-ACM for sparse attributes when the number of buckets is the same, and $\mu=0.1$

Actual Freq. Value	$\tau = 2$				$\tau = 3$			
	Estimated Values		Estimated Errors		Estimated Values		Estimated Errors	
	Equi- width	Avg. R-ACM	Equi- width	Avg. R-ACM	Equi- width	Averaged R-ACM	Equi- width	Avg. R-ACM
0	4.5399	3.82672	4.540	3.827	4.4901	4.06502	4.490	4.065
5	4.43831	6.36100	0.334	0.324	4.5039	5.79786	0.351	0.235
6	4.59745	6.93684	0.360	0.252	4.5579	6.26263	0.353	0.195
7	4.48089	7.37138	0.409	0.205	4.5221	6.85384	0.398	0.201
8	4.49393	8.96263	0.456	0.193	4.4674	7.28013	0.459	0.190
9	4.61617	9.85045	0.496	0.174	4.3990	7.64985	0.516	0.215
10	4.59085	10.32030	0.544	0.151	4.5315	8.08032	0.549	0.230

Table 4.3 Results comparing the Equi-width with the Averaged R-ACM for sparse attributes when the number of buckets is the same, and $\mu=0.15$

Actual Freq. Value	$\tau = 2$				$\tau = 3$			
	Estimated Values		Estimated Errors		Estimated Values		Estimated Errors	
	Equi-width	Avg. R-ACM	Equi-width	Avg. R-ACM	Equi-width	Averaged R-ACM	Equi-width	Avg. R-ACM
0	5.94606	4.99594	5.946	4.995	6.0209	5.28454	6.021	5.285
5	5.94760	7.96788	0.361	0.595	6.0398	7.27495	0.373	0.457
6	6.02232	8.47467	0.292	0.419	6.0763	7.8059	0.279	0.313
7	5.89206	8.94806	0.285	0.294	5.9945	8.34211	0.275	0.230
8	5.96869	10.39980	0.313	0.308	6.0039	8.93655	0.307	0.179
9	5.93942	11.03630	0.364	0.243	6.0630	9.33957	0.353	0.152
10	5.92609	11.74580	0.419	0.196	6.0940	10.0363	0.399	0.136

Table 4.4 Results comparing the Equi-width with the Averaged R-ACM for sparse attributes when the number of buckets is the same, and $\mu=0.2$

Actual Freq. Value	$\tau = 2$				$\tau = 3$			
	Estimated Values		Estimated Errors		Estimated Values		Estimated Errors	
	Equi-width	Avg. R-ACM	Equi-width	Avg. R-ACM	Equi-width	Averaged R-ACM	Equi-width	Avg. R-ACM
0	1.42369	1.26190	1.424	1.262	1.4641	1.37818	1.464	1.378
5	1.38522	2.56925	0.759	0.567	1.3488	2.19141	0.734	0.612
6	1.43710	2.68582	0.778	0.605	1.3539	2.56259	0.776	0.626
7	1.57232	2.96928	0.790	0.613	1.3585	2.66921	0.807	0.639
8	1.43366	6.39233	0.822	0.293	1.3640	2.92765	0.830	0.654
9	1.36437	7.35554	0.850	0.267	1.3533	3.15208	0.850	0.673
10	1.37302	7.79062	0.863	0.278	1.3628	3.10929	0.864	0.701

Table 4.5 Results comparing the Equi-width with the Averaged R-ACM for sparse attributes when the storage requirements are the same, and $\mu=0.05$

Actual Freq. Value	$\tau = 2$				$\tau = 3$			
	Estimated Values		Estimated Errors		Estimated Values		Estimated Errors	
	Equi-width	Avg. R-ACM	Equi-width	Avg. R-ACM	Equi-width	Averaged R-ACM	Equi-width	Avg. R-ACM
0	2.89858	2.48465	2.899	2.485	2.7985	2.66339	2.799	2.663
5	2.95281	4.58294	0.530	0.301	2.8540	4.0039	0.493	0.305
6	2.80591	4.94817	0.580	0.309	2.7399	4.45986	0.561	0.332
7	2.92268	5.16608	0.603	0.327	2.7711	4.79788	0.609	0.365
8	2.84372	7.41585	0.653	0.215	2.8238	5.11728	0.648	0.386
9	2.89659	8.40906	0.684	0.195	2.8085	5.4936	0.688	0.401
10	2.86758	8.98380	0.714	0.193	2.7502	5.69063	0.725	0.441

Table 4.6 Results comparing the Equi-width with the Averaged R-ACM for sparse attributes when the storage requirements are the same, and $\mu=0.1$

Actual Freq. Value	$\tau = 2$				$\tau = 3$			
	Estimated Values		Estimated Errors		Estimated Values		Estimated Errors	
	Equi-width	Avg. R-ACM	Equi-width	Avg. R-ACM	Equi-width	Averaged R-ACM	Equi-width	Avg. R-ACM
0	4.59856	3.82192	4.599	3.822	4.5230	4.06739	4.523	4.067
5	4.57576	6.46176	0.397	0.338	4.5288	5.66557	0.351	0.223
6	4.60038	6.92471	0.394	0.242	4.4950	6.19978	0.364	0.199
7	4.58526	7.45897	0.414	0.200	4.5749	6.71228	0.383	0.188
8	4.50107	9.01203	0.463	0.197	4.4796	7.27322	0.453	0.188
9	4.71524	9.73562	0.491	0.172	4.4960	7.59619	0.507	0.215
10	4.66234	10.3880	0.540	0.154	4.5694	8.25061	0.545	0.225

Table 4.7 Results comparing the Equi-width with the Averaged R-ACM for sparse attributes when the storage requirements are the same, and $\mu=0.15$

Actual Freq. Value	$\tau = 2$				$\tau = 3$			
	Estimated Values		Estimated Errors		Estimated Values		Estimated Errors	
	Equi-width	Avg. R-ACM	Equi-width	Avg. R-ACM	Equi-width	Averaged R-ACM	Equi-width	Avg. R-ACM
0	5.92439	4.99265	5.924	4.993	6.0240	5.29387	6.024	5.294
5	5.92535	8.01036	0.402	0.602	6.1340	7.23056	0.384	0.447
6	5.96445	8.54346	0.313	0.430	6.0713	7.67693	0.282	0.294
7	6.03754	8.98353	0.296	0.298	6.0842	8.33782	0.270	0.228
8	5.98983	10.4439	0.323	0.311	6.0724	8.88128	0.301	0.177
9	6.01349	11.0548	0.369	0.242	5.9871	9.30486	0.357	0.158
10	5.89851	11.7724	0.423	0.200	6.0369	9.93822	0.404	0.143

Table 4.8 Results comparing the Equi-width with the Averaged R-ACM for sparse attributes when the storage requirements are the same, and $\mu=0.2$

4.3.3 Result Analysis

With regard the overall performance of these two algorithms, it is obvious that the averaged R-ACM has yielded superior estimates over the Equi-width. Also, we notice that the estimations of non-zero frequency values made by the Equi-width are almost the average of all the frequency values, which does not accurately reflect the overall non-zero

frequency distribution *at all*. On the other hand, the estimations made by the averaged R-ACM vary as the non-zero frequencies vary, and are far more accurate.

In order to show the experimental results from another perspective, we present the trends in Figures 4.2 and 4.3. In both Figures 4.2 and 4.3, the X-axis represents the value of μ , and the Y-axis represents the corresponding estimations, except that in the former we depict the *absolute* error for the zero frequency values, and Figure 4.3 depicts the *relative* error for non-zero frequencies. As can be observed from the estimations made on both zero and non-zero frequencies, the averaged R-ACM yields *consistent* superior performance over the Equi-width. This is especially observable when $\mu=0.2$ and $\tau=2$ for zero frequency estimates, and when $\mu=0.15$ for non-zero frequency estimates.

Figure 4.2 Estimations on zero frequencies

Figure 4.3 Estimate errors on non-zero frequencies

Another noticeable perspective inferred from Figure 4.2 and Figure 4.3 is that the error of the estimates made by the Equi-width algorithm remain almost the same while the storage requirement parameters and the tolerance values change. The corresponding quantities for the “same storage requirements” for the averaged R-ACM are marginally inferior to those requiring the “same number of buckets”. Therefore, the extra storage

requirements for the averaged R-ACM do not pose to be a concern. It is clear from the experiments that the averaged R-ACM can still yield superior performance over the Equi-width even with the same storage requirements.

4.4 Conclusions

In this Chapter, we have explained, in fair detail, the issues of determining the QEPs, namely the statistics, the selectivity and the cost. Also, we have discussed the most fundamental and critical considerations involved in integrating the R-ACM and the T-ACM into the query optimizer of a real-life DBMS. The approaches discussed in this Chapter are exactly those that will be used in our implementation, and the corresponding experiment in Chapter 5, where we will extend a real-life ORACLE DBMS to utilize these concepts.

The problems encountered for the sparse data cases have also been explored in this Chapter. This is a relatively new area of research, and the actual available results are few. We have discussed a novel solution to the problems by introducing a modified scenario of the R-ACM, namely the averaged R-ACM. We have also presented the experimental results obtained from the sparse data cases using the averaged R-ACM, and demonstrated its superiority over the well-known traditional methods.

Chapter 5: TESTING WITH TPC-H DATA SETS

In this Chapter, we present the experimental results, obtained by using estimates of query result sizes made by the histogram-based algorithms discussed earlier. In Section 5.1, we explain the testbed used in the experiments. In brief, the database system is ORACLE 9i, and the data sets are the uniformly-distributed TPC-H databases, which are artificially skewed using the multi-fractal distribution. In Sections 5.2 and 5.3, we present the methods used in the experiments and the results obtained from the experiments. Also we shall compare the experimental results with those obtained from the algorithm currently being used by the commercial ORACLE query optimizer, namely the Equi-depth algorithm. One of principle contributions of this Thesis is that the R-ACM and the T-ACM have been integrated into the ORACLE database system. *We believe that this is no small achievement.*

5.1 Testbed Introduction

In this Section, we introduce the testbed used for all the experiments in this Thesis. Since the focus of this Thesis involves utilizing the new histogram-based algorithms in the ORACLE query optimizer, the testbed consists of three major components:

- (i) The underlying database system, which is the objective DBMS we want to incorporate the algorithms into,
- (ii) The data sets, which are the actual data, skewed with certain underlying distributions on which the histograms are tested, and,
- (iii) The sample queries, which are some queries chosen for the experiments to test the performance of the algorithms.

5.1.1 Test Database System

The database system we gave chosen to work with in this Thesis is the ORACLE database system, whose version, the ORACLE 9i Enterprise Edition, has been developed for the Windows XP operating system. For the experiments, our goal is to integrate the histogram-based algorithms, namely the R-ACM and the T-ACM, into the existing database query optimizer to serve the purpose of a “plug-in”. Since the ORACLE database system is one of the most widely used commercial database systems, we believe that this renders our experiment results to be more meaningful. Furthermore, the numerous development tools supplied by the ORACLE DBMS provide great flexibility for application development, and more importantly, make the integration feasible.

5.1.2 Test Data Sets

With regard to the selection of the data sets, our major considerations focused on the *reality* of the data. As discussed earlier, testing and presenting the performance of the R-ACM and T-ACM on a real database system is one of our objectives for this Thesis. If

the data sets we are testing have real-life (or close to real-life) characteristics, the test results are more “accountable” and trustworthy.

Based on the above considerations, the TPC-H data sets became our final choice. The TPC-H benchmark is a decision support benchmark designed and implemented by the Transaction Processing Performance Council (TPC) organization. It is a standardized benchmark in the broader industry database, and is universally accepted. It consists of a data generation tool, a suite of business oriented ad-hoc queries, and concurrent data modifications. The queries, and the data comprising the database have been chosen as an industry-wide pertinent standard. This benchmark illustrates most of the major characteristics required of decision support systems, which involve examining large volumes of data, executing queries with a high degree of complexity, and giving answers to critical business questions.

Although the design of the relations in the TPC-H benchmark is not complicated, it reflects the real-life business requirements in an elegant and complete manner. There are a total of eight relations in the benchmark. Each of them is described in the tables below.

Table Name	Number of Tuples	Primary Key
CUSTOMER	75000	C_CUSTKEY, C_NATIONKEY
LINEITEM	2999671	L_ORDERKEY, L_PARTKEY, L_SUPPKEY
NATION	25	N_NATIONKEY, N_REGIONKEY
ORDERS	750000	O_ORDERKEY, O_CUSTKEY
PART	100000	P_PARTKEY
PARTSUPP	400000	PS_PARTKEY, PS_SUPPKEY
REGION	5	R_REGIONKEY
SUPPLIER	5000	S_SUPPKEY, S_NATIONKEY

Table 5.1 The TPC-H relations with a total capacity of 500MB data

5.1.3 Sample Queries

There are 12 sample queries which are used in our entire experiments. Among them, Q2 to Q12 are simplified TPC-H queries, and Q18', Q19' and Q20' are constructed by us specifically for our suite of experiments. On the other hand, Q6, Q18', Q19' and Q20' are simple equi-selection queries. The rest of queries are equi-join queries, in which Q17 involves two relations in the joins, Q3 involves three relations, Q10 and Q11 involve four relations, Q2, Q7 and Q9 involve five relations, and Q5 involves six relations. These queries are tabulated below.

Query No	Query	Description
Q2	SELECT * FROM PART, SUPPLIER, PARTSUPP, NATION, REGION WHERE R_REGIONKEY = 3 AND N_REGIONKEY = R_REGIONKEY AND S_NATIONKEY = N_NATIONKEY AND P_SIZE = 20 AND PS_PARTKEY = P_PARTKEY AND PS_SUPPKEY = S_SUPPKEY;	This query finds which supplier should be selected to place an order for a given part in a given region.
Q3	SELECT * FROM LINEITEM, ORDERS, CUSTOMER WHERE L_ORDERKEY = O_ORDERKEY AND C_CUSTKEY = O_CUSTKEY AND O_CUSTKEY = 800;	This query retrieves the information about given customer.
Q5	SELECT * FROM ORDERS, LINEITEM, CUSTOMER, SUPPLIER, NATION, REGION WHERE O_ORDERKEY = L_ORDERKEY AND L_SUPPKEY = S_SUPPKEY AND C_CUSTKEY = O_CUSTKEY AND C_NATIONKEY = S_NATIONKEY AND S_NATIONKEY = N_NATIONKEY AND N_REGIONKEY = R_REGIONKEY AND S_NATIONKEY = 8;	This query lists information about suppliers.
Q6	SELECT * FROM LINEITEM WHERE L_QUANTITY = 25;	This query gives LINEITEM information.
Q7	SELECT * FROM LINEITEM, ORDERS, CUSTOMER, SUPPLIER, NATION WHERE S_SUPPKEY = L_SUPPKEY AND L_ORDERKEY = O_ORDERKEY AND C_CUSTKEY = O_CUSTKEY AND C_NATIONKEY = S_NATIONKEY AND N_NATIONKEY = S_NATIONKEY AND S_NATIONKEY = 3;	This query determines the shipping information.

Table 5.2.a. Sample Queries for Query Types Q2, Q3, Q5, Q6 and Q7 of the TPC-H Database

Query No	Query	Description
Q9	SELECT * FROM SUPPLIER, PARTSUPP, LINEITEM, ORDERS, NATION WHERE PS_SUPPKEY = L_SUPPKEY AND L_ORDERKEY = O_ORDERKEY AND S_SUPPKEY = L_SUPPKEY AND S_NATIONKEY = N_NATIONKEY AND S_NATIONKEY =3;	This query determines the information on lines of parts.
Q10	SELECT * FROM LINEITEM, ORDERS, CUSTOMER, NATION WHERE C_CUSTKEY = O_CUSTKEY AND L_ORDERKEY = O_ORDERKEY AND C_NATIONKEY = N_NATIONKEY AND O_CUSTKEY = 5000;	This query identifies customers who have parts shipped to them.
Q11	SELECT * FROM PARTSUPP, SUPPLIER, NATION WHERE PS_SUPPKEY = S_SUPPKEY AND S_NATIONKEY = N_NATIONKEY AND S_NATIONKEY = 3;	This query finds the suppliers' stock in a given nation.
Q12	SELECT * FROM ORDERS, LINEITEM WHERE O_ORDERKEY = L_ORDERKEY AND L_QUANTITY = 25;	This query determines the relationship between the shipping modes and priority orders.
Q18'	SELECT * FROM PART WHERE P_SIZE =20;	This query lists parts information with certain size.
Q19'	SELECT * FROM CUSTOMER WHERE C_NATIONKEY = 10;	This query lists information about customers in given nation.
Q20'	SELECT * FROM SUPPLIER WHERE S_NATIONKEY=3;	This query lists supplier information for a given nation.

Table 5.2.b Sample Queries for Query Types Q9, Q10, Q11, Q12, Q18', Q19' and Q20' of the TPC-H Database

5.2 Test Methods

The test methods used in this Thesis are a flow of procedures used to achieve our fundamental goal, namely, that of integrating the histogram-based algorithms into the query optimizer. While the basic theory of the test methods is completely based on what has been explained in Chapter 4, the architecture of the implementation is composed of three major components: (i) the operators, (ii) the catalogue, and (iii) the algorithms being incorporated (See Figure 5.1.). Recalling Figure 4.1, all of these three components are inside the query optimizer, and furthermore, the components (i) and (iii) are inside the *estimator* routine itself. In other words, we intend to extend the query optimizer by utilizing the new histogram-based algorithms, and computing corresponding estimates using these algorithms.

Figure 5.1 The Fundamental Core Architecture of Our Implementation

5.2.1 Catalogues

As discussed in the previous Chapter, special catalogues are created in the database system in order to store the statistical information for the R-ACM and T-ACM histograms. These specially created catalogues are identical to the other catalogues or relations stored in the ORACLE database system, except that their utility lies in the fact that they are used for the R-ACM and T-ACM histograms. Therefore, these catalogues can be stored in a user's private space, or in a public space which is accessible for all the database users.

Populating these catalogues is accomplished by invoking ORACLE's standard module that is used for collecting statistical information. Deleting the content of these catalogues is done in an analogous manner.

5.2.2 Operators

The term "Operator" is used in ORACLE to allow the query optimizer to compute a user-defined selectivity and cost, and to collect and delete user-defined statistical information. The operator can be treated as an extension of the SQL syntax, and can be used in any standard SQL statement, whenever it is needed.

There are two operators developed especially for our present sets of experiments, namely, the RACM and the TACM, which represent the process of utilizing the R-ACM and the T-ACM algorithms respectively. Following exactly the same methods of computation discussed in the previous Chapters, these operators achieve the following major functions listed below:

(i) Collecting and deleting corresponding statistical information

When the standard collecting or deleting statistics command is invoked, the corresponding statistical information for the R-ACM (or the T-ACM) is alternatively collected or deleted for the underlying attribute of the corresponding relation. In other words, this process has been rendered so automatic that it, inherently, is associated with the entire DBMS.

(ii) Computation of the selectivity

When an operator is invoked in a SQL statement, the selectivity for that particular predicate is computed by using the corresponding algorithm, namely, either the R-ACM or the T-ACM, in the experiments.

(iii) Computation of the cost

Since the cost is a measurement which involves many physical parameters such as the CUP and I/O parameters, and more importantly, the resources required, the testing algorithms are carefully controlled by choosing appropriate tolerance value, for example, for τ . The cost of using the histogram testing algorithms should hence be considered as proportional to that of using the standard histogram algorithm which is currently being used in ORACLE.

5.2.3 Multi-Fractal Distribution

One of the remarkable characteristics of the standard benchmark is that the TPC-H data sets are nearly uniformly distributed. Unfortunately, this is not the phenomenon observed in most real-world data, where the data distributions encountered are, typically, skewed. In order to test the performance of the underlying histogram-like algorithms on

distributions which are more similar to real-life applications, the original TPC-H data distributions were transformed by using a multi-fractal “transformer”. The recursive decomposition methods used in the experiments were conducted based on those explained in [FMS96].

Figure 5.2 Generation of a Multifractal Distribution – first three steps

As illustrated in Figure 5.2(a), the left half is chosen with probability $(1-p)$, while the right half with p . In each interval, the distribution is processed recursively in the same manner. At the end of the process, the left-most bucket will hold $(1-p)^k$ of the probability mass, and the right-most bucket will hold p^k . Therefore, in Figure 5.2(b) the left-most bucket holds $(1-p)^3$ of the probability mass, while the right-most bucket holds p^3 . The other buckets are correspondingly weighted by their respective multiplying factors.

In our reported experiments, the value of the decomposition factor used was 0.2, and the attributes were randomly chosen based on the sample queries as explained in Section 5.3.2. Similar results are available for other values of the decomposition factor.

5.2.4 Special Experiment of Setup for the T-ACM

Because of its nature, the superiority of the T-ACM can only be shown on approximately linear distributions. Clearly, approximating a non-linear function in a linear manner is meaningless! In practice, there are only a few problems which lead to a direct application of the T-ACM. First of all, it is hard to find an exactly matched linear frequency distribution. Secondly, determining what kind of distribution a real-life frequency distribution belongs to, remains an open question, and is yet unsolved.

Our aim here is to work with the spirit of the ideal T-ACM solution, and to also be able to have it applicable for a broader spectrum of distributions. Consider the attribute values for a specific bucket, and J consecutive attribute values in this bucket. The question to be solved is whether the $(J+1)^{\text{st}}$ attribute value should also be in this bucket. Suppose there is a best straight line approximation that would approximate the frequencies of the J values in the bucket. This can be achieved by minimizing the mean-square error between the *approximation* of the frequencies, and the actual frequencies. Indeed, this involves solving a system of equations by computing the generalized inverse (Moore-Penrose inverse) and is fairly straightforward. If now, the extrapolated frequency for the $(J+1)^{\text{st}}$ attribute value differs from the true value by *less* than a user-defined threshold, τ , the $(J+1)^{\text{st}}$ attribute is included in the same bucket and the process is repeated for the next attribute value. Otherwise, the $(J+1)^{\text{st}}$ attribute value becomes the starting point of a new bucket. This also accounts for the name of the structure, it is the *Bounded* T-ACM.

The reader will observe that the true BT-ACM can be computationally expensive. To save on the computation, we introduce a simplified version. In the latter, rather than seek for the best straight-line approximation, we have considered if the frequency of the $(J+1)^{\text{st}}$ attribute in the bucket differs from the frequency of the *first* attribute by at most τ . If it does, this attribute is included in the present bucket ; otherwise, it becomes the start of a new bucket. This renders the computation simple, and makes it more suitable for uniform-like distributions. The pseudo-code for *this version* of the BT-ACM follows.

Algorithm 5.1 Generate_BT-ACM

Input: Tolerance value τ , frequency distribution of attribute X as $A[0..L-1]$.

Output: The BT-ACM

```
begin
  Initialize_ACM; /* set all entries in ACM to zero */
  ACM[1].a := A[0]; /* set  $a_1$  to  $A[0]$  */
  s := 1;
  for j:=0 to L-1 do
    if abs(A[j] - ACM[s].a) <  $\tau$ 
    then ACM[s].n := ACM[s].n + A[j];
    else begin
      ACM[s].b:=2*ACM[s].n/l - ACM[s].a;
      s := s + 1;
      ACM[s].a := A[j+1];
    end;
  end if;
end for;
end
```

End Algorithm Generate_BT-ACM;

Notice that the bucket widths are not all the same, and the frequency variation in a bucket is now bounded to vary by *at most* τ from the straight line approximation. In essence, it preserves the error considerations that motivated the design of the T-ACM. An important consequence of this, is that the BT-ACM is suitable for more kinds of practical distributions, and can thus be tested on the same data sets and the same queries as

described in previous sections. Hence, the experimental results presented in the next Section are tested by using the BT-ACM algorithm, instead of the T-ACM.

5.3 Test Results

In this Section, we shall present the results obtained from conducting the experiments formulated as part of the research work done in this Thesis. Several sets of different experiments were carried out, namely those which evaluate error-rate growth, and experiments involving estimations on query result sizes for uniformly distributed data, and for multi-fractally distributed data by utilizing the R-ACM, the BT-ACM and the Equi-depth algorithms.

5.3.1 Error Rate Growth Experiment

As discussed in previous Chapters, the theoretical analysis concludes that the error rate of the estimate grows exponentially as the join level increases. In our experiments, we would like to verify whether this conclusion holds in practice, especially on a real-life DBMS with “approximately” real-life data. These experiments were conducted using the standard TPC-H data sets, which, as mentioned earlier, are nearly uniformly distributed. A few queries with different level of joins were randomly chosen, as shown below.

- (i) The equi-selection query

```
SELECT *  
FROM SUPPLIER  
WHERE S_NATIONKEY = 22;
```


(ii) The first level of joins

```
SELECT *
FROM SUPPLIER, CUSTOMER
WHERE S_NATIONKEY = C_NATIONKEY
AND S_NATIONKEY = 22;
```

(iii) The second level of joins

```
SELECT *
FROM SUPPLIER, CUSTOMER, ORDERS
WHERE S_NATIONKEY = C_NATIONKEY
AND O_CUSTKEY = C_CUSTKEY
AND S_NATIONKEY = 22;
```

(iv) The third level of joins

```
SELECT *
FROM SUPPLIER, CUSTOMER, ORDERS, PARTSUPP
WHERE S_NATIONKEY = C_NATIONKEY
AND O_CUSTKEY = C_CUSTKEY
AND S_SUPPKEY = PS_SUPPKEY
AND S_NATIONKEY = 22;
```

The corresponding results obtained are given in Table 5.3 below.

Query	Actual Result Size	Estimated Result Size		Error Rate (%)	
		R-ACM	Equi-depth	R-ACM	Equi-depth
Equi-selection	191	194	201	1.570681	5.23560
First Level	575865	582000	601732	1.065354	4.49185
Second Level	5754448	5820000	9026345	1.139154	56.85857
Third Level	460355840	457440000	722107609	0.633388	56.85857

Table 5.3 Results Demonstrating the Growth of the Error Rate with the Number of Joins

It can be seen easily from Table 5.3 that the error rate of the estimations grows when the join level increases. Note that for the Equi-depth algorithm, there is a huge

jump in the error rate when the join level increases to two levels from only one level, and for the R-ACM the error rate does not change much when the join level changes. Thus, this quantity for the equi-join increases from approximately 4.49% to 56.85%. The analogous figures for the R-ACM are from approximately 1.07% to 0.63%. The advantages are obvious.

5.3.2 R-ACM vs. Equi-depth with Uniform Distributions Experiments

Experiments in this Section and the following sections are conducted so as to compare the performance of different algorithms on estimating query result sizes. There are two sets of comparisons, namely the R-ACM vs. the Equi-depth, and the BT-ACM vs. the Equi-depth. Also, the comparisons are made with two different kinds of data distributions, namely the uniform and the multifractal distributions.

For each set of experiments, all the sample queries listed in Table 5.2 are tested. Particularly for the R-ACM experiments, two different storage requirement options have also been chosen, namely with the option of using the same number of buckets, and the option of using the same storage requirements. The formula used to compute the storage requirements is shown below.

$$S_{(Equi-depth)} = 2 * S_{(R-ACM)} - 1,$$

where s is the number of buckets.

With respect to the histograms built for the attributes in the sample queries, a few issues have to be clarified. First, since the R-ACM and the BT-ACM are integrated into the ORACLE query optimizer by using their corresponding “operators”, they can perform all the same functions which the default histogram algorithm, the Equi-depth does.

However, the only problem with the current implementation of the integration is that the operators must be presented every time that the underlying algorithm has to be invoked on the specified attribute. This renders the query a little more complex specially when there are a number of predicates in the query.

To simplify this problem, in our current implementation, we chose only a few attributes for which we built the underlying histograms. Attributes in this category are those appearing in the equi-selection predicates. The reason why we can achieve such a simplification is because of a fundamental fact that we discovered after a careful investigation of the ORACLE's query optimizer. As far as we understand, the ORACLE query optimizer utilizes histograms in estimating query result sizes for equi-selections, while it does *not* utilize these histograms for joins^[PC02]. This is quite important for the correctness and reliability of our experimental results. Thus, the attributes in our relations, namely, "C_NATIONKEY", "L_QUANTITY", "O_CUSTKEY", "P_SIZE", and "S_NATIONKEY", have different underlying histograms built.

The experimental results of the R-ACM vs. Equi-depth for uniform distributions are shown below.

^[PC02] Personal communication with some ORACLE experts: This has been confirmed informally.

Query	Actual Result size	Estimated Result size			Error Rate (%)		
		R-ACM (Same # of buckets)	R-ACM (Same Storage)	Equi-depth	R-ACM (Same # of buckets)	R-ACM (Same Storage)	Equi-depth
Q2	1,533	1,538	1,597	1,601	0.326158	4.174821	4.435747
Q3	30	40	40	71	33.333333	33.333333	136.666667
Q5	5,203	5,135	4,967	7,220	1.306938	4.535845	38.766111
Q6	60,251	60,222	60,282	59,994	0.048132	0.051451	0.426549
Q7	5,190	5,021	4,967	7,220	3.256262	4.296724	39.113681
Q9	10,111,600	10,042,899	9,934,910	9,624,692	0.679428	1.747399	4.815341
Q10	39	41	40	71	5.128205	2.564103	82.051282
Q11	16,800	16,740	16,560	16,043	0.357143	1.428571	4.505952
Q12	60,251	60,222	60,282	59,994	0.048132	0.051451	0.426549
Q18'	1,997	1,997	1,997	2,001	0	0	0.200300
Q19'	2,961	2,961	2,980	3,002	0	0.641675	1.384667

Table 5.4 Experimental Results of the R-ACM vs. Equi-depth for Uniform Distributions

5.3.3 R-ACM vs. Equi-depth with Multi-Fractal Distributions Experiments

The experimental results of the R-ACM vs. Equi-depth with multi-fractal distributions are shown in Table 5.5 below.

Query	Actual Result size	Estimated Result size			Error Rate (%)		
		R-ACM (Same # of buckets)	R-ACM (Same storage)	Equi-depth	R-ACM (Same # of buckets)	R-ACM (Same storage)	Equi-depth
Q2	1256	1282	1390	1098	2.070	10.6688	12.580
Q3	3702	43227	34671	104881	1067.666	836.5478	2733.090
Q5	71	977	365	5669	1276.056	414.0845	7884.507
Q6	196585	196585	196585	187479	0	0	4.632
Q7	13	45	87	5669	246.154	569.2308	43507.690
Q9	1519760	920629	2523616	12970317	39.423	66.0536	753.445
Q10	3702	42498	34087	104881	1047.974	820.7726	2733.090
Q11	2560	1535	4206	21620	40.0391	64.2969	744.531
Q12	196585	196585	196585	187449	0	0	4.647
Q18'	1638	1638	1775	1402	0	8.3639	14.408
Q19'	1536	1536	1232	3727	0	19.7917	142.643

Table 5.5 Experimental Results of the R-ACM vs. Equi-depth for Multi-Fractal Distributions ($p = 0.2$)

It is obvious that the R-ACM yielded superior performance over the Equi-depth for all the queries, and with any storage requirements considerations, whether the R-ACM had the same number of buckets or the same storage requirements. Also noticeable is the outstanding performance of the R-ACM for the multi-fractal distributions (See Tables 5.4 and 5.5). For example, in Table 5.4 when we consider the option of using the same storage, the error rates of sample queries Q3, Q5, Q6, Q10, Q11 and Q18' for the R-ACM are greatly less than those for the Equi-depth. More specifically, for the sample query Q10 which involves a four relation-join, the error rate for the R-ACM, which is approximately 2.56%, is nearly 40 times less than that for the Equi-depth, which is approximately 82.05%. Similarly, consider Table 5.5 where we analyze the option of using the same storage. In this case, the sample query Q7 involves five relations in the underlying join. The error rate of this query for the R-ACM, which is approximately 569.23%, is nearly 80 times less than that for the Equi-depth, which is approximately 43507.69%. These results and the tables in their entirety, prove the superiority of the R-ACM on estimating query result sizes. The concern of increasing the storage requirements for the R-ACM is not really a valid concern.

5.3.4 BT-ACM vs. Equi-depth with Uniform Distributions Experiments

The T-ACM operator was earlier discussed in Section 5.3.2. There are argued that it can not yield superior performance on non-linear distributions. As a variation of the T-ACM, the BT-ACM keeps the most significant feature of the T-ACM, which is, that each bucket has a trapezoidal shape. It is obvious that it can not yield superior performance on the multi-fractal distributions, since the multi-fractal distributions are the extreme cases of

non-uniform distributions. For these reasons, only uniform distributions were tested for methods using the BT-ACM histograms.

Different from the tests conducted in the previous experiments, we chose four equi-selection queries, and tested all the possible values for the underlying attributes. The experimental results are shown in the following table.

Query	Number of Better Estimations		Overall Estimation Error Rate (%)	
	BT-ACM	Equi-depth	BT-ACM	Equi-depth
Q6	38	12	0.120071	0.3354168
Q18	41.5	8.5	0.618160	2.1322956
Q19'	19	6	0.51595	1.2104582
Q20'	21	4	1.429133	4.4767138

Table 5.6 Experimental Results of the BT-ACM vs. Equi-depth for Uniform Distributions

For the BT-ACM experiments, we observe in the first place, that the number of buckets is far less than those used for the Equi-depth algorithm. Therefore we can effectively consider both of these two algorithms to be having the same storage requirements. Note that during the experiments, a tolerance value, which is similar to the one used in the R-ACM, was carefully chosen to yield more accurate estimates, and to control the number of generated buckets.

From Table 5.6, we can see that the number of cases when the BT-ACM yields better estimates is much more than when the Equi-depth is superior. Besides, the BT-ACM yields superior overall performance. We are thus fairly confident that we can effectively utilize the BT-ACM in approximating frequency distributions for these database applications.

5.4 QEP Selection Comparison

As mentioned earlier, another contribution of this Thesis is to demonstrate that better QEPs are chosen by utilizing the R-ACM in the ORACLE query optimizer. To show this we present more details of our “plug-in” implementation.

First of all, we mention that in our entire implementation, we did not modify or change the fundamental rules of how the ORACLE query optimizer selects the “best” QEP. As described earlier, what we have implemented can be considered as an extension to the original ORACLE query optimizer, which allows it to utilize external histogram algorithms to achieve the task of estimating query result sizes and choosing QEPs. Thus, in our entire implementation, we did not consider the issues involving QEP search-space pruning and QEP selection rules optimization. Instead, these are accomplished by following the design and execution of the original ORACLE query optimizer.

Since the tolerance value, τ , was properly chosen (as described earlier), the storage requirements for the R-ACM and the Equi-depth can be effectively considered to be the same, and hence the cost for the operations in the QEPs involving these algorithms can therefore be considered to be identical. As a result of this, we can now compare these two algorithms by comparing the final costs of the QEPs that they respectively choose. The accuracy of the selectivity, which is determined by the estimate of the query result sizes, is therefore the most critical factor in selecting the optimal QEP. In this connection we observe that with respect to the join orders for join operations, it is also true that the R-ACM seems to always choose superior solutions. To illustrate this, we would like to take a few examples showing the respective QEPs chosen by using these two algorithms.

It is certainly difficult to explain this and present the trace for a *large* number of cases. But we shall do it for a few representative queries.

Example 5.1. Consider the last query in Section 5.3.1, where the results given below are obtained from the uniform distribution.

```
SELECT *
FROM SUPPLIER, CUSTOMER, ORDERS, PARTSUPP
WHERE S_NATIONKEY = C_NATIONKEY
AND O_CUSTKEY = C_CUSTKEY
AND S_SUPPKEY = PS_SUPPKEY
AND S_NATIONKEY = 22;
```

The corresponding QEPs are given in Figure 5.3.

Figure 5.3 The QEPs Chosen for the Sample Query

The cost for each step of the operations in the QEPs is shown in parenthesis in Figure 5.3. Note that the join methods chosen by the R-ACM for each join operation is the hash join, which is the most efficient method when a join operation returns a large

number of tuples. Subsequently, the cost in (b), for every step after the first step of the join (which is represented at the bottom of the operation tree), is much larger than those in (a). Furthermore, the final cost for the Equi-depth's QEP is significantly larger than of for the R-ACM's QEP which is shown at the top of the operation tree.

Example 5.2. All the sample queries from the TPC-H database have been tested using the R-ACM and the Equi-depth, based on the same storage configurations. The results (presented below) showed that all the QEPs chosen for the R-ACM are better than those chosen for the Equi-depth. This is true for criteria measured in terms of both the cost and the efficiency. We list below the SQL statements for the queries, and the screenshots for the chosen QEPs. Rather than give a detailed analysis of all the cases (which can be quite repetitive), we shall conclude this example with an analysis of **two** particular scenarios.

Example 5.2.a

SQL Queries for Q2.

```
SELECT *
FROM PART, SUPPLIER, PARTSUPP, NATION, REGION
WHERE R_REGIONKEY = 3
AND N_REGIONKEY = R_REGIONKEY
AND S_NATIONKEY = N_NATIONKEY
AND P_SIZE = 20
AND PS_PARTKEY = P_PARTKEY
AND PS_SUPPKEY = S_SUPPKEY;
```

Actual result size: 1,533

Screenshots for the two QEPS chosen for Q2.

Figure 5.4(a) QEP of Query Q2 for Equi-depth

Figure 5.4(b) QEP of Query Q2 for R-ACM

Figure 5.4 QEPs Chosen for Query 2.

Example 5.2.b

SQL Queries for Q3.

```
SELECT *
FROM LINEITEM, ORDERS, CUSTOMER
WHERE L_ORDERKEY = O_ORDERKEY
AND C_CUSTKEY = O_CUSTKEY
AND O_CUSTKEY = 800;
```

Actual result size: 30

Screenshots for the two QEPS chosen for Q3.

Figure 5.5(a) QEP of Query Q3 for Equi-depth

Figure 5.5(b) QEP of Query Q3 for R-ACM

Figure 5.5 QEPs Chosen for Query 3.

Example 5.2.c

SQL Queries for Q5.

```
SELECT * FROM
ORDERS, LINEITEM, CUSTOMER, SUPPLIER, NATION, REGION
WHERE O_ORDERKEY = L_ORDERKEY
AND L_SUPPKEY = S_SUPPKEY
AND C_CUSTKEY = O_CUSTKEY
AND C_NATIONKEY = S_NATIONKEY
AND S_NATIONKEY = N_NATIONKEY
AND N_REGIONKEY = R_REGIONKEY
AND S_NATIONKEY = 8;
```

Actual result size: 5,203

Screenshots for the two QEPS chosen for Q5.

Figure 5.6(a) QEP of Query Q5 for Equi-depth

Figure 5.6(b) QEP of Query Q5 for R-ACM

Figure 5.6 QEps Chosen for Query 5.

Example 5.2.d

SQL Queries for Q6.

```
SELECT * FROM LINEITEM WHERE L_QUANTITY = 25;
```

Actual result size: 60,251

Screenshots for the two QEPS chosen for Q6.

Figure 5.7(a) QEP of Query Q6 for Equi-depth

Figure 5.7(b) QEP of Query Q6 for R-ACM

Figure 5.7 QEPs Chosen for Query 6.

Example 5.2.e

SQL Queries for Q7.

```
SELECT *
FROM LINEITEM, ORDERS, CUSTOMER, SUPPLIER, NATION
WHERE S_SUPPKEY = L_SUPPKEY
AND L_ORDERKEY = O_ORDERKEY
AND C_CUSTKEY = O_CUSTKEY
AND C_NATIONKEY = S_NATIONKEY
AND N_NATIONKEY = S_NATIONKEY
AND S_NATIONKEY = 3;
```

Actual result size: 5,190

Screenshots for the two QEPS chosen for Q7.

Figure 5.8(a) QEP of Query Q7 for Equi-depth

Figure 5.8(b) QEP of Query Q7 for R-ACM

Figure 5.8 QEPs Chosen for Query 7.

Example 5.2.f

SQL Queries for Q9.

```
SELECT *
FROM SUPPLIER, PARTSUPP, LINEITEM, ORDERS, NATION
WHERE PS_SUPPKEY = L_SUPPKEY
AND L_ORDERKEY = O_ORDERKEY
AND S_SUPPKEY = L_SUPPKEY
AND S_NATIONKEY = N_NATIONKEY
AND S_NATIONKEY = 3;
```

Actual result size: 10,111,600

Screenshots for the two QEPS chosen for Q9.

Figure 5.9(a) QEP of Query Q9 for Equi-depth

Figure 5.9(b) QEP of Query Q9 for R-ACM

Figure 5.9 QEPs Chosen for Query 9.

Example 5.2.g

SQL Queries for Q10.

```
SELECT *  
FROM LINEITEM, ORDERS, CUSTOMER, NATION  
WHERE C_CUSTKEY = O_CUSTKEY  
AND L_ORDERKEY = O_ORDERKEY  
AND C_NATIONKEY = N_NATIONKEY  
AND O_CUSTKEY = 5000;
```

Actual result size: 39

Screenshots for the two QEPS chosen for Q10.

Figure 5.10(a) QEP of Query Q10 for Equi-depth

Figure 5.10(b) QEP of Query Q10 for R-ACM

Figure 5.10 QEPs Chosen for Query 10.

Example 5.2.h

SQL Queries for Q11.

```
SELECT * FROM PARTSUPP, SUPPLIER, NATION
WHERE PS_SUPPKEY = S_SUPPKEY
AND S_NATIONKEY = N_NATIONKEY
AND S_NATIONKEY = 3;
```

Actual result size: 16,800

Screenshots for the two QEPS chosen for Q11.

Figure 5.11(a) QEP of Query Q11 for Equi-depth

Figure 5.11(b) QEP of Query Q11 for R-ACM

Figure 5.11 QEps Chosen for Query 11.

Example 5.2.i

SQL Queries for Q12.

```
SELECT *  
FROM ORDERS, LINEITEM  
WHERE O_ORDERKEY = L_ORDERKEY  
AND L_QUANTITY = 25;
```

Actual result size: 60,251

Screenshots for the two QEPS chosen for Q12.

Figure 5.12(a) QEP of Query Q12 for Equi-depth

Figure 5.12(b) QEP of Query Q12 for R-ACM

Figure 5.12 QEPs Chosen for Query 12.

Example 5.2.j

SQL Queries for Q18'.

```
SELECT *  
FROM PART  
WHERE P_SIZE =20;
```

Actual result size: 1,997

Screenshots for the two QEPS chosen for Q18'.

Figure 5.13(a) QEP of Query Q18' for Equi-depth

Figure 5.13(b) QEP of Query Q18' for R-ACM

Figure 5.13 QEPs Chosen for Query 18'

Example 5.2.k

SQL Queries for Q19'.

```
SELECT *  
FROM CUSTOMER  
WHERE C_NATIONKEY = 10;
```

Actual result size: 2,961

Screenshots for the two QEPS chosen for Q19'.

Figure 5.14(a) QEP of Query Q19' for Equi-depth

Figure 5.14(b) QEP of Query Q19' for R-ACM

Figure 5.14 QEPs Chosen for Query 19'

Analysis of the QEPs for Q11 and Q5.

Consider the query Q11, where the results given in Figure 5.11 are obtained from the uniform distribution. The traces of the chosen QEPs show that, for this query, the cost of the QEP for the R-ACM, 861, is less than that of the QEP for the Equi-depth, which is 908. And the estimate of the ultimate query result size by using the R-ACM is more accurate than that obtained by using the Equi-depth, where the real query result size is

16,800. The estimate for the Equi-depth is 16,043, and for the R-ACM is 16,560. The superiority of the R-ACM is clear.

The traces of the QEPs selected for Query 5 are shown in Figures 5.6 (a) and (b) for the Equi-depth and the R-ACM algorithms respectively. This is the query which involves the most relations from the set of sample queries. The ultimate cost of the QEP for the R-ACM was 11,119, while that for the Equi-depth was 13,466. The final query result size of this sample query made by the R-ACM was 4,967, and the corresponding quantity for the Equi-depth was 7,220. Notice that the real result size is 5,203. It is quite easy to see that the R-ACM leads to a lower cost, and to a more accurate estimation of the ultimate query result sizes than the Equi-depth. Also, we can conclude that the lower costs result purely from the more accurate estimates, because the selectivity criterion is the only issue that we considered when we attempted to obtain improvements for the QEP selection process. Clearly, the performance of the QEP is also improved by this enhancement.

The results presented here are representative and true for all the queries from the TPC-H benchmark. It appears as if the R-ACM *always* leads to superior QEPs than the Equi-depth scheme.

5.5 Conclusions

In this Chapter, we have considered how the newly introduced histograms, the R-ACM and the T-ACM, perform in a real-life database system. We have explained various aspects of the testbed, namely the database system, the data sets and the sample queries

with which we have experimented. We have also discussed the major implementation issues involved in the integration of the new histograms, namely those involving the catalogues and the operators, and the uniform and non-uniform frequency distributions used in the experiments. Also, since the basic T-ACM is not directly applicable for most distributions, we have extended it to yield a modified T-ACM, referred to as the BT-ACM. The latter has been tested in all the experiments that we have conducted.

A major contribution of this Thesis is the integration of the new histograms in the actual ORACLE DBMS. This Chapter contains the experimental results that have been obtained in this regard. These issues have been presented in terms of the accuracy of the estimates, and in terms of the efficiency of the QEP ultimately selected.

Chapter 6: CONCLUSIONS

The field of databases is central to a large variety of applications in modern-day computer science. As the sizes and the complexity of databases grow, their efficiency is becoming even more important, and obviously critical to the performance of the entire application. In this context, the estimation of query result sizes, which is one of fundamental functionalities performed by the query optimizer in any DBMS, is an active research area, and has been the focus of years of research.

In this Thesis, we have focused on the utilization of some new histogram-based techniques for estimating the query result sizes, namely the R-ACM and the T-ACM. These two algorithms have been integrated into the most widely used commercial DBMS, ORACLE, for queries with equi-selections and equi-joins. Our main goal in this research work was to incorporate these two algorithms into some real-life DBMS. The superiority of these two algorithms in accurately estimating query result sizes in the ORACLE DBMS has also been verified by the experimental results obtained. The integration has been successfully accomplished by using various development tools provided by ORACLE.

6.1 Contributions

The main contributions of this Thesis can be summarized below.

As part of the background research of this Thesis, we have conducted a comprehensive literature survey and a survey of the patents which used techniques similar to the histogram algorithms studied here. The survey showed us that histogram techniques have been used in various areas and industries, which were far beyond what we had imagined. The results of the survey, and a brief comparison of some of the patents with the R-ACM and the T-ACM, were given in Chapter 2.

Apart from this, the main contributions are itemized below.

(i) Use The R-ACM in Sparse Data Cases

The sparse data cases, which contain zero frequencies for great number of attribute values for a given attribute, have been an open area for research. In this Thesis, we have explored the applicability of the R-ACM for attributes of this nature. We proposed a specially designed data structure and its associated algorithm, namely the Averaged R-ACM, in Chapter 4. In brief, it consisted of two phases of processing, a value distribution compression phase which uses an Equi-width-like module, and an averaged R-ACM computation obtained by invoking the R-ACM. The results from the experiments, which were compared with those obtained by using the straightforward Equi-width method, have also been presented in Chapter 4.

(ii) The BT-ACM – an enhancement on the T-ACM

Apart from having discussed the issues of utilization of the T-ACM in practice, we have also modified it by introducing a new rule, the BT-ACM. In the latter, we enforce that the frequencies of attribute values within a bucket follow the same linear trend. This modification enabled us to carry out the experiments using the same data sets, queries and data distributions as those used for the R-ACM experiments. The experimental results have been given in Chapter 5.

(iii) TPC-H Benchmarking Tests

We also conducted an extensive set of experiments on both the R-ACM and the T-ACM. These experiments used the industrial standard benchmarking queries and databases, namely the TPC-H, where the flat data size of the database was 512MB. Since the attribute frequency distributions of the TPC-H database are nearly uniform, we tested the performance of these two algorithms on a few exceptionally skewed distributions, involving the multi-fractal distributions, which were composed based on the standard TPC-H databases.

Another aspect which has been considered in this Thesis was the cost of the algorithms in run-time. Therefore, all in our experiments, we considered two options of storage and memory (in other words, the CPU and I/O costs in run-time), and tested the algorithms for the values of different distributions. All the experimental results obtained by using these two algorithms have been compared with those obtained by using the original histogram algorithm in ORACLE, namely the Equi-depth, in Chapter 5.

(iv) Integration the R-ACM and the T-ACM into ORACLE

Most important of all, we have integrated the R-ACM and the T-ACM into the query optimizer of ORACLE DBMS. In order to maintain its original architecture and mechanism, and conveniently compare different histogram algorithms, the integration has been designed and implemented in a way that it can be considered as a “plug-in”. The entire implementation is implemented in the programming language C++ (or C), and PL/SQL (which is a program language provided by ORACLE). Since the implementation works like a “plug-in”, it allows users to easily invoke the underlying histogram algorithms in queries where they are needed, so as to improve the accuracy of the estimation of query result sizes.

6.2 Future Work

Since the area of databases is so vast, there is still much work to be done in the future regarding the problems studied in this Thesis. In particular, a lot of work remains to be done to extend the functionalities of the implementation discussed in this Thesis. Since the histogram algorithms, namely the R-ACM and the T-ACM have already been shown to be superior to their earlier counterparts, we would definitely like to see them have a greater influence in improving the performance of databases in the future. We list below a few potential areas.

(i) Cost Considerations

As discussed in Chapter 5, in the implementation of the “plug-in” detailed in this Thesis, we assumed that the cost for all underlying histogram algorithms was the same, because the run-time cost were carefully controlled in the experiments. However, it is not easy to accomplish this in practice. Assigning specific costs for the underlying algorithms will become important if they are to be used in real-life, and especially, commercial applications. Although the cost is related to many physical factors, such as the CPU, memory capacity, and I/O speed, we believe that it is not a difficult task.

(ii) Join Order Optimization

The join order in QEPs is an important aspect which determines the efficiency of the query. We have shown that more optimal QEPs have been chosen by using the R-ACM instead of the Equi-depth. A consequence of this is that the join order in the QEPs chosen by the former was superior to the QEPs chosen by the latter. However, the improvement then can be obtained is not optimized as yet. Indeed, the whole area of optimizing join order in the query optimizer is open to future research. Unfortunately, due to the complexity of all possible queries, this is far from trivial, and it could easily lead to future Graduate and Doctoral theses.

(iii) Extention To Non-equi Queries

We have explained in Chapter 5 that the sample queries tested in this Thesis are limited to equi-selections and equi-joins. These are, indeed, only a small portion of the queries used in practice. The issue of testing enhanced implementations using the R-ACM and the T-ACM is open. For example, non-equi-selection and aggregation queries, which are

predominant, and largely used in decision support applications and data warehouse applications, could be the next ones to be implemented and tested.

(iv) Integration Into Other Commercial DBMSs

To improve the flexibility of the implementation in this Thesis, we propose that these histograms be also integrated into commercial DBMSs other than ORACLE. Although ORACLE is currently the most widely used commercial DBMS, other DBMSs, such as SQL Server, and SYBASE, still have quite a large share of the market. However, the only issue is that, different DBMSs may have different architectures and mechanisms in performing the same database application tasks. The only way to successfully integrate the histogram algorithms, namely the R-ACM and the T-ACM, into other DBMSs is by carefully investigating how this can be done for the particular system, and by utilizing the peculiarities of the specific systems in the optimization.

6.3 Summary

In summary, in this Thesis we have studied application of two new histogram-based algorithms, and attempted to integrate them into the ORACLE DBMS. By experimental validation, these algorithms have demonstrated that they are superior in estimating query result sizes, and in yielding QEPs, to the traditional one, namely the Equi-depth algorithm. We believe that these algorithms will be much more widely used in the future, even as their superiority is demonstrated in more general terms.

Bibliography

- [AbC99] Ashraf Aboulnaga and Surajit Chaudhuri, "Self-tuning Histograms: Building Histograms Without Looking at Data", in *Proceedings of SIGMOD Conference*, 1999, pp. 181-192.
- [ALU01] F. N. Afrati, C. Li and J. D. Ullman, "Generating efficient plans for queries using views", in *Proceedings of SIGMOD Conference*, 2001, pp. 319-330.
- [AMS99] N. Alon, Y. Matias and M. Szegedy, "Apparatus And Method For Approximating Frequency Moments", Patent Number: 5,950,185, Sep 7, 1999.
- [AnV01] M. Andrei and P. Valduriez, "User-Optimizer Communication using Abstract Plans in Sybase ASE", in *Proceedings of Conference on Very Large Data Bases*, 2001, pp. 29-38.
- [BFI91] K. Bennett, M. C. Ferris and Y. E. Ioannidis, "A Genetic Algorithm for Database Query Optimization", in *Proceedings of the Fourth International Conference on Genetic Algorithms*, 1991, pp. 400-407.
- [BIE77] M. W. Blasgen and K. P. Eswaran, "Storage and Access in Relational Data Bases", *IBM System Journal* 16, 4, 1977, pp. 363-377.
- [BMU86] F. Bancilhon, D. Maier, Y. Sagiv and J. D. Ullman, "Magic Sets and Other Strange Ways to Implement Logic Programs", in *Proceedings of the Fifth ACM SIGMOD-SIGACT Symposium on Principles of Database Systems*, 1986, pp. 1-16.
- [CCK00] J. Y. Cai, V. T. Chakaravarthy, R. Kaushik and J. F. Naughton, "On the complexity of join predicates", in *Proceedings of the Nineteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, 2000, pp. 207-214.

- [CCM00] M. Charikar, S. Chaudhuri, R. Motwani and V. Narasayya , “Towards estimation error guarantees for distinct values”, in *Proceedings of the Nineteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, 2000, pp. 268–279.
- [CDD01] S. Chaudhuri, G. Das and M. Datar, “Overcoming Limitations of Sampling for Aggregation Queries”, in *Proceedings of the Seventeenth International Conference on Data Engineering*, 2001, pp. 534-542.
- [Cha98] S. Chaudhuri, “An Overview of Query Optimization in Relational Systems”, *Microsoft Research*, 1998.
- [ChO03] J. Chen, and J. Oommen, “Integrating the R-ACM and T-ACM into the ORACLE”, 2003.
- [Chr84] S. Christodoulakis, “Implications of certain assumptions in database performance evaluation”, *ACM Trans. Database Syst.* 9, 2 (June), 1984, pp. 163-186.
- [ChS95] S. Chaudhuri and K. Shim, “An Overview of Cost-based Optimization of Queries with Aggregates”, *Data Engineering Bulletin* 18(3), 1995, pp. 3-9.
- [CKA92] C. A. Czeisler, R. E. Kronauer, and J. S. Allan, “Assessment And Modification of Endogenous Circadian Phase and Amplitude”, Patent Number: 5,167,228, Dec 1, 1992.
- [CKK00] Z. Chen, N. Koudas, F. Korn and S. Muthukrishnan, “Selectively estimation for Boolean queries”, in *Proceedings of the Nineteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, 2000, pp. 216-225.
- [CoG94] R. L. Cole and G. Graefe, “Optimization of Dynamic Query Evaluation Plans”, in *Proceedings of SIGMOD Conference*, 1994, pp. 150-160.
- [FMS96] C. Faloutsos, Y. Matias, and A. Silberschatz, “Modeling skewed distributions using multifractals and the ’80-20 law””, in *Proceedings of Conference on Very Large Data Bases*, 1996, pp. 307-317.
- [FrS00] S. H. Friend, and R. Stoughton, “Method For Determining The Presence Of A Number Of Primary Targets Of A Drug”, Patent Number: 6,146,830, Nov 14, 2000.
- [GGM96] P. Godfrey, J. Gryz and J. Minker, “Semantic Query Optimization for Bottom-Up Evaluation”, in *Proceedings of the Ninth International*

Symposium on Methodologies for Intelligent Systems (ISMIS), Zakopane, Poland, June 1996, pp. 561-571.

- [GKM00] Anna C. Gilbert, Yannis Kotidis, S. Muthukrishnan and Marin J. Strauss, "Optimal and approximate computation of summary statistics for range aggregates", in *Proceedings of the Nineteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, 2000, pp. 227-236.
- [GLQ99] J. Gryz, L. Liu and X. Qian, "Semantic Query Optimization Techniques in IBM DB2: Initial Results", *Technical Report CS-1999-01*, York University, February 19, 1999.
- [GoG96] P. Godfrey and J. Gryz, "Intensional Query Optimization", *workshop on Deductive Databases and Logic Programming*, 1996, pp. 57-68.
- [GoL01] J. Goldstein and P. Larson, "Optimizing queries using materialized views: a practical, scalable solution", in *Proceedings of SIGMOD Conference*, 2001, pp. 331-342.
- [GrW89] G. Graefe and K. Ward, "Dynamic Query Evaluation Plans", in *Proceedings of SIGMOD Conference*, 1989, pp. 358-366.
- [GTK01] Lise Getoor, Benjamin Taskar and Daphne Koller, "Selectivity Estimation using probabilistic models", in *Proceedings of SIGMOD Conference*, 2001, pp. 461-472.
- [HAI90] M. Hara, Y. Adachi, M. Ishida, and N. Nakjima, "Radiation Image Read-out Method", Patent Number: 4,950,894, Aug 21, 1990.
- [HaS92] P. J. Haas and A. N. Swami, "Sequential Sampling Procedures for Query Size Estimation", in *Proceedings of SIGMOD Conference*, 1992, pp. 341-350.
- [HNP95] J. M. Hellerstein, J. F. Naughton and A. Pfeffer, "Generalized Search Trees for Database Systems", in *Proceedings of Conference on Very Large Data Bases*, 1995, pp. 562-573.
- [HNS95] P. J. Haas, J. F. Naughton, S. Seshadri and L. Stokes, "Sampling-Based Estimation of the Number of Distinct Values of an Attribute", in *Proceedings of Conference on Very Large Data Bases*, 1995, pp. 311-322.
- [IbK84] T. Ibaraki and T. Kameda, "On the optimal nesting order for computing N-relational joins", in *ACM Transactions on Database Systems*, 9(3), September 1984, pp. 482-502.

- [Ioa91] Y. E. Ioannidis, and Younkyung Cha Kang, "Left-deep vs. Bushy Trees: An Analysis of Strategy Spaces and Its Implications for Query Optimization", in *Proceedings of SIGMOD Conference*, 1991, pp. 168-177.
- [Ioa95] Yannis Ioannidis, "Histogram-Based Solutions to Diverse Database Estimation Problems", *IEEE Computer Society*, Vol. 8 No. 3, Sep 1995, pp. 10-18.
- [Ioa96] Y. Ioannidis, "Query Optimization", *ACM Computing Surveys, Symposium Issue on the 50th Anniversary of ACM*, Vol. 28, No. 1, March 1996, pp. 121-123.
- [IoP95] Y. Ioannidis and V. Poosala, "Balancing Histogram Optimality and Practicality for Query Result Size Estimation", in *Proceedings of SIGMOD Conference*, San Jose, CA, May 1995, pp. 233-244.
- [IoP99] Yannis E. Ioannidis and Viswanath Poosala, "Histogram-Based Approximation of Set-Valued Query-Answers", in *Proceedings of Conference on Very Large Data Bases*, 1999, pp. 174-185.
- [JaK84] M. Jarke and J. Koch, "Query Optimization in Database Systems", *ACM Computing Surveys*, Vol. 16, No. 2, June 1984, pp. 111-152.
- [JiS00] T. Jim and D. Suciu, "Dynamically distributed query evaluation", in *Proceedings of the Nineteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, 2000, pp. 28 – 39.
- [JNS02] H. V. Jagadish, R. Ng, and D. Srivastava, "Method And Apparatus for Substring Selectivity Estimation", Patent Number: US 6,401,088 B1, Jun 4, 2002.
- [Koo80] R. P. Kooi. "The Optimization of Queries in Relational Databases", PhD thesis, Case Western Reserve University, Sep 1980.
- [Kor99] M. Kornacker, "High-Performance Extensible Indexing", in *Proceedings of Conference on Very Large Data Bases*, 1999, pp. 699-708.
- [KMS00] Nick Koudas, S. Muthukrishnan and Divesh Srivastava, "Optimal histograms for hierarchical range queries (extended abstract)", in *Proceedings of the Nineteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, 2000, pp. 196–204.

- [LiN90] R. J Lipton and J. F Naughton, "Practical Selectivity Estimation through Adaptive Sampling", in *Proceedings of SIGMOD Conference*, 1990, pp. 1-11.
- [LoT00] K. Loney and M. Theriault, *ORACLE8i DBA Handbook*, McGraw-Hill, 2000.
- [LPK94] C. A. Galindo-Legaria, A. Pellenkoff and M. L. Kersten, "Fast, Randomized Join-Order Selection - Why Use Transformations?", in *Proceedings of Conference on Very Large Data Bases*, 94, pp. 85-95.
- [MaL96] L. F. Mackert and G. M. Lohman, "R* Optimizer: Validation and performance evaluation for distributed queries", in *Proceedings of Conference on Very Large Data Bases*, 1986, pp. 149-159.
- [MBK00] S. Manegold, P. A. Boncz and M. L. Kersten, "What Happens During A Join? Dissecting CPU and Memory Optimization Effects", in *Proceedings of Conference on Very Large Data Bases*, 2000, pp. 339-350.
- [MCS88] M. V. Mannino, P. Chu and T. Sager, "Statistical Profile Estimation in Database Systems", *ACM Computing Surveys*, Vol. 20, No. 3, September 1988.
- [MiE92] P. Mishra and M. H. Eich, "Join Processing in Relational Databases", *ACM Computing Surveys* 24(1), 1992, pp. 63-113.
- [MPK00] S. Manegold, J. Pellenkoff and M. L. Kersten, "A multi-query optimizer for Monet", *CWI Reports of Information Systems (INS)*, 2000.
- [OoC03] J. Oommen, and J. Chen, "The Averaged Rectangular Attribute Cardinality Map – A New Histogram Method for Sparse Attributes", submitted to *the 7th International Database Engineering and Application Symposium*, 2003.
- [OoR02] B. J. Oommen and L. G. Rueda, "The Efficiency of Histogram-like Techniques for Database Query Optimization", *The Computer Journal*, Vol. 45, No. 5, 2002, pp. 494-510.
- [OoT99] B. J. Oommen and M. Thiyagarajah, "Rectangular Attribute Cardinality Map: A New Histogram-like Technique for Query Optimization", *International Database Engineering and Applications Symposium*, 1999, pp. 3-15.

- [OoT00] B. J. Oommen and M. Thiagarajah, "Query Result Size Estimation Using the Trapezoidal Attribute Cardinality Map", *International Database Engineering and Applications Symposium*, 2000, pp. 236-242.
- [Ora02] "Query Optimization in Oracle9i", ORACLE Company, February 2002.
- [Orb02] "Oracle Database Performance Tuning Guide and Reference", ORACLE Company, March 2002.
- [PIH96] V. Poosala, Y. Ioannidis, P. Haas, and E. Shekita, "Improved Histograms for Selectivity Estimation of Range Predicates", in *Proceedings of SIGMOD Conference*, May 1996, pp. 294-305.
- [PsC84] G. Piatetsky-Shapiro, and C. Connell, "Accurate Estimation of the Number of Tuples Satisfying a Condition", in *Proceedings of ACM SIGMOD Conference*, 1984, pp. 256-276.
- [RPN00] K. Ramasamy, J. M. Patel, J. F. Naughton and R. Kaushik, "Set Containment Joins: The Good, The Bad and The Ugly", in *Proceedings of Conference on Very Large Data Bases*, 2000, pp. 351-362.
- [SAC79] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie and T. G. Price, "Access Path Selection in a Relational Database Management System", in *Proceedings of SIGMOD Conference*, 1979, pp. 23-34.
- [SAR99] V. Singh, K. Alsabti, and S. Ranka, "Structure And Method For Efficient Parallel High-dimensional Similarity Join", Patent Number: 5,987,468, Nov 16, 1999.
- [SHP96] P. Seshadri, J. M. Hellerstein, H. Pirahesh, T. Y. C. Leung, R. Ramakrishnan, D. Srivastava, P. J. Stuckey and S. Sudarshan, "Cost-Based Optimization for Magic: Algebra and Implementation", in *Proceedings of SIGMOD Conference*, 1996, pp. 435-446.
- [ShZ90] A. Shibamia and M. Zimowaki, "Data Base Optimizer Using Most Frequency Values Statistics", Patent Number: 4,956,774, Sep 11, 1990.
- [SLM01] M. Stillger, G. M. Lohman, V. Markl and M. Kandil, "LEO - DB2's LEarning Optimizer", in *Proceedings of Conference on Very Large Data Bases*, 2001, pp. 19-28.
- [StF 95] M. Stillger and J. C. Freytag, "Testing the Quality of a Query Optimizer", *Data Engineering Bulletin* 18(3), 1995, pp. 41-48.

- [Thi99] Murali Thiagarajah, "Attribute Cardinality Maps: New Query Result Size Estimation Techniques for Database Systems", Ph.D. Thesis, School of Computer Science, Carleton Univ., Ottawa, Canada, May 1999.
- [VaM96] B. Vance and D. Maier, "Rapid Bushy Join-order Optimization with Cartesian Products", in *Proceedings of SIGMOD Conference*, 1996, pp. 35-46.
- [WaG00] F. Waas and C. Galindo-Legaria, "Counting, enumerating, and sampling of execution plans in a cost-based query optimizer", in *Proceedings of SIGMOD Conference*, 2000, pp. 499-509.
- [WaP96] F. Waas and J. Pellenkoft, "Probabilistic bottom-up join order selection --- breaking the curse of NP-completeness", *CWI Reports of Information Systems (INS)* 1999.
- [Wu99] M. C. Wu, "Query optimization for selections using bitmaps", in *Proceedings of SIGMOD Conference*, 1999, pp. 227-238.
- [WRW99] C. W. Welch, J. M. Rozmus, J. L. Whiteman, M. Negin, and W. J. Herd, "System And Methods for Preventing Fraud In Retail Environments, Including The Detection Of Empty And Non-empty Shopping Carts", Patent Number: 5,883,968, Mar 16, 1999.
- [ZCL00] M. Zaharioudakis, R. Cochrane, G. Lapis, H. Pirahesh and M. Urata, "Answering complex SQL queries using automatic summary tables", in *Proceedings of SIGMOD Conference*, 2000, pp. 105-116.
- [ZMT00] D. Zhang, A. Markowetz, V. Tsotras, D. Gunopulos and B. Seeger, "Efficient computation of temporal aggregates with range predicates", in *Proceedings of the Nineteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, 2000, pp. 237-245.