

Expectations Associated with Compound  
Selection and Join Operations

By

Don-lin Yang

Computer Science Report #RM-85-0~~X~~3

July 5, 1985

**Expectations Associated with Compound  
Selection and Join Operations**

---

**A Dissertation**

**Presented to**

**the Faculty of the School of Engineering and Applied Science**

**University of Virginia**

---

**In Partial Fulfillment**

**of the Requirements for the Degree**

**Doctor of Philosophy ( Computer Science )**

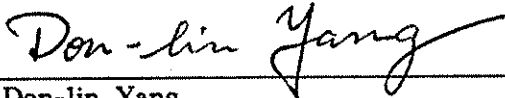
**by**

**Don-lin Yang**

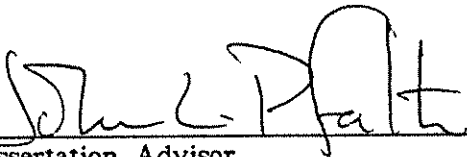
**May 1985**


## APPROVAL SHEET

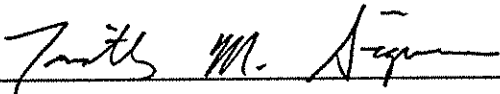
This dissertation is submitted in partial fulfillment of the  
requirements for the degree of  
Doctor of Philosophy ( Computer Science )

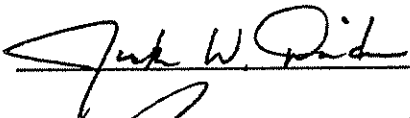
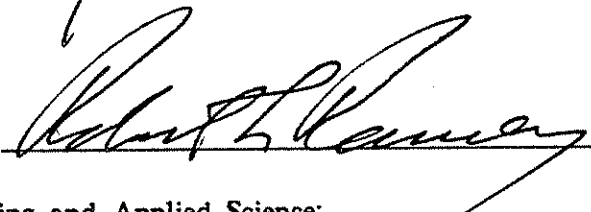
  
Don-lin Yang

This dissertation has been read and approved by the Examining  
Committee:

  
Dissertation Advisor

  
Committee Chairman

  
\_\_\_\_\_

  
  
\_\_\_\_\_

Accepted for the School of Engineering and Applied Science:

  
Dean, School of Engineering  
and Applied Science

May 1985

© Copyright by

Don-lin Yang

All rights reserved

May, 1985

## Abstract

The primary cost in processing relational database queries is the cost of joining two or more relations. In order to develop more efficient join algorithms or to optimize query strategies at run time, we must be able to accurately compute the expected size of a join relation and its expected cost. Size of a relation is expressed in terms of the number of tuples. Cost is measured by the number of secondary storage (disk) blocks retrieved in the join operation. We will also study the evaluation of selection operations, since they are normally included in general query processing.

In this thesis we conduct a comprehensive investigation on the cost of performing join and selection operations and the expected size of resultant relations. We present some approaches to evaluate the behavior of the natural join operation along with optional selection formulas. It is clear that the attribute value distribution has a great effect on the outcome of join and selection operations. Therefore, statistical parameters are employed in the forming of mathematical formulas for the size of resultant relations. Formulas are developed independent of file structure and access path, then they are applied to the specific systems. Error analysis is also used to find the maximum estimation errors. Further we show some algorithms to minimize the maximum errors and various methods to control statistical parameters in a dynamic environment.

Our goal is to develop general models for improving the design and analysis of query processing algorithms. With these tools one can begin the formidable task of query optimization and achieve efficient resource allocation (e.g. main memory, temporary storage, etc.). With that in mind, we emphasize the following three aspects in this research. First, we analyze the requirements and assumptions to be used in the mathematical models. Second, precise formulas are developed and proved analytically under realistic assumptions. Last, we perform empirical studies in specific systems.

## Acknowledgments

I would like to express my sincere thanks to my advisor, Professor John L. Pfaltz, for his time, patience, and guidance during the course of this research.

I am also grateful to the other members of my committee: Professors Alan P. Batson, Robert L. Ramey, Timothy M. Sigmon, and Jack W. Davidson for their direction throughout my graduate studies. Many thanks are extended to my friend Dennis Blankenbeckler for his encouragement.

My deepest appreciation is due to my wife, Mei-chiao and two lovely daughters Sunny and Cindy. Their support and understanding made this work possible and made it worthwhile. Finally, I would like to give my gratitude to my parents for their endless support over the years.

This research was supported in part by NSF Grant MCS-83-02654.

## Table of Contents

<i>Chapter</i>	<i>page</i>
1. Introduction .....	1
1.1 Relational terminology .....	3
1.2 Role of join in relational queries .....	11
1.3 Previous work .....	13
2. Expected Size of the Natural Join .....	17
2.1 Exact size of the natural join .....	17
2.2 Expected size of the natural join .....	23
3. Error Analysis of the Expected Join Size .....	27
3.1 Estimation errors .....	27
3.2 Maximum estimation errors .....	28
3.3 Minimization of the maximum estimation errors .....	30
4. Applications in a Dynamic Database .....	49
4.1 Maintaining $c$ and $\delta$ in a dynamic environment .....	50
4.2 Frequency tables .....	54
4.3 Random sampling approaches .....	58
4.4 Estimating standard deviations .....	61
5. Expected Size of Selection .....	65
5.1 Probability of satisfying a selection function .....	65
5.1.1 Simple probabilistic approaches .....	66
5.1.2 Attribute distribution tables .....	67
5.1.3 Grouped frequency schemes .....	69
5.1.4 Random sampling methods .....	72
5.2 Compound selections .....	76
5.3 Expected block accesses of selection .....	86

6. Access Cost of Queries .....	89
6.1 Issues in cost analysis .....	89
6.2 Partial-match retrieval .....	92
6.3 Indexed Descriptor Access Method .....	93
6.4 Access cost of selection on IDAM files .....	94
6.5 Access cost of a query involving join and selection .....	99
7. Conclusions .....	105
7.1 Major contributions .....	105
7.2 Summary .....	106
7.3 Further work .....	110
References .....	112



## List of Tables and Figures

<i>Table / Figure</i>	<i>page</i>
Table 1.1.1 A relation $r$ with $R = \{\text{Supplier, Part}\}$ .....	5
Table 1.1.2 A relation $s$ and its projections .....	6
Table 1.1.3 A relation after the selection ( $\text{part} = p_2$ ) .....	7
Table 1.1.4 Natural join .....	8
Table 1.1.5 Lossy join .....	9
Table 1.1.6 Lossless decomposition .....	10
Table 1.2.1 Cartesian product .....	12
Table 2.1.1 Attribute value distributions .....	18
Table 2.1.2 Results of the natural-join .....	19
Table 2.1.3 Results of join size parameters .....	21
Table 2.1.4 Results of the less-join .....	22
Table 3.3.1 An A-partition .....	35
Table 3.3.2 An equal-sized partition ("ER") .....	41
Table 3.3.3 The "EP" and "ED" partitions .....	47
Table 4.1.1 An A-partition of a skewed distribution .....	54
Table 4.2.1 A frequency table .....	56
Table 4.2.2 A grouped frequency table .....	57
Figure 4.3.1 A cumulative frequency distribution .....	60
Table 5.1.2.1 An attribute distribution table .....	68
Figure 5.1.3.1 A histogram .....	70
Figure 5.1.3.2 An equal-height histogram .....	71

Figure 5.1.3.3 Distribution steps .....	72
Table 5.1.4.1 Sample sizes .....	74
Table 5.1.4.2 Distribution step values .....	75
Table 5.1.4.3 Maximum deviation $\leq 0.05$ .....	75
Figure 5.2.1 Independent selection terms .....	80
Table 5.3.1 Block accesses .....	87
Table 6.4.1 The access cost of a compound selection .....	98
Table 6.5.1 The access cost of a bitwise join .....	104

## List of Theorems and Corollaries

<i>Theorem/Corollary</i>	<i>page</i>
Theorem 2.1.1 Exact join size .....	18
Corollary 2.1.1 Exact size of a join when the join attribute value distribution of either relation is perfectly uniform .....	22
Corollary 2.2.1 Expected join size .....	23
Corollary 2.2.2 Expected size of a join when source relations have unequally sized join domains .....	25
Theorem 2.2.1 Expected size of a join with selections .....	26
Theorem 3.3.1 To estimate join size, $\text{MaxErr}(\text{Unpartitioned}) \geq \text{MaxErr}$ (A-partition) .....	32
Corollary 3.3.1 To estimate join size, $\text{MaxErr}(\text{Unpartitioned}) \geq \omega +$ $\text{MaxErr}(\text{A-partition})$ .....	36
Corollary 3.3.2 To estimate join size, $\text{MaxErr}(\text{Unpartitioned}) \geq k *$ $\text{MaxErr}(\text{ER method})$ .....	39
Corollary 3.3.3 To estimate join size, $\text{MaxErr}(\text{ED method}) \geq \text{MaxErr}$ (EP method) .....	42
Theorem 5.2.1 Expected size of a compound selection .....	82
Corollary 5.2.1 Expected size of a compound selection when common terms exist .....	84
Corollary 5.2.2 Expected size of a compound selection with independent formulae .....	85
Theorem 6.4.1 Access cost of a compound selection .....	95

Corollary 6.4.1	Access cost of a compound selection with no common term .....	97
Corollary 6.5.1	Access cost of a join with selections .....	101

## List of Symbols

$A, B, ..$	Attributes
$A_k, B_l, ..$	Attributes with subscripts
$a, b, ..$	Attribute values, $a = A(t), b = B(t), ..$
$a_k, b_l, ..$	Attribute values, $a_k = A_k(t), b_l = B_l(t), ..$
$BF$	Blocking factor
$B_i$	Number of blocks in level $i$ file
$c$	Correlation coefficient
$D_A$	Domain of attribute $A$
$D_Q$	Query descriptor
$D_R$	Record descriptor
$D_\beta$	Block descriptor
$\bar{d}$	Expected number of tuples with distinct join attribute values in each block of relation $\sigma_\rho(r)$
$exp( \dots )$	Expectation
$F$	Set of functional dependencies
$G(\sigma_k(r))$	Ratio between $B_0(\sigma_k(r))$ and $B_0(r)$
$g(\sigma_k(r))$	Ratio between $ \sigma_k(r) $ and $ r $
$n_{i+1}$	Number of tuples in level $(i+1)$ file
$\bar{\pi}$	Expected number of tuples per block in $\sigma_\rho(r)$

## List of Symbols (cont.)

$p(i+1, l) = \frac{\overline{s}_{l, i+1}}{w_l}$	Probability that field $l$ of query descriptor matches field $l$ of any descriptor at level $(i+1)$
$Q$	Query
$R$	Relational scheme, $R = \{A_1, \dots, A_n\}$
$R_i$	Subscheme, $R_i \subseteq R$
$r, s$	Relations with schemes $R$ and $S$ respectively
$r \cup s$	Union of relations $r$ and $s$
$r \cap s$	Intersection of relations $r$ and $s$
$r - s$	Complement of relations $r$ and $s$
$r \div s$	Quotient of relations $r$ and $s$
$r \bowtie s$	Natural join of relations $r$ and $s$
$\Pi_S(r)$	Projection of relation $r$ on attribute set $S$
$\sigma_\rho(r)$	Selection of relation $r$ with selection formula $\rho$
$\overline{s}_{l, i+1}$	Expected number of 1-bits in field $l$ of a descriptor at level $(i+1)$
$t, u, v, \dots$	Tuples
$U$	Universe (set of all recognized attributes)
$\hat{u}$	Universal relation (or often just $r$ )
$w = \sum_{i=0}^k w_i$	Descriptor width

## List of Symbols (cont.)

$X, Y, Z, ..$	Attribute sets, $X = \{..., A_k, ...\}$
$X \rightarrow Y$	$Y$ is functionally dependent on $X$
$\alpha = \sum_{i=0}^h \alpha_i$	Access cost (expected number of disk accesses)
$\mu_i(a_k)$	Number of tuples with join attribute value $a_k$ in relation $r_i$
$\delta$	Standard deviation
$\emptyset$	Empty set
$\wedge$	Logical operator AND
$\vee$	Logical operator OR
$\exists$	There exists
$\forall$	For all
$  \dots  $	Size

# CHAPTER 1

## Introduction

One of the major advantages of the relational database is its simplicity and uniformity of data representation. In the relational model, data is viewed as being stored in relations (or tables) with attribute names (e.g., Name, Address, Tel#) as column headings describing the format for each row (or tuple). Users need not use indexes or pointers to retrieve the data from databases. The relational view introduced by Codd [Cod70,71,71a] succeeds in removing the need for physical navigation.

However, some operations are required to link different relations together and select the desired entries to answer queries. One of the most important operators is **join**, which is a generalized composition. Two or more relations can be combined by join operations with respect to joining (common) attributes to yield a new relation. Useful information can be derived from this correlated relation. The other frequently used operation is **selection**, which restricts the resultant entries to meet certain criteria, such as the restriction of (subject = "database"). In the following sections a detailed discussion on the roles of join and selection in relational queries and their formal definitions will be presented.

The primary cost of processing general database queries depends heavily on the size (cardinality) of the join of two or more relations [Ros81]. In order to develop and evaluate more efficient join algorithms, we must be able to accurately compute the expected size and cost of a join. Expected cost is normally measured in terms of the expected number of secondary storage (disk) blocks retrieved in the join operation. To optimize query processing strategies at run time an intelligent system must know the expected size of various resultant relations. For example, the



performance of processing the join of three relations  $r_1 \bowtie r_2 \bowtie r_3$  is generally evaluated by the size of generated intermediate relations and the total number of disk blocks accessed. Three different sequences of join operations may be considered:

$$(r_1 \bowtie r_2) \bowtie r_3, \text{ or } (r_1 \bowtie r_3) \bowtie r_2, \text{ or } r_1 \bowtie (r_2 \bowtie r_3).$$

Let  $r_{ij} = r_i \bowtie r_j$ . To assess the best choice without actually performing join operations, we must be able to compute the expected size and cost of joining each pair of relations  $r_i \bowtie r_j$  and  $r_{ij} \bowtie r_k$ , where  $i, j, k = 1, 2, 3$  and  $i \neq j \neq k$ . With this data we can determine the best join sequence by comparing the sizes of intermediate relations and the total costs for each of the three alternatives. We also need to consider the evaluation of selection operations, since they are normally included in general database queries.

In this research we investigate in depth the cost of performing selection and join operations on the relational databases and the expected size of the resultant relations. Our goal is to develop general models for improving the design and analysis of query processing algorithms. With these tools one can begin the formidable task of query optimization and achieve efficient resource allocation (e.g., main memory, temporary storage, etc.).

The next three sections describe the background of our study. In Chapter 2, we first examine the behavior of the distribution of join attribute values between two join relations. Then, two statistical variables are used to derive the formulae for the size of a join. Chapter 3 deals with the analysis of (maximum) estimation errors and algorithms to minimize them. In Chapter 4, applications in a dynamic database system are addressed regarding the estimation of a join size. Chapter 5 discusses the expected sizes of the selection and compound selection. The access cost of performing join and selection operations is the main topic in Chapter 6. A special index descriptor access method, IDAM [Pfa80], file is used in the study for

the development of access cost formulas. A generalized version of the cost formula for general file systems is our ultimate result.

Finally, we conclude in Chapter 7 with a summary of our study and some future research topics.

### 1.1. Relational terminology

The setting of our study is the relational database model. Unfortunately, there is a confusion of terminology and notation. Part of the problem stems from Codd's [Cod70] original formulation in terms of **tables**. Other problems arise from inconsistent use of the terms **tuples**, **schemes**, and **relations** in different contexts. In the following pages we review the relational model using our own terminology.

We use the term **tuple** (or **record**) as the basic element of a relational database. It is primitive and is denoted by lower case letters  $t$ ,  $u$ , or  $v$ . An **attribute** is a single valued (partial) function defined on tuples. It is a partial function because it need not be defined on all tuples. A tuple may have one or more attributes defined. Attributes are denoted by upper case letters from the beginning of the alphabet, such as  $A$  and  $B$  (subscripts can be used to distinguish different attribute functions, e.g.,  $A_k$  and  $B_l$ ).

We use corresponding lower case letters to denote **attribute values**. For example,  $a_k = A_k(t)$  and  $b = B(t)$  denote the values of attributes  $A_k$  and  $B$  on tuple  $t$ . The values that an attribute function  $A$  may assume are called its **domain**, denoted  $D_A$ . If  $A_1, \dots, A_n$  are defined on a tuple  $t$ , then by convention the tuple can be represented (not defined) by its functional values, viz.

$$t = (A_1(t), A_2(t), \dots, A_n(t)) = (a_1, a_2, \dots, a_n)$$

Note that  $t$  is a **set** of attribute values.

We let upper case letters,  $X, Y, Z$  denote sets of attributes. As a notational convention, if

$$X = \{..., A_k, ...\}$$

we let  $X(t)$  denote  $\{..., A_k(t), ...\}$  and say

$$X(t) = X(u) \text{ if for all } A_k \in X, A_k(t) = A_k(u)$$

$$X(t) \neq X(u) \text{ if there exists } A_k \in X, A_k(t) \neq A_k(u).$$

We define a relation **scheme**  $R$  to be a set of attributes, viz.

$$R = \{A_1, \dots, A_n\}$$

A scheme  $R_i$  is said to be a **subscheme** of  $R$  if  $R_i \subseteq R$ . The set of all recognized attributes is called the **universe**,  $U$ .

A **relation**  $r_i$  with scheme  $R_i$  is a set of tuples  $r_i = \{t\}$  such that  $A_k(t)$  is defined if and only if  $A_k \in R_i$ . If  $t$  is any tuple in a relation  $r_i$ , then  $t$  is a tuple with scheme  $R_i$ , that is with precisely those attribute functions defined on it. Usually the scheme of a tuple is clear implicitly. Whenever there is a possibility of ambiguity, we will use the notation  $t$  **with**  $R_i$ .

The example in Table 1.1.1 illustrates a relation  $r$  with two attributes *supplier* and *part*. We can see that *part* "p1" is supplied by *supplier* "sp1" and *part* "p2" is supplied by both *suppliers* "sp1" and "sp2". Here

$$R = \{A_1, A_2\} = \{\text{supplier}, \text{part}\} \text{ and}$$

$$r = \{t, u, v\} = \{(sp1, p1), (sp1, p2), (sp2, p2)\}.$$

We note that  $sp1 = A_1(t)$  and  $p1 = A_2(t)$  are attribute values of the first tuple  $t$ . And  $u = (sp1, p2), v = (sp2, p2)$  are the remaining tuples in the relation  $r$ . The domains of  $A_1$  and  $A_2$  are  $D_{A_1} = (\text{all valid supplier names})$  and  $D_{A_2} = (\text{all valid part names})$ .

---

Supplier	Part
sp1	p1
sp1	p2
sp2	p2

---

Table 1.1.1 a relation  $r$  with  $R = \{\text{Supplier}, \text{Part}\}$

---

Since relations are sets, we can apply the traditional set operations to them. However, the operations discussed here are the most common and important ones used in the relational database : projection, selection, and join.

The first operation is **projection**. Since every user (or application) may have a different view or interest in a common database, it is desirable to restrict a relation to some smaller set of attributes (or information). To do this, we select a set of attributes (or subscheme) and their corresponding attribute values from one relation to form a new relation with all duplicate tuples removed.

For example, Table 1.1.2 (b) and (c) show two relations projected from relation  $s$  (a) with scheme  $S = \{B_1, B_2\} = \{\text{part}, \text{project}\}$ . We use  $\Pi$  to denote the projection operator and define its operation as the following :

$$\Pi_{S_i}(s) = \{t \text{ with } S_i | (\exists u \in s)[S_i(t) = S_i(u)]\}$$

where  $s$  is a relation with scheme  $S$  and subscheme  $S_i \subseteq S$ . In Table 1.1.2 (b), we project relation  $s$  onto attribute  $B_1$ , i.e., *part* to form a new relation  $\Pi_{B_1}(s)$  which contains all the three parts in the original relation  $s$ . Attribute  $B_2$  (*project*) is specified in (c) to generate the second projection  $\Pi_{B_2}(s)$  from relation  $s$ .

---

(a)		(b)	(c)
Part	Project	Part	Project
p1	pj1	p1	pj1
p1	pj2	p2	pj2
p2	pj1		pj3
p2	pj2		
p2	pj3		

Table 1.1.2 relations  $s, \Pi_{B_1}(s), \Pi_{B_2}(s)$

---

It is also useful to be able to select a subset of tuples from a relation. For example, in responding to a query concerning some specific attribute values, a new relation can be formed by the tuples having these attribute values. In this case, the new relation has the original relation scheme. We call this operation **selection** and denote its operator by  $\sigma$ . Let  $\rho$  be a well formed formula in the predicate calculus involving one or more attributes, then we define the selection operation as follows :

$$\sigma_{\rho}(r) = \{t \text{ with } R \mid t \in r \text{ and } \rho(t)\}$$

By  $\rho(t)$ , we mean that the selection formula  $\rho$  is true for the tuple  $t$  selected from relation  $r$ . Selection formulae can be formed by connecting comparison terms on the specified attributes with logical operators  $\wedge$  (AND),  $\vee$  (OR), and  $\neg$  (NOT), e.g.,  $(A > 3) \wedge (B > C)$ . The comparison operators  $<, =, >, \leq, \neq$  and  $\geq$  can be used provided the domain  $D_A$  is ordered.

Table 1.1.3 displays a subset of tuples from relation  $s$  of Table 1.1.2 (a). In this new relation  $\sigma_{B_1="p2"}(s)$ , all the values of attribute  $part$  are equal to "p2", i.e., the selection formula is  $B_1 = "p2"$ . In the result of this selection operation it is easy to see that part "p2" is used in all three projects of relation  $s$ .

The **join** operation is used to combine two relations. It comes in various forms. A completely unrestricted join is the same as a **Cartesian product** of two relations. The most commonly used join operation is called a **natural join** and denoted by  $\bowtie$ . We define the natural join of two relations  $r$  and  $s$  with schemes  $R$  and  $S$  respectively as follows :

$$r \bowtie s = \{t \text{ with } R \cup S \mid (\exists u \in r) (\exists v \in s) [R(t) = R(u) \text{ and } S(t) = S(v)]\}$$

If  $R \cap S = \emptyset$ , then  $r \bowtie s = r \times s$  (i.e., Cartesian product).

If  $R = S$ , then  $r \bowtie s = r \cap s$  (i.e., intersection).

To perform the join operation, the equivalent entries in the matching attributes are selected from both relations and all possible combinations of tuple pairs are created. For example, Table 1.1.4 shows a natural join of relation  $r$  and  $s$  given in Table 1.1.1 and 1.1.2 (a) respectively. Note that only one occurrence of attribute  $part$  remains in the new scheme  $R \cup S = \{supplier, part, project\}$ .

---

Part	Project
p2	pj1
p2	pj2
p2	pj3

---

Table 1.1.3  $\sigma_{B_1="p2"}(s)$

---

$r$		$s$		$r \bowtie s$		
Supplier	Part	Part	Project	Supplier	Part	Project
sp1	p1	p1	pj1	sp1	p1	pj1
sp1	p2	p1	pj2	sp1	p1	pj2
sp2	p2	p2	pj1	sp1	p2	pj1
		p2	pj2	sp1	p2	pj2
		p2	pj3	sp1	p2	pj3
				sp2	p2	pj1
				sp2	p2	pj2
				sp2	p2	pj3

Table 1.1.4 Natural join

Suppose we are given the rightmost relation  $rel = r \bowtie s$  in Table 1.1.4 and project the relation on attribute sets  $R_1 = \{supplier, part\}$  and  $S_1 = \{part, project\}$  to get two new relations  $r_1$  and  $s_1$  respectively. It is obvious that  $r_1 = r$ ,  $s_1 = s$  and  $rel = r_1 \bowtie s_1$ . This illustrates an instance of a **lossless join** because the join of projected relations contains no additional spurious data. However, the decomposition in the example of Table 1.1.4 is not lossless (called **lossy**) because for some instances of the relation  $rel$ ,  $r_1 \bowtie s_1 \neq rel$ . In Table 1.1.5 we show an instance of a lossy join where  $r_1 = \Pi_{R_1}(rel)$ ,  $s_1 = \Pi_{S_1}(rel)$ , and  $r_1 \bowtie s_1 \neq rel$ . Since the lossless (faithful) property is a very important feature of natural join in the decomposition of database relations, we will give a formal definition in the following paragraphs.

Let  $R$  be any relational scheme. A set of subschemes  $R_1, R_2, \dots, R_n$  is called a **decomposition** of  $R$  if each  $R_i \subseteq R$  and  $R = \bigcup_{i=1}^n R_i$ . Note that a decomposition of  $R$  is similar to a partition of the set, except that the subschemes  $R_i$  need not be disjoint subsets, and indeed in general are not disjoint.

<i>rel</i>			<i>r</i> <sub>1</sub>		<i>s</i> <sub>1</sub>		<i>r</i> <sub>1</sub> ⋈ <i>s</i> <sub>1</sub>		
Supplier	Part	Project	Supplier	Part	Part	Project	Supplier	Part	Project
sp1	p1	pj1	sp1	p1	p1	pj1	sp1	p1	pj1
sp1	p1	pj2	sp2	p1	p1	pj2	sp1	p1	pj2
sp2	p1	pj1					sp2	p1	pj1
							sp2	p1	pj2

Table 1.1.5 Lossy join

Let  $X, Y$  be non-empty sets of attributes.  $X$  **functionally determines**  $Y$ , (or  $Y$  is **functionally dependent** on  $X$ ) denoted  $X \rightarrow Y$ , if for all tuples  $t, u$

$$X(t) = X(u) \text{ implies } Y(t) = Y(u)$$

or equivalently

$$Y(t) \neq Y(u) \text{ implies } X(t) \neq X(u).$$

Let  $F$  be a set of such functional dependencies. A decomposition  $R_1, R_2, \dots, R_n$  is said to have the **lossless join property** if for any relation  $r$  with scheme  $R = \bigcup_{i=1}^n R_i$  that satisfies the set of functional dependencies  $F$ ,  $\bigjoin_{i=1}^n \Pi_{R_i}(r) = r$ . This is to say that we can recover relation  $r$  by taking the natural join of its subschemes  $R_1, R_2, \dots, R_n$ . In other words, we want to be able to find a database scheme  $\{R_1, \dots, R_n\}$  that faithfully represents "every instance" of the universe.

Again consider the same decomposition in Table 1.1.5 and let  $F = \{Part \rightarrow Supplier\}$ . With the addition of a functional dependency we can show that the decomposition is lossless. It has been proven in [Ull82, p.230] that the decomposition of  $R$  into  $(R_1, R_2)$  is lossless with respect to  $F$  if and only if  $(R_1 \cap R_2) \rightarrow (R_1 - R_2)$ . We know that  $R_1 \cap R_2 = (supplier, part) \cap (part,$



$project) = (part)$  and  $R_1 - R_2 = (supplier, part) - (part, project) = (supplier)$ . Obviously, the decomposition is lossless since  $(part \rightarrow supplier) \in F$ . Table 1.1.6 displays an instance of lossless decomposition where  $r_1 \bowtie s_1 = rel$ . One of the important objectives of a faithful decomposition is to eliminate some of the problems, such as the data redundancy and inconsistency of relations in the design and maintenance of a database.

Finally, we shall discuss a more general form of join commonly called  $\theta$ -join [Cod71]. A  $\theta$ -join is determined by a comparison operator  $\theta$  performed between the values of specified attributes in two source relations, where  $\theta$  can be any of the comparison operators  $<$ ,  $=$ ,  $>$ ,  $\leq$ ,  $\neq$ , and  $\geq$ . If they satisfy the relationship specified, then the corresponding tuples of the relations are combined to form a new relation.

Let  $R_i$  and  $S_i$  be subschemes of  $R$  and  $S$  respectively, then  $\theta$ -join of relations  $r$  and  $s$  is denoted by  $r \bowtie_{R_i \theta S_i} s$  with new scheme  $R \cup S$ . For  $\theta$ -join to exist, it is required that  $R_i(u) \theta S_i(v)$  is either true or false (not undefined) for all  $u \in r$  and  $v \in s$ . If  $\theta$  is equal to "=", then it is called **equi-join**. The difference between equi-join and natural join is that the join attributes are explicitly specified and one of the redundant attributes is not removed from  $R \cup S$  in the equi-join. For

<i>rel</i>			<i>r</i> <sub>1</sub>		<i>s</i> <sub>1</sub>		<i>r</i> <sub>1</sub> $\bowtie$ <i>s</i> <sub>1</sub>		
Supplier	Part	Project	Supplier	Part	Part	Project	Supplier	Part	Project
sp1	p1	pj1	sp1	p1	p1	pj1	sp1	p1	pj1
sp1	p1	pj2	sp2	p2	p1	pj2	sp1	p1	pj2
sp2	p2	pj1			p2	pj1	sp2	p2	pj1

Table 1.1.6 Lossless decomposition

example, if we change the natural join operator in Table 1.1.4 to equi-join, then the resultant relation  $r \bowtie_{A_2=B_1} s$  would have two attributes that are exactly the same, i.e., each tuple has two *part* attributes with the same attribute values.

## 1.2. Role of join in relational queries

In addition to its important role in the decomposition of relations, the natural join is also frequently used in general database queries. Since relations are usually decomposed into normalized forms (e.g., 3rd Normal Form), it is common to combine two or more relations in a query processing. Conceptually, the easiest method is to use the Cartesian product. For example, Table 1.2.1 shows a Cartesian product  $r \times s$  for relations  $r$  and  $s$  given in the last section. The resultant relation has  $|r| \cdot |s| = 3 \cdot 5 = 15$  tuples.

As we can see, this operation involves all the tuples in both relations. Forming a Cartesian product is computationally expensive and the derived relation takes a huge storage space if large relations are used. Fortunately, most queries have restriction criteria that select only a subset of the resultant relation. For example, given the relations  $r$  and  $s$  in Table 1.2.1, a query may look like the following :

"Which suppliers provide the parts for project 3 ?"

We can get an answer from the Cartesian product  $r \times s$  by applying the selection formula (*project* = "pj3") and projecting its resultant relation on the attribute *supplier*. The result is (*sp2*) :

$$\Pi_{A_1}(\sigma_{A_2=B_1 \wedge B_2="pj3"}(r \times s)) = (sp2)$$

where the schemes of  $r$  and  $s$  are  $R = \{Supplier, Part\}$  or  $\{A_1, A_2\}$  and  $S = \{Part, Project\}$  or  $\{B_1, B_2\}$  respectively. Since creation of the Cartesian product is expensive in terms of processing time and storage space, we should modify the

$r$		$s$		$r \times s$			
Supplier ( $A_1$ )	Part ( $A_2$ )	Part ( $B_1$ )	Project ( $B_2$ )	Supplier ( $A_1$ )	Part ( $A_2$ )	Part ( $B_1$ )	Project ( $B_2$ )
sp1	p1	p1	pj1	sp1	p1	p1	pj1
sp1	p2	p1	pj2	sp1	p1	p1	pj2
sp2	p2	p2	pj1	sp1	p1	p2	pj1
		p2	pj2	sp1	p1	p2	pj2
		p2	pj3	sp1	p1	p2	pj3
				sp1	p2	p1	pj1
				sp1	p2	p1	pj2
				sp1	p2	p2	pj1
				sp1	p2	p2	pj2
				sp1	p2	p2	pj3
				sp2	p2	p1	pj1
				sp2	p2	p1	pj2
				sp2	p2	p2	pj1
				sp2	p2	p2	pj2
				sp2	p2	p2	pj3

Table 1.2.1 Cartesian product

expression to improve the performance. Selection operations are commutative [Ull82, p.276], so that the expression can be rewritten as

$$\Pi_{A_1}(\sigma_{B_2="pj3"}(\sigma_{A_2=B_1}(r \times s))).$$

This enables us to combine the Cartesian product  $r \times s$  and selection  $\sigma_{A_2=B_1}$  into a join  $r \bowtie s$ . The resultant relation is much smaller (eight tuples created) as shown in the last section, since we only create those tuples satisfying the selection criterion ( $A_2=B_1$ ). Therefore, the new expression becomes

$$\Pi_{A_1}(\sigma_{B_2="pj3"}(r \bowtie s)).$$

Furthermore, if we take the selection criterion ( $B_2 = "pj3"$ ) inside the join operation, then it is obvious that the resultant relation would become even smaller. In our example, only one out of five tuples in relation  $s$  needs to be joined. The

final expression becomes the following :

$$\Pi_{A_1}(r \bowtie \sigma_{B_2 = "p_j 3"}(s)).$$

The above discussed process is a very typical example of query processing. Joins are much more frequently used than Cartesian products in general queries. We also showed one way of rephrasing the query expression to optimize the process. Especially, note the improvement one can make when a selection operation is performed before a join.

So far, we have presented the use of joins in general database queries, and shown the improvement possible whenever a Cartesian product followed by a selection is replaced by a join operation. Rearranging the order of relational operators, especially performing selections as early as possible, is one of the strategies used in query optimization. Another subject of optimization is to take advantage of the structure of relations and their access paths in performing the operations. We will see a few examples in the next section when we examine some of the work that has been done in the past.

### 1.3. Previous work

Little work has been done on the analysis of the join operation. In [Got75] Gotlieb considers computing the join of relations using a theoretical analysis. Three algorithms for computing (performing) the natural join are presented and evaluated in terms of storage usage, computing time, and I/O time. Some specific storage structures and workspace are used to minimize the operation cost.

In his first algorithm, Gotlieb uses sequential search to perform the join operation on two sequentially organized relations  $r$  and  $s$  without any external index. The second algorithm indexes the join attribute of relation  $s$  with a collection of inverted lists. The first part of the algorithm constructs a second set

of inverted lists on the join attribute of relation  $r$  corresponding to the entries of the first set. It is the second part of the algorithm which performs the join operation by looping through all the actual values of the join attribute. The last algorithm considers the case when the join attributes of two relations are keys. Ordered key lists are used instead of inverted lists in the third algorithm.

Three cost evaluation factors are adopted in the paper. The first consideration is the usage of main memory in characters or bytes. Computing time is the second criterion and is measured by the number of logical comparisons performed. I/O time is the last factor and is denoted by the number of characters read from and written to the secondary storage where relations are stored. Disk seek time and latency time are also included in the cost expression.

As Gotlieb points out in the paper, a primary requirement for efficiency is to cut down I/O activity. In his algorithms, however, it is accomplished by using as large an internal workspace as possible, and carefully organizing external (index) files. Since the algorithm's performance depends primarily on the availability of main memory, his results are not generally comparable across machines. In addition, Gotlieb treats each block of data in the secondary storage as an independent unit. Therefore, each data block has the same access cost of (seek time + latency time + block size  $\cdot$  transfer time per character) regardless of the storage organization implemented. It is obvious that, for example, seek time need only be counted for the first block of a contiguous file in a sequential access. The computing time is not a good evaluation parameter either, since logical comparison is not the only operation involved in performing the algorithms, e.g., the creation of multilists, inverted lists, and sorted key lists. Furthermore, the accuracy of Gotlieb's algorithms can not be fully analyzed, since no experimental results are given in the paper.

On the other hand, as Blasgen and Eswarn point out in [Bla77], the cost of access to secondary storage is the most critical performance parameter. Therefore, they consider neither the cpu time nor the cost of virtual storage management in the cost expression. In the paper four methods are described to evaluate a fairly general query involving the operations of join, projection, and selection.

Comparisons are made of the four methods under some typical situations according to their physical organizations. One interesting result is that there are circumstances under which each method is the best. The conclusion is that the available access path, the physical clustering of relations and the characteristics of the query determine the best way of evaluating the query.

Both of these papers present the best algorithms for their specific system environment. However, neither of them has an expression for computing the expected size of the derived relation which can then be used to estimate the minimal amount of work needed to be performed in the query processing.

In [Ros81] Rosenthal derives the following expression for the expected size of an equi-join

$$exp(|r \bowtie_A s|) = \frac{|r| \cdot |s|}{|A|}$$

where  $|r|$ ,  $|s|$ , and  $|A|$  denote the sizes of relations  $r$  and  $s$ , plus the domain size of attribute  $A$  respectively. His proof technique requires two conditions to derive this result. First, the distribution of join attribute values in  $r$  and  $s$  must be independent and second, the distribution of join attribute values must also be uniform in either relation. These are stringent constraints that are seldom realized in practice. Moreover, they are really unneeded. A major result of this paper will be to show that this expression is still valid under much more general conditions.

Richard in [Ric81] also uses a probabilistic model to evaluate the size of relations derived from given relations for most of the relational algebra operators. Different assumptions about the inter and intra-relational independence are used in developing the expression for the expected size of an equi-join. The resultant expressions are so complex that they are difficult to use or to verify. In our study, both the exact and expected size of a join are presented under much weaker assumptions. We also develop general formulae for the access costs of join and selection with respect to the secondary storage accesses.

Of the above mentioned papers only Rosenthal provides a clear and generalized formula for the evaluation of derived relation size. He ignores access cost. The others provide expressions that are either empirically derived or theoretically analyzed. In all cases the results are based on specific implementation structures. In this study, we intend to develop a model which is sufficiently general to derive analytic results, yet detailed enough to yield accurate predictions in specific systems.

In addition to the mathematical need and interest, our primary goal is to analyze and develop generalized models which are applicable to real world systems. One imagines an intelligent database system, which can accurately compute the expected size of retrieved data and the costs of performing relational operations with respect to general queries. With that in mind, we emphasize the following three aspects in this research. First, we analyze the requirements and assumptions to be used in the mathematical models. Second, precise formulas are developed and proved analytically under realistic assumptions. Last, we perform empirical studies in specific systems.

## CHAPTER 2

### Expected Size of the Natural Join

#### 2.1. Exact size of the natural join

For notational convenience, we assume all joins are over a single attribute  $A_j$  which we call the **join attribute** and denote simply by  $A$ . Of course, one can join relations over several attributes, but there is no loss of generality in our analysis since they can always be regarded as a single attribute mapping into the Cartesian product of the original attribute domains.

As we know, the size of a joined relation depends on the distribution of join attribute values between two source relations. Different combinations of two join attribute value distributions will yield different sizes of joins ranging from minimum of 0 to maximum of  $|r_1| \cdot |r_2|$ . This variability is illustrated by Tables 2.1.1 and 2.1.2.

In Table 2.1.1 we assume two small relations  $r_1$  and  $r_2$  of size  $|r_1| = 6$  and  $|r_2| = 9$ . The join attribute  $A_j$  has an equally small domain, consisting of just the three integers  $\{1, 2, 3\}$ . we consider seven different possible distributions for the join attribute values of these two relations. The ascending sequence of join values in each set is not necessary. It only serves the purpose of convenience and easy understanding. In fact, they were generated randomly in our actual experiment.

The distributions of attribute values in the resulting natural join for each pair  $r_1, r_2$  in sets (1) - (7) are shown in Table 2.1.2. For example, in set (1) all six elements (tuples) of  $r_1$  have value 3 for the join attribute, and similarly for all nine elements of  $r_2$ . The resulting join for case (1) must be the Cartesian product  $r_1 \times r_2$  of the two relations with size  $|r_1| \cdot |r_2| = 6 \cdot 9 = 54$ ; as it is. In case (7)



---

Distributions of join attribute, A, values for relations $r_1$ and $r_2$													
(1)		(2)		(3)		(4)		(5)		(6)		(7)	
$r_1$	$r_2$	$r_1$	$r_2$	$r_1$	$r_2$	$r_1$	$r_2$	$r_1$	$r_2$	$r_1$	$r_2$	$r_1$	$r_2$
3	3	1	1	1	1	1	1	1	1	1	1	3	1
3	3	2	2	2	1	1	1	2	1	2	1	3	1
3	3	3	2	2	2	2	1	2	1	3	1	3	1
3	3	3	3	3	2	2	2	3	1	3	1	3	1
3	3	3	3	3	3	3	2	3	1	3	1	3	1
3	3	3	3	3	3	3	2	3	2	3	1	3	1
	3		3		3		3		2		2		1
	3		3		3		3		3		2		1
	3		3		3		3		3		3		1

---

Table 2.1.1 Attribute value distributions

the two relations have no common join attribute values, so readily  $|r_1 \bowtie r_2| = 0$ . Cases (2) through (6) simply illustrate other possible distributions between these extremes.

**Theorem 2.1.1** Let  $r_1$  and  $r_2$  be two relations with a common attribute  $A$ . Let  $c$  be the correlation coefficient between the distributions of join attribute values in  $r_1$  and  $r_2$ , and  $\delta_1, \delta_2$  be their corresponding standard deviations. Also let  $|A|$  denote the number of attribute values in  $A$ . Then

$$|r_1 \bowtie_A r_2| = \frac{|r_1| \cdot |r_2|}{|A|} + |A|c\delta_1\delta_2$$

**Proof :**

Let  $\{a_1, a_2, \dots, a_k, \dots, a_{|A|}\}$  be the join attribute values in  $A$ . Let  $\mu_1(a_k)$  and  $\mu_2(a_k)$  be the number of tuples having join attribute value  $a_k$  in  $r_1$  and  $r_2$ .

Distributions of attribute values in natural-join														
Join Domain Values	(1)		(2)		(3)		(4)		(5)		(6)		(7)	
	r <sub>1</sub>	r <sub>2</sub>	r <sub>1</sub>	r <sub>2</sub>	r <sub>1</sub>	r <sub>2</sub>	r <sub>1</sub>	r <sub>2</sub>	r <sub>1</sub>	r <sub>2</sub>	r <sub>1</sub>	r <sub>2</sub>	r <sub>1</sub>	r <sub>2</sub>
1	0	0	1	1	1	2	2	3	1	5	1	6	0	9
2	0	0	1	2	2	2	2	3	2	2	1	2	0	0
3	6	9	4	6	3	5	2	3	3	2	4	1	6	0
Join Size	54		27		21		18		15		12		0	

Table 2.1.2 Results of the natural-join

respectively. We know that

$$\begin{aligned}
 |r_1 \bowtie_A r_2| &= \sum_A (\mu_1(a_k) \cdot \mu_2(a_k)) = |A| \cdot \frac{\sum_A (\mu_1(a_k) \cdot \mu_2(a_k))}{|A|} \\
 &= |A| \cdot \exp(\mu_1 \cdot \mu_2)
 \end{aligned}$$

Let  $X = \mu_1$ ,  $Y = \mu_2$  and using the standard definition [Mey65]

$c = \frac{\exp((X - \exp(X)) \cdot (Y - \exp(Y)))}{\delta_x \delta_y}$ , we have the expression of the covariance

[Wal53, p.248]

$$\begin{aligned}
 c \delta_x \delta_y &= \exp((X - \exp(X)) \cdot (Y - \exp(Y))) \\
 &= \exp(XY - X \exp(Y) - \exp(X)Y + \exp(X)\exp(Y)) \\
 &= \exp(XY) - \exp(X \exp(Y)) - \exp(\exp(X)Y) + \exp(\exp(X)\exp(Y)) \\
 &= \exp(XY) - \exp(X)\exp(Y) - \exp(X)\exp(Y) + \exp(X)\exp(Y) \\
 &= \exp(XY) - \exp(X)\exp(Y)
 \end{aligned}$$

That is  $\exp(XY) = \exp(X)\exp(Y) + c \delta_1 \delta_2$ .

So that  $\exp(\mu_1 \cdot \mu_2) = \exp(\mu_1) \cdot \exp(\mu_2) + c \delta_1 \delta_2$ , then

$$\begin{aligned}
|r_1 \bowtie_A r_2| &= |A| \cdot (\exp(\mu_1) \cdot \exp(\mu_2) + c \delta_1 \delta_2) \\
&= |A| \cdot \left( \frac{\sum_A \mu_1(a_k)}{|A|} \cdot \frac{\sum_A \mu_2(a_k)}{|A|} + c \delta_1 \delta_2 \right) \\
&= |A| \cdot \left( \frac{|r_1|}{|A|} \cdot \frac{|r_2|}{|A|} + c \delta_1 \delta_2 \right) \\
&= |A| \cdot \frac{|r_1|}{|A|} \cdot \frac{|r_2|}{|A|} + |A| c \delta_1 \delta_2 \\
&= \frac{|r_1| \cdot |r_2|}{|A|} + |A| c \delta_1 \delta_2 \quad \square
\end{aligned}$$

To empirically verify the formula of Theorem 2.1.1, we extended the number of distribution sets from Table 2.1.2 to include all possible combinations. For  $r_1$  we can have a total number of  $1 + 2 + \dots + (|r_1| + 1) = 28$  different distributions and  $1 + 2 + \dots + (|r_2| + 1) = 55$  for  $r_2$ . Thus the total number of possible combinations is  $28 \times 55 = 1540$  distribution sets. It will be easier to display this if we restrict ourselves to the same seven attribute distributions that were shown in Table 2.1.2.

Table 2.1.3 (b) row displays the correlation coefficient,  $c$ , for each distribution set of join attribute values between  $r_1$  and  $r_2$ . In row (c),  $|A| c \delta_1 \delta_2$  represents the correction factor of the join size. The result of  $|r_1| \cdot |r_2| / |A|$  is shown in row (d) and is equal to the average join size over the entire possible experiment space of 1540 distribution sets. We will discuss its implication later in Corollary 2.2.1. The sum of (c) and (d) shown in (e) matches the exact join size as we expected in (a).

Analysis of the more general  $\theta$ -join operation is much more difficult and in general intractable. Table 2.1.4 is constructed to exhibit the distribution of less-join attribute values for the same seven join attribute sets of  $r_1$  and  $r_2$  in Table 2.1.1.

---

	Distribution sets						
	(1)	(2)	(3)	(4)	(5)	(6)	(7)
(a) join size	54	27	21	18	15	12	0
(b) $c$	1.0	.98	.87	0	-.87	-.65	-.50
(c) $ A c\delta_1\delta_2$	36	9	3	0	-3	-6	-18
(d) $ r_1  \cdot  r_2  /  A $	18	18	18	18	18	18	18
(e) (c) + (d)	54	27	21	18	15	12	0

---

Table 2.1.3 Results of join size parameters

We notice that the analysis for the distribution of  $r_1$  join attribute values can be similar to that used in the theorem, but the distribution of  $r_2$  is quite different in that the number of attribute values decreases as the join attribute value increases, except the set (7) in Table 2.1.4. It is evident that a more complex model is needed to represent the join distribution space and the expected join size in case of  $\theta$ -joins. Since it is not in the scope of this study, we will not pursue this discussion.

Once again reconsidering only the natural join, we look at Table 2.1.3. This example demonstrates the dependence problem between the distributions of join attribute values in the relations  $r_1$  and  $r_2$ . If it is *known* that  $\mu_1(a_k)$  and  $\mu_2(a_k)$  are two independent variables in Theorem 2.1.1, i.e., the correlation coefficient  $c = 0$ , then we have a simpler formula  $|r_1 \bowtie_A r_2| = |r_1| \cdot |r_2| / |A|$ . Or if it is *known* that there is a perfectly uniform distribution of join attribute values in *either*  $r_1$  or  $r_2$ , then the following corollary shows that the same result holds regardless of the distribution of join attribute values in the other relation!

Distribution of "less"-join attribute values														
Join Domain Values	(1)		(2)		(3)		(4)		(5)		(6)		(7)	
	r <sub>1</sub>	r <sub>2</sub>	r <sub>1</sub>	r <sub>2</sub>	r <sub>1</sub>	r <sub>2</sub>	r <sub>1</sub>	r <sub>2</sub>	r <sub>1</sub>	r <sub>2</sub>	r <sub>1</sub>	r <sub>2</sub>	r <sub>1</sub>	r <sub>2</sub>
1	0	0	1	8	1	7	2	6	1	4	1	3	0	0
2	0	0	1	6	2	5	2	3	2	2	1	1	0	0
3	6	0	4	0	3	0	2	0	3	0	4	0	6	0
Join Size	0		14		17		18		8		4		0	

Table 2.1.4 Results of the less-join

**Corollary 2.1.1** If the join attribute values of either relation have a perfectly uniform distribution, that is, if for all values  $a_k$  in join attribute A,

either  $\mu_1(a_k) = \frac{|r_1|}{|A|}$  or  $\mu_2(a_k) = \frac{|r_2|}{|A|}$ , then we have the exact equality:

$$|r_1 \bowtie_A r_2| = \frac{|r_1| \cdot |r_2|}{|A|}$$

**Proof :**

From Theorem 2.1.1 we have

$$|r_1 \bowtie_A r_2| = \sum_A (\mu_1(a_k) \cdot \mu_2(a_k))$$

Assume the distribution of join attribute values in  $r_1$  is perfectly uniform,

then we have  $\mu_1(a_k) = \frac{|r_1|}{|A|}$ . So that

$$|r_1 \bowtie_A r_2| = \sum_A \left( \frac{|r_1|}{|A|} \cdot \mu_2(a_k) \right)$$

$$\begin{aligned}
&= \frac{|r_1|}{|A|} \cdot \sum_A \mu_2(a_k) \\
&= \frac{|r_1| \cdot |r_2|}{|A|} \quad \square
\end{aligned}$$

## 2.2. Expected size of the natural join

So far, we have shown that the join size depends on the sizes of two joining relations and their join attribute size. This expression was asserted in [Ros81] as an expectation. We have refined this earlier work by introducing a correction factor reflecting the actual distribution of the join attribute values, i.e., the correlation coefficient and standard deviations, to obtain an exact expression for the *actual* size of the resultant join.

Unfortunately, in real applications, it is often not practical to derive the exact join size by computing the correlation coefficient and standard deviations. An expectation is usually sufficient. In the following results we rigorously derive a series of important expectations. In addition, we have used probabilistic simulations to verify most of our claims. The relations, or files, we used in our empirical simulations were all generated by some probability distribution of join or selection attribute values.

As the results of Table 2.1.3 (d) indicate, we can define an expression for the *expected* size of a joined relation as in the next corollary.

**Corollary 2.2.1** Let  $r_1$  and  $r_2$  be two randomly generated relations with a common attribute  $A$ . Then

$$\exp(|r_1 \bowtie_A r_2|) = \frac{|r_1| \cdot |r_2|}{|A|}$$

**Proof :**

From Theorem 2.1.1 we know

$$|r_1 \bowtie_A r_2| = \frac{|r_1| \cdot |r_2|}{|A|} + |A|c\delta_1\delta_2$$

To find the expected join size, we consider computing the average join size from all possible combinations of join attribute value distributions for relations  $r_1$  and  $r_2$ .

First, we show two special cases whose correction terms ( $|A|c\delta_1\delta_2$ ) in the expression of join size are equal to zeroes. For a perfectly uniform distribution, there are exactly  $\frac{|r|}{|A|}$  occurrences for each value in the join attribute as stated in Corollary 2.1.1. From [Wal78] we know that the value of the correlation coefficient is equal to 0 in this case. It has been shown in [Hoe62] that  $c = 0$  is also true for all other distributions in which the variables  $\mu_1(a_k)$  and  $\mu_2(a_k)$  are independent with respect to each other, where  $\mu_i(a_k)$  is the number of tuples with join attribute value  $a_k$  in relation  $r_i$ .

In general, we know that the value of correlation coefficient  $c$  is from  $-1.0$  to  $1.0$  [Mey65]. And for every distribution  $x$  with  $-1.0 \leq c < 0$ , there always exists a distribution  $y$  such that  $0 < c \leq 1.0$  and  $(c\delta_1\delta_2)_x + (c\delta_1\delta_2)_y = 0$ . Then it is obvious that  $\exp(c\delta_1\delta_2) = 0$  since the sum of all possible  $c\delta_1\delta_2$ 's is 0.

$$\text{Hence } \exp(|r_1 \bowtie_A r_2|) = \exp\left(\frac{|r_1| \cdot |r_2|}{|A|}\right) = \frac{|r_1| \cdot |r_2|}{|A|} \quad \square$$

Now we can claim that the expected join size of randomly generated relations is the join size without the correction term, in other words,  $\exp(\mu_1 \cdot \mu_2) = \exp(\mu_1) \cdot \exp(\mu_2)$  and  $c = 0$  can be assumed in the calculation.

Care must be used in determining the size of a attribute; by  $|A|$  we mean the *actual number of distinct values* in the attribute. From the result of Corollary 2.2.1, we notice that  $\exp(|r_1 \bowtie_A r_2|)$  will decrease when  $|A|$  increases and  $|r_1|$  and  $|r_2|$

remain the same size.

In general applications, selection operators are often used in queries. To cut down the access cost of the join, which is the topic of our next section, we usually perform the specified selection operations on both relations before joining them. The next corollary and theorem give the estimation of the final join size.

**Corollary 2.2.2** Let  $r_1$  and  $r_2$  have a common attribute  $A$ . Assume that the actual attribute values of  $A$  in  $r_1$  and  $r_2$  are subsets of the attribute domain of  $A$  and denote their sizes by  $|A_1|$  and  $|A_2|$  respectively. Also let  $A_{12} = A_1 \cap A_2$ , i.e.,  $A_{12}$  is the set of common attribute values in both  $r_1$  and  $r_2$ . Then

$$\exp(|r_1 \bowtie_A r_2|) = |A_{12}| \cdot \frac{|r_1|}{|A_1|} \cdot \frac{|r_2|}{|A_2|}$$

**Proof :**

From Theorem 2.1.1 and Corollary 2.2.1 we have the following two equations for  $r_1$  and  $r_2$  having common join attribute  $A$

$$|r_1 \bowtie_A r_2| = |A| \cdot \exp(\mu_1 \cdot \mu_2)$$

$$\exp(|r_1 \bowtie_A r_2|) = |A| \cdot \exp(\mu_1) \cdot \exp(\mu_2) = |A| \cdot \frac{|r_1|}{|A_1|} \cdot \frac{|r_2|}{|A_2|}$$

Then, in our case here

$$\begin{aligned} \exp(|r_1 \bowtie_A r_2|) &= \exp(|R_1 \bowtie_{A_{12}} R_2|) \\ &= |A_{12}| \cdot \exp(\mu_1) \cdot \exp(\mu_2) \\ &= |A_{12}| \cdot \frac{|r_1|}{|A_1|} \cdot \frac{|r_2|}{|A_2|} \quad \square \end{aligned}$$



**Theorem 2.2.1** Let  $\rho_1$  and  $\rho_2$  be selection formulae on relations  $r_1$  and  $r_2$  respectively, and assume  $P_1$ ,  $P_2$  and  $P_{12}$  are the probabilities that at least one tuple with join attribute value  $a_k$  will be selected in  $r_1$ ,  $r_2$ , and  $r_1 \Join_A r_2$  accordingly. Let  $\sigma_{\rho_j}(r_j) = r_j^*$  and  $|r_j^*| = g_j \cdot |r_j|$ , where  $g_j$  is the fraction of tuples selected in  $r_j$  and  $j = 1, 2$ . Then

$$\exp(|r_1^* \Join_A r_2^*|) = \frac{P_{12}}{P_1 P_2} \cdot g_1 \cdot g_2 \cdot \exp(|r_1 \Join_A r_2|)$$

**Proof :**

Let  $A_1$  and  $A_2$  be the attributes with actual attribute values in  $r_1^*$  and  $r_2^*$  respectively, and  $A_{12} = A_1 \cap A_2$ , then  $|A_1| = P_1 \cdot |A|$ ,  $|A_2| = P_2 \cdot |A|$ , and  $|A_{12}| = P_{12} \cdot |A|$ . From Corollary 2.2.2 we know that

$$\exp(|r_1 \Join_{A_{12}} r_2|) = |A_{12}| \cdot \frac{|r_1|}{|A_1|} \cdot \frac{|r_2|}{|A_2|}$$

Since  $|r_1^*| = g_1 \cdot |r_1|$  and  $|r_2^*| = g_2 \cdot |r_2|$ , then

$$\begin{aligned} \exp(|r_1^* \Join_A r_2^*|) &= \exp(|r_1^* \Join_{A_{12}} r_2^*|) \\ &= |A_{12}| \cdot \frac{|r_1^*|}{|A_1|} \cdot \frac{|r_2^*|}{|A_2|} \\ &= P_{12} |A| \cdot \frac{g_1 |r_1|}{P_1 |A|} \cdot \frac{g_2 |r_2|}{P_2 |A|} \\ &= \frac{P_{12}}{P_1 P_2} \cdot g_1 \cdot g_2 \cdot \frac{|r_1| \cdot |r_2|}{|A|} \\ &= \frac{P_{12}}{P_1 P_2} \cdot g_1 \cdot g_2 \cdot \exp(|r_1 \Join_A r_2|) \quad \square \end{aligned}$$

## CHAPTER 3

### Error Analysis of the Expected Join Size

#### 3.1. Estimation errors

In the previous chapter we presented a set of formulas for the exact and expected size of a join. The major difficulty of utilizing the exact size formulas of a join is the high cost of computing and maintaining the statistical parameters, correlation coefficient and standard deviations between the join attribute value distributions of two source relations. The overhead to maintain these statistics in a run time database manager is prohibitive. Since an expectation is usually sufficient in most applications, the expected join size formula we derived is most often used. However, one would like be able to bound the degree of error in order to insure that these expectations are within acceptable ranges.

We have shown in Chapter 2 that the difference between the exact join size and the expected size is the correction term  $|A|c\delta_1\delta_2$ , i.e.,

$$\begin{aligned} |r_1 \bowtie_A r_2| - \exp(|r_1 \bowtie_A r_2|) &= \left( \frac{|r_1| \cdot |r_2|}{|A|} + |A|c\delta_1\delta_2 \right) - \left( \frac{|r_1| \cdot |r_2|}{|A|} \right) \\ &= |A|c\delta_1\delta_2 \end{aligned}$$

where  $|A|$  is the domain size of a join attribute  $A$ ,  $c$  and  $\delta_i$  are the correlation coefficient and standard deviation for the distribution of join attribute values respectively. The correction term is just the error of estimating the join size. This we call the estimation error. However, determining the statistical parameters used to calculate the estimation error  $|A|c\delta_1\delta_2$  has precisely the same cost of computing the exact join size. We need a different approach to determine the estimation error.

Two strategies are considered in our study. First, in a system that requires a high precision on the join size we use approximation methods to estimate the

statistical parameters. Since the exact size of a join depends on the distributions of the join attribute values, the correlation coefficient and standard deviation are the best measurements to indicate the degree of error. A random sampling technique and other two methods will be presented in the next chapter to give a reasonable estimation of the correction term. The second strategy is for systems that require less accuracy. Our approach is to use the expected size formula and minimize the worst-case estimation error without using statistical parameters. By the worst-case estimation error we mean the maximum possible error. In the rest of this chapter we will investigate the maximum estimation errors that can be computed without involving statistical variables.

Consider the example of the attribute value distributions in the beginning of Chapter 2 (Tables 2.1.1). From Tables 2.1.2 and 2.1.3 we know the expected join size for all distributions is  $\frac{|r| \cdot |s|}{|A|} = \frac{6 \cdot 9}{3} = 18$  while the upper and lower bounds are  $|r| \cdot |s| = 6 \cdot 9 = 54$  and 0 respectively. In this case the maximum estimation error is the maximum value between  $(54 - 18) = 36$  and  $(18 - 0) = 18$ , that is 36. In the next section we will formulate an exact expression of the maximum estimation error for the expected join size. Then three algorithms will be presented in Section 3 in which the maximum estimation errors can be reduced significantly.

### 3.2. Maximum estimation errors

Let  $m$  denote the method of estimating the join size of two relations, and let  $Est_m(|r \bowtie_A s|)$  and  $Err_m(|r \bowtie_A s|)$  denote the estimated join size and its estimation error for relations  $r$  and  $s$  and join attribute  $A$ . Then

$$|r \bowtie_A s| = Est_m(|r \bowtie_A s|) + Err_m(|r \bowtie_A s|) \quad \text{or equivalently}$$

$$Err_m(|r \bowtie_A s|) = |r \bowtie_A s| - Est_m(|r \bowtie_A s|).$$

We use *UP* (for UnPartitioned) to denote an estimation method that does not partition the source relations and join attribute domain into smaller subsets. Rosenthal's estimate is unpartitioned, that is

$$Est_{UP}(|r \bowtie_A s|) = \frac{|r| \cdot |s|}{|A|} \text{ and}$$

$$Err_{UP}(|r \bowtie_A s|) = |r \bowtie_A s| - \frac{|r| \cdot |s|}{|A|}$$

where  $|r|$ ,  $|s|$ , and  $|A|$  represent the sizes of relations  $r$  and  $s$  and the size of a join attribute  $A$ .

Since the maximum and minimum values of  $|r \bowtie_A s|$  are  $|r| \cdot |s|$  and 0 respectively, we know that the maximum estimation error, denoted by  $MaxErr_{UP}(|r \bowtie_A s|)$  is either

$$|r| \cdot |s| - \frac{|r| \cdot |s|}{|A|} = \frac{|r| \cdot |s| (|A| - 1)}{|A|} \quad \text{or} \quad \frac{|r| \cdot |s|}{|A|} - 0 = \frac{|r| \cdot |s|}{|A|}$$

and for  $|A| \geq 2$ , the former is dominant. For  $|A| = 1$  we know the estimation error is zero since the exact join size is equal to the expected join size, i.e.,

$$|r| \cdot |s| = \frac{|r| \cdot |s|}{1} = \frac{|r| \cdot |s|}{|A|}$$

Therefore, the maximum estimation error is

$$MaxErr_{UP}(|r \bowtie_A s|) = \frac{|r| \cdot |s| (|A| - 1)}{|A|}$$

Note that we are concerned only with the "absolute value" of the estimation error, because the sign of the estimation error can not be determined without actually computing the correlation coefficient of the join attribute value distribution. Therefore,

$$MaxErr_{UP}(|r \bowtie_A s|) \geq Err_{UP}(|r \bowtie_A s|) \geq 0$$

Again, consider the example in the previous section where  $|r| = 6$ ,  $|s| = 9$ ,  $|A| = 3$ , and  $\exp(|r \bowtie_A s|) = \frac{6 \cdot 9}{3} = 18$ . The maximum estimation error is

$$\frac{|r| \cdot |s| (|A| - 1)}{|A|} = \frac{6 \cdot 9 \cdot (3 - 1)}{3} = 36$$

which agrees with the result  $(54 - 18) = 36$  we observed in Section 1.

### 3.3. Minimization of the maximum estimation errors

We know that the join size depends on the distribution of join attribute values between the joining relations  $r$  and  $s$ . The maximum estimation error of a join size can be reduced if we partition the source relations and join attribute domain into subsets. This follows, because the sum of the maximum estimation error in each subset will not be greater than the maximum estimation error of the unpartitioned set (a complete proof will be shown in the following theorem). For example, let us partition the join domain  $D_A$  and relations  $r$  and  $s$  into  $k$  subsets such that  $|A| = k$ , and  $|A_i| = |A_j| = 1$  for  $i, j = 1, \dots, k$ . Then we know the (maximum) estimation error is equal to zero because the expected join size in each partition is the same as the exact join size, i.e.,

$$\frac{|r_i| \cdot |s_i|}{|A_i|} = \frac{|r_i| \cdot |s_i|}{1} = |r_i| \cdot |s_i|$$

Therefore, the effect of this partition in estimating the join size is the same as if we performed the join operation by sequentially scanning  $r$  and  $s$  to find the matched tuples on the join attribute  $A$ . A formal definition of partitioning will be described before we show that  $\text{MaxErr}_{UP}(|r \bowtie_A s|) \geq \text{MaxErr}_P(|r \bowtie_A s|)$  in Theorem 3.3.1. Here  $P$  is used to denote the estimation method with partitioning.

An **A-partition** of a relation  $r$  is a subdivision of  $r$  into disjoint subsets  $r_1, r_2, \dots, r_i, \dots, r_k$  such that for every two tuples  $t_1, t_2$  with the property that their attribute values on the join attribute  $A$  are members of the subdomain  $D_{A_i}$ ,  $t_1$

and  $t_2$  are in the same subset  $r_i$ . That is, one first partitions the domain of  $A$ , and then lets this induce a second partition on  $r$  and similarly for  $s$ . Note that the domain of attribute  $A_i$  contains the attribute values in the subsets  $r_i$  and  $s_i$ , such that

$$r \bowtie_A s = \bigcup_{i=1}^k r_i \bowtie_{A_i} s_i$$

and

$$|r \bowtie_A s| = \sum_{i=1}^k |r_i \bowtie_{A_i} s_i|$$

From Section 2, we know that for each pair of subsets in  $r$  and  $s$

$$Est_{UP}(|r_i \bowtie_{A_i} s_i|) = \frac{|r_i| \cdot |s_i|}{|A_i|} \text{ and}$$

$$MaxErr_{UP}(|r_i \bowtie_{A_i} s_i|) = \frac{|r_i| \cdot |s_i| (|A_i| - 1)}{|A_i|}.$$

Therefore

$$Est_P(|r \bowtie_A s|) = \sum_{i=1}^k Est_{UP}(|r_i \bowtie_{A_i} s_i|) = \sum_{i=1}^k \frac{|r_i| \cdot |s_i|}{|A_i|} \text{ and}$$

$$MaxErr_P(|r \bowtie_A s|) = \sum_{i=1}^k MaxErr_{UP}(|r_i \bowtie_{A_i} s_i|) = \sum_{i=1}^k \frac{|r_i| \cdot |s_i| (|A_i| - 1)}{|A_i|}$$

Now in the following theorem we show that the maximum estimation error can be reduced if the partition method is used to estimate the join size.

**Theorem 3.3.1** Let  $|A_i|$ ,  $|r_i|$ , and  $|s_i|$  denote the sizes of  $A$ -partitioned join attribute

$A$ , and relations  $r$  and  $s$  respectively,  $i = 1$  to  $k$ , with  $|r| = \sum_{i=1}^k |r_i|$ ,

$|s| = \sum_{i=1}^k |s_i|$ , and  $|A| = \sum_{i=1}^k |A_i|$ . Then  $MaxErr_{UP}(|r \bowtie_A s|) \geq$

$$MaxErr_P(|r \bowtie_A s|) \text{ or } \frac{|r| \cdot |s| \cdot (|A| - 1)}{|A|} \geq \sum_{i=1}^k \frac{|r_i| \cdot |s_i| \cdot (|A_i| - 1)}{|A_i|}$$

**Proof :**

Let  $n_i$ ,  $m_i$ ,  $a_i$  and  $a$  denote  $|r_i|$ ,  $|s_i|$ ,  $|A_i|$  and  $|A|$  respectively, then

$$RHS = \sum_{i=1}^k \frac{|r_i| \cdot |s_i| \cdot (|A_i| - 1)}{|A_i|} = \sum_{i=1}^k n_i m_i \left( \frac{a_i - 1}{a_i} \right)$$

$$LHS = \frac{|r| \cdot |s| \cdot (|A| - 1)}{|A|} = \left( \sum_{i=1}^k n_i \right) \left( \sum_{j=1}^k m_j \right) \left[ \frac{(a - 1)}{a} \right]$$

Let  $\left[ \frac{(a - 1)}{a} \right] = d$ , then we can rewrite  $LHS$  as follows :

$$\begin{aligned} LHS &= \left( \sum_{i=1}^k n_i \right) \left( \sum_{j=1}^k m_j \right) \cdot d = (n_1 + n_2 + \dots + n_k) \cdot (m_1 + m_2 + \dots + m_k) \cdot d \\ &= [(n_1 m_1 + \dots + n_1 m_k) + (n_2 m_1 + \dots + n_2 m_k) + \dots + (n_k m_1 + \dots + n_k m_k)] \cdot d \\ &= [(n_1 m_1 + n_2 m_2 + \dots + n_k m_k) + (n_1 m_2 + \dots + n_1 m_k + \dots + n_k m_1 + \dots + n_k m_{k-1})] \cdot d \\ &= \left[ \left( \sum_{i=1}^k n_i m_i \right) + \left( \sum_{i=1}^k \sum_{j \neq i} n_i m_j \right) \right] \cdot d \end{aligned}$$

So that

$$LHS = \left[ \sum_{i=1}^k n_i m_i \frac{(a - 1)}{a} \right] + \left[ \sum_{i=1}^k \sum_{j \neq i} n_i m_j \frac{(a - 1)}{a} \right] \quad (a)$$

Since  $|r_i| = n_i \geq 0$ ,  $|s_i| = m_i \geq 0$ , and  $|A| = a \geq 1$ , we know that

$$[\sum_{i=1}^k \sum_{j \neq i} n_i m_j \frac{(a-1)}{a}] \geq 0$$

Hence

$$LHS \geq [\sum_{i=1}^k n_i m_i \frac{(a-1)}{a}] \quad (b)$$

Now we will show that

$$\frac{a-1}{a} \geq \frac{a_i-1}{a_i} \text{ for } i = 1, 2, \dots, k.$$

or equivalently

$$\frac{a-1}{a} - \frac{a_i-1}{a_i} \geq 0 \text{ for } i = 1, 2, \dots, k.$$

First, we know that  $a_i \geq 0$ ,  $a \geq 0$ , and  $a \geq a_i$ , i.e.,  $a - a_i \geq 0$  for  $i = 1, 2, \dots, k$ .

Then

$$\begin{aligned} \frac{a-1}{a} - \frac{a_i-1}{a_i} &= \frac{a_i(a-1) - a(a_i-1)}{a \cdot a_i} \\ &= \frac{(a_i \cdot a - a_i) - (a \cdot a_i - a)}{a \cdot a_i} \\ &= \frac{a - a_i}{a \cdot a_i} \geq 0 \end{aligned}$$

So we have proved that

$$\frac{a-1}{a} \geq \frac{a_i-1}{a_i} \text{ for } i = 1, 2, \dots, k.$$

Since  $|r_i| = n_i \geq 0$  and  $|s_i| = m_i \geq 0$ , it is obvious that

$$n_i m_i \frac{(a-1)}{a} \geq n_i m_i \frac{a_i-1}{a_i} \text{ and}$$

$$\sum_{i=1}^k [n_i m_i \frac{(a-1)}{a}] \geq \sum_{i=1}^k [n_i m_i \frac{a_i-1}{a_i}] = RHS \quad (c)$$

From the expressions (b) and (c) we know



$$LHS \geq \sum_{i=1}^k [n_i m_i \frac{(a-1)}{a}] \geq \sum_{i=1}^k [n_i m_i \frac{a_i-1}{a_i}] = RHS$$

That is 
$$\frac{|r| \cdot |s| \cdot (|A| - 1)}{|A|} \geq \sum_{i=1}^k \frac{|r_i| \cdot |s_i| \cdot (|A_i| - 1)}{|A_i|} \quad \square$$

The theorem shows that the maximum estimation error obtained by first partitioning the join domain, can never be worse than that using the unpartitioned estimate. But in practice, it is very much superior since we are at liberty to select the partition of  $A$ .

To show how the partition method works, a simplified example is given in Table 3.3.1. In part (a) two distributions of join attribute values are given for relations  $r$  and  $s$  of size  $|r| = 30$  and  $|s| = 20$ . The join attribute domain  $D_A$  consists of five integers  $\{1, 2, \dots, 5\}$ . In relations  $r$  and  $s$ , for example, there are 6 and 7 tuples having join attribute value 1 respectively. The last row of each relation shows the relation size. Now we can compute the unpartitioned maximum estimation error as follows:

$$MaxErr_{UP}(|r \bowtie_A s|) = \frac{|r| \cdot |s| (|A| - 1)}{|A|} = \frac{30 \cdot 20 (5 - 1)}{5} = 480.$$

Next, we partition the join domain into two subsets,  $D_{A_1} = \{1, 2, 3\}$  and  $D_{A_2} = \{4, 5\}$ . Accordingly,  $r$  and  $s$  are divided into two subsets  $(r_1, s_1)$  and  $(r_2, s_2)$  as shown in Table 3.3.1 (b) and (c) respectively. Here  $|r_1| = 20$ ,  $|s_1| = 12$ ,  $|r_2| = 10$  and  $|s_2| = 8$ . Using the formula of the preceding section

$$exp(|r \bowtie_A s|) = \frac{20 \cdot 12}{3} + \frac{10 \cdot 8}{2} = 120$$

with a maximum possible error

$$MaxErr_P(|r \bowtie_A s|) = \sum_{i=1}^2 \frac{|r_i| \cdot |s_i| (|A_i| - 1)}{|A_i|}$$

Join Domain Values	(a)		(b)		(c)	
	$r$	$s$	$r_1$	$s_1$	$r_2$	$s_2$
1	6	7	6	7		
2	4	3	4	3		
3	10	2	10	2		
4	2	5			2	5
5	8	3			8	3
	<hr/>	<hr/>	<hr/>	<hr/>	<hr/>	<hr/>
	30	20	20	12	10	8

Table 3.3.1 An A-partition

$$\begin{aligned}
&= \frac{|r_1| \cdot |s_1| (|A_1| - 1)}{|A_1|} + \frac{|r_2| \cdot |s_2| (|A_2| - 1)}{|A_2|} \\
&= \frac{20 \cdot 12 (3 - 1)}{3} + \frac{10 \cdot 8 (2 - 1)}{2} = 160 + 40 \\
&= 200
\end{aligned}$$

It is clear that  $MaxErr_{UP}(r \bowtie_A s) \geq MaxErr_P(r \bowtie_A s)$ , because  $480 > 200$ .

Note that if the partition size  $k = |A|$ , i.e.,  $|A_i| = |A_j| = 1$  for  $i, j = 1, 2, \dots, k$ , then

$$MaxErr_P(r \bowtie_A s) = \sum_{i=1}^k \frac{|r_i| \cdot |s_i| (|A_i| - 1)}{|A_i|} = 0$$

Now we have clearly demonstrated the advantage of partitioning. The next issue is how much performance improvement one can expect. In the following corollary we will show that  $MaxErr_{UP}(r \bowtie_A s) \geq \omega + MaxErr_P(r \bowtie_A s)$ . Here  $\omega =$

$$\sum_{i=1}^k \sum_{j \neq i} \frac{|r_i| \cdot |s_j| \cdot (|A| - 1)}{|A|}.$$

That is, the maximum error for estimating the join size can be reduced by  $\omega$  when the source relations and join attribute domain are

partitioned into subsets.

**Corollary 3.3.1** Let  $|A_i|$ ,  $|r_i|$ , and  $|s_i|$  denote the sizes of A-partitioned join attribute

$A$ , and relations  $r$  and  $s$  respectively,  $i = 1$  to  $k$ , with  $|r| = \sum_{i=1}^k |r_i|$ ,

$|s| = \sum_{i=1}^k |s_i|$ , and  $|A| = \sum_{i=1}^k |A_i|$ . Then

$$\frac{|r| \cdot |s| \cdot (|A| - 1)}{|A|} \geq \omega + \sum_{i=1}^k \frac{|r_i| \cdot |s_i| \cdot (|A_i| - 1)}{|A_i|}$$

$$\text{where } \omega = \sum_{i=1}^k \sum_{j \neq i} \frac{|r_i| \cdot |s_j| \cdot (|A| - 1)}{|A|}.$$

**Proof :**

Let  $n_i$ ,  $m_i$ ,  $a_i$  and  $a$  denote  $|r_i|$ ,  $|s_i|$ ,  $|A_i|$  and  $|A|$  respectively, then

$$RHS = \omega + \sum_{i=1}^k \frac{|r_i| \cdot |s_i| \cdot (|A_i| - 1)}{|A_i|} = \omega + \sum_{i=1}^k n_i m_i \left( \frac{a_i - 1}{a_i} \right)$$

$$LHS = \frac{|r| \cdot |s| \cdot (|A| - 1)}{|A|} = \left( \sum_{i=1}^k n_i \right) \left( \sum_{j=1}^k m_j \right) \left[ \frac{(a - 1)}{a} \right]$$

From the expressions (a) and (c) in Theorem 3.3.1 we know that

$$\begin{aligned} LHS &= \left[ \sum_{i=1}^k n_i m_i \frac{(a - 1)}{a} \right] + \left[ \sum_{i=1}^k \sum_{j \neq i} n_i m_j \frac{(a - 1)}{a} \right] \\ &= \left[ \sum_{i=1}^k n_i m_i \frac{(a - 1)}{a} \right] + \omega \end{aligned}$$

and

$$\sum_{i=1}^k n_i m_i \frac{(a - 1)}{a} \geq \sum_{i=1}^k n_i m_i \frac{a_i - 1}{a_i}$$

We also know that  $\omega \geq 0$ , since  $|r_i|, |s_j| \geq 0$  and  $|A| \geq 1$ . Then

$$LHS = \omega + \sum_{i=1}^k n_i m_i \frac{(a-1)}{a} \geq \omega + \sum_{i=1}^k n_i m_i \frac{a_i-1}{a_i} = RHS$$

$$\text{That is } \frac{|r| \cdot |s| \cdot (|A|-1)}{|A|} \geq \omega + \sum_{i=1}^k \frac{|r_i| \cdot |s_i| \cdot (|A_i|-1)}{|A_i|} \quad \square$$

To empirically verify the expression of Corollary 3.3.1, we consider the same example in Table 3.3.1. We know  $|r_1| = 20$ ,  $|s_1| = 12$ ,  $|r_2| = 10$ ,  $|s_2| = 8$  and  $|A| = 5$ . Then

$$\begin{aligned} \omega &= \sum_{i=1}^k \sum_{j \neq i} \frac{|r_i| \cdot |s_j| \cdot (|A|-1)}{|A|} \\ &= \frac{|r_1| \cdot |s_2| \cdot (|A|-1)}{|A|} + \frac{|r_2| \cdot |s_1| \cdot (|A|-1)}{|A|} \\ &= \frac{20 \cdot 8 \cdot (5-1)}{5} + \frac{10 \cdot 12 \cdot (5-1)}{5} = 128 + 96 \\ &= 224 \end{aligned}$$

Since  $MaxErr_{UP}(|r \bowtie_A s|) = 480$ ,  $MaxErr_P(|r \bowtie_A s|) = 200$  and  $480 > 224 + 200 = 424$ , we have verified that

$$MaxErr_{UP}(|r \bowtie_A s|) \geq \omega + MaxErr_P(|r \bowtie_A s|)$$

From the above discussion, one can notice that the more subsets we have in the partition method the smaller the maximum estimation error we get. It is not hard to show that the observation is true. Consider a possible case that we have  $n$  subsets of  $r_i$  and  $s_i$  in the first partition. If we further partition each (parent) subset into  $m$  smaller (son) subsets,  $r_{ij}$  (or  $s_{ij}$ ) for  $j = 1, \dots, m$ . Then from Theorem 3.3.1 we know that for each parent subset the sum of the maximum estimation errors from the son subsets of the second partition will not be greater than its own maximum estimation error. That is

$$MaxErr_{UP}(|r_i \bowtie_{A_i} s_i|) \geq MaxErr_P(|r_i \bowtie_{A_i} s_i|) = \sum_{j=1}^m MaxErr_{UP}(|r_{ij} \bowtie_{A_{ij}} s_{ij}|)$$

Therefore, the total value of the maximum estimation errors from the second partition will not be greater than the maximum estimation error of the unpartitioned set. That is

$$\begin{aligned} MaxErr_{UP}(|r \bowtie_A s|) &\geq MaxErr_P(|r \bowtie_A s|) \\ &= \sum_{i=1}^n MaxErr_{UP}(|r_i \bowtie_{A_i} s_i|) \\ &\geq \sum_{i=1}^n MaxErr_P(|r_i \bowtie_{A_i} s_i|) \\ &= \sum_{i=1}^n \sum_{j=1}^m MaxErr_{UP}(|r_{ij} \bowtie_{A_{ij}} s_{ij}|) \end{aligned}$$

As yet no specific approach has been mentioned of how to make a partition. That is to say we can partition the source relations by arbitrarily subdividing the join attribute domain into subsets. Three methods of partitioning will be investigated in the remainder of this chapter. First, one of the source relations is equally divided into  $k$  subsets. Second, we partition the join attribute domain into  $k$  equal size subdomains. Third, two source relations are partitioned such that  $|r_i| \cdot |s_i| = |r_j| \cdot |s_j|$  for  $i, j = 1, \dots, k$ .

In the next corollary we present the first approach of partitioning the source relations and join attribute domain into subsets, and show that the maximum estimation error can be reduced by a factor of the partition size. Let  $ER$  denote the method in which one of the source relations is partitioned into  $k$  equal size subsets. Then

$$MaxErr_{UP}(|r \bowtie_A s|) \geq k \cdot MaxErr_{ER}(|r \bowtie_A s|).$$

**Corollary 3.3.2** If we partition relation  $s$  (or  $r$ ) into  $k$  equal size subsets, i.e.,  $|s_i| = |s_j|$  for  $i, j = 1, 2, \dots, k$ , then the maximum estimation error of join size can be reduced by a factor of  $k$  compared to estimation with no partitioning. By using the same notation of Theorem 3.3.1, we have the following:

$$\frac{|r| \cdot |s| \cdot (|A| - 1)}{|A|} \geq k \sum_{i=1}^k \frac{|r_i| \cdot |s_i| \cdot (|A_i| - 1)}{|A_i|}$$

**Proof :**

If  $|s_i| = |s_j|$  for  $i, j = 1, 2, \dots, k$ , then  $|s_j| = \frac{|s|}{k}$  and

$$\begin{aligned} \omega &= \sum_{i=1}^k \sum_{j \neq i} \frac{|r_i| \cdot |s_j| \cdot (|A| - 1)}{|A|} \\ &= \sum_{i=1}^k \sum_{j \neq i} \frac{|r_i| \cdot \frac{|s|}{k} \cdot (|A| - 1)}{|A|} \\ &= (k-1) \sum_{i=1}^k \frac{|r_i| \cdot \frac{|s|}{k} \cdot (|A| - 1)}{|A|} \\ &= (k-1) \frac{\frac{|s|}{k} \cdot (|A| - 1)}{|A|} \sum_{i=1}^k |r_i| \\ &= \left(\frac{k-1}{k}\right) \frac{|r| \cdot |s| \cdot (|A| - 1)}{|A|} \end{aligned}$$

From corollary 3.3.1 we know that

$$\frac{|r| \cdot |s| \cdot (|A| - 1)}{|A|} \geq \omega + \sum_{i=1}^k \frac{|r_i| \cdot |s_i| \cdot (|A_i| - 1)}{|A_i|}$$

Then

$$\begin{aligned}
\frac{|r| \cdot |s| \cdot (|A| - 1)}{|A|} - \omega &\geq \sum_{i=1}^k \frac{|r_i| \cdot |s_i| \cdot (|A_i| - 1)}{|A_i|} \\
\frac{|r| \cdot |s| \cdot (|A| - 1)}{|A|} - \left(\frac{k-1}{k}\right) \frac{|r| \cdot |s| \cdot (|A| - 1)}{|A|} &\geq \sum_{i=1}^k \frac{|r_i| \cdot |s_i| \cdot (|A_i| - 1)}{|A_i|} \\
\left(\frac{1}{k}\right) \frac{|r| \cdot |s| \cdot (|A| - 1)}{|A|} &\geq \sum_{i=1}^k \frac{|r_i| \cdot |s_i| \cdot (|A_i| - 1)}{|A_i|} \\
\frac{|r| \cdot |s| \cdot (|A| - 1)}{|A|} &\geq k \sum_{i=1}^k \frac{|r_i| \cdot |s_i| \cdot (|A_i| - 1)}{|A_i|} \quad \square
\end{aligned}$$

Corollary 3.3.2 provides a very powerful way to reduce the maximum estimation errors by partitioning one of the source relations into  $k$  equal size subsets. The improvement is a factor of  $k$ . Most importantly, this method enables us to arbitrarily reduce the maximum estimation errors by increasing the partition size  $k$ . When  $k$  reaches the size of the join attribute domain, the maximum estimation error becomes zero. For example, consider the same example in Table 3.3.1. This time we use the *ER* partition method as shown in Table 3.3.2.

Now  $|r_1| = |r_2| = |r_3| = 10$ ,  $|s_1| = 10$ ,  $|s_2| = 2$ ,  $|s_3| = 8$ ,  $|A_1| = 2$ ,  $|A_2| = 1$  and  $|A_3| = 2$ . Using the formula of the preceding section

$$\exp(|r \bowtie_A s|) = \frac{10 \cdot 10}{2} + \frac{10 \cdot 2}{1} + \frac{10 \cdot 8}{2} = 110$$

with a maximum possible error

$$\begin{aligned}
MaxErr_{ER}(|r \bowtie_A s|) &= \sum_{i=1}^3 \frac{|r_i| \cdot |s_i| \cdot (|A_i| - 1)}{|A_i|} \\
&= \frac{10 \cdot 10(2 - 1)}{2} + \frac{10 \cdot 2(1 - 1)}{1} + \frac{10 \cdot 8(2 - 1)}{2} \\
&= 50 + 0 + 40 = 90
\end{aligned}$$

Since  $MaxErr_{UP}(|r \bowtie_A s|) = 480$  and  $k = 3$ , we know

Join Domain Values	(a)		(b)		(c)	
	$r_1$	$s_1$	$r_2$	$s_2$	$r_3$	$s_3$
1	6	7				
2	4	3				
3			10	2		
4					2	5
5					8	3
	10	10	10	2	10	8

Table 3.3.2 An equal-sized partition (*ER*)

$$MaxErr_{UP}(|r \bowtie s|) \geq k \cdot MaxErr_{ER}(|r \bowtie s|) = 3 \cdot 90 = 270$$

As shown in the example we divided the joining relations into  $k$  subsets such that one of the relations is equally divided, i.e.,  $|r_i| = |r_j|$  for  $i, j = 1, 2, \dots, k$ . In the real world it is not practical to first partition one relation  $r$  into equal size subsets and then find out the subdomain for each  $A_j$  in order to partition the other relation  $s$ .

In practice, we divide the domain of join attribute  $A$  into equal size subdomains for  $A_j$  and partition the joining relations  $r$  and  $s$  accordingly to reduce the maximum error of estimating the join size. We call this partitioning method *ED*. More specifically, we divide the domain of join attribute  $A$  into  $k$  equal size subdomains, i.e.,  $|A_j| = \frac{|A|}{k}$  for  $j = 1, 2, \dots, k$ . And the joining relations  $r$  and  $s$  are partitioned according to  $A_j$ . Then the maximum error of estimating the join size is the following:



$$\begin{aligned}
MaxErr_{ED}(r \bowtie_A s) &= \sum_{j=1}^k |r_j| \cdot |s_j| \cdot \frac{|A_j| - 1}{|A_j|} = \sum_{j=1}^k |r_j| \cdot |s_j| \cdot \frac{\frac{|A|}{k} - 1}{\frac{|A|}{k}} \\
&= \sum_{j=1}^k |r_j| \cdot |s_j| \cdot \frac{|A| - k}{|A|} = \sum_{j=1}^k |r_j| \cdot |s_j| \cdot \left(1 - \frac{k}{|A|}\right)
\end{aligned}$$

However, in theory a better result can be obtained if the source relations could be partitioned to have equal value of  $|r_i| \cdot |s_i|$ , i.e., we divide  $r$  and  $s$  into subsets  $r_i$  and  $s_i$  for  $i = 1, 2, \dots, k$  such that  $|r_i| \cdot |s_i| = |r_j| \cdot |s_j|$  for  $i, j = 1, 2, \dots, k$ . Note that  $|r_i| = |r_j|$  and  $|s_i| = |s_j|$  is just a special case. If we use *EP* for the name of this method, then

$$MaxErr_{EP}(r \bowtie_A s) = \sum_{i=1}^k |r_i| \cdot |s_i| \cdot \frac{|A_i| - 1}{|A_i|} = \sum_{i=1}^k \left( \frac{\sum_{j=1}^k |r_j| \cdot |s_j|}{k} \right) \cdot \frac{|A_i| - 1}{|A_i|}$$

In the following corollary we will show that

$$MaxErr_{ED}(r \bowtie_A s) \geq MaxErr_{EP}(r \bowtie_A s).$$

**Corollary 3.3.3** The maximum error in estimating the join size by partitioning the relations  $r$  and  $s$  into subsets  $r_j$  and  $s_j$  with equally sized subdomains of the join attribute  $A_j$  is greater than the maximum error resulting from partitioning these relations so that  $|r_j| \cdot |s_j|$  have equal values. By using the same notation of Theorem 3.3.1, we have the following:

$$\sum_{j=1}^k |r_j| \cdot |s_j| \cdot \frac{\frac{|A|}{k} - 1}{\frac{|A|}{k}} \geq \sum_{i=1}^k \left( \frac{\sum_{j=1}^k |r_j| \cdot |s_j|}{k} \right) \cdot \frac{|A_i| - 1}{|A_i|}$$

**Proof :**

For the method *ED* we have  $|A_j| = \frac{|A|}{k}$  for  $j = 1, 2, \dots, k$ , then

$$MaxErr_{ED}(|r \otimes_A s|) = LHS$$

$$= \sum_{j=1}^k |r_j| \cdot |s_j| \cdot \frac{\frac{|A|}{k} - 1}{\frac{|A|}{k}}$$

$$= (1 - \frac{k}{|A|}) \sum_{j=1}^k |r_j| \cdot |s_j|$$

For the method *EP* we know  $|r_i| \cdot |s_i| = |r_j| \cdot |s_j|$  for  $i, j = 1, 2, \dots, k$ , then

$$MaxErr_{EP}(|r \otimes_A s|) = RHS$$

$$= \sum_{i=1}^k \left( \frac{\sum_{j=1}^k |r_j| \cdot |s_j|}{k} \right) \cdot \frac{|A_i| - 1}{|A_i|}$$

$$= \left( \frac{\sum_{j=1}^k |r_j| \cdot |s_j|}{k} \right) \sum_{i=1}^k \frac{|A_i| - 1}{|A_i|}$$

$$= \left( \frac{1}{k} \sum_{i=1}^k \frac{|A_i| - 1}{|A_i|} \right) \sum_{j=1}^k |r_j| \cdot |s_j|$$

First, we will show that

$$\sum_{i=1}^k \left( 1 - \frac{k}{|A|} \right) \geq \sum_{i=1}^k \frac{|A_i| - 1}{|A_i|}.$$

Let  $x = \sum_{i=1}^k \left( 1 - \frac{k}{|A|} \right) - \sum_{i=1}^k \frac{|A_i| - 1}{|A_i|}$ , then we need to prove that  $x \geq 0$  :

$$x = \sum_{i=1}^k \left( 1 - \frac{k}{|A|} \right) - \sum_{i=1}^k \left( 1 - \frac{1}{|A_i|} \right)$$

$$= \left( k - \sum_{i=1}^k \frac{k}{|A|} \right) - \left( k - \sum_{i=1}^k \frac{1}{|A_i|} \right)$$

$$= k - k \frac{k}{|A|} - k + \sum_{i=1}^k \frac{1}{|A_i|}$$

$$= \left( \sum_{i=1}^k \frac{1}{|A_i|} \right) - \frac{k^2}{|A|}$$

Let  $|A_i| = n_i$  and  $|A| = \sum_{i=1}^k n_i$ , then

$$\begin{aligned}
x &= \left( \sum_{i=1}^k \frac{1}{n_i} \right) - \frac{k^2}{\sum_{i=1}^k n_i} \\
&= \left( \frac{1}{n_1} + \frac{1}{n_2} + \cdots + \frac{1}{n_k} \right) - \frac{k^2}{n_1 + n_2 + \cdots + n_k} \\
&= \frac{n_2 n_3 \cdots n_k + n_1 n_3 \cdots n_k + \cdots + n_1 n_2 \cdots n_{k-1}}{n_1 n_2 n_3 \cdots n_k} - \frac{k^2}{n_1 + n_2 + \cdots + n_k} \\
&= \frac{n_2 n_3 \cdots n_k (n_1 + \cdots + n_k) + n_1 n_3 \cdots n_k (n_1 + \cdots + n_k) + \cdots + n_1 n_2 \cdots n_{k-1} (n_1 + \cdots + n_k)}{n_1 n_2 n_3 \cdots n_k (n_1 + n_2 + \cdots + n_k)} \\
&\quad - \frac{k^2 (n_1 n_2 n_3 \cdots n_k)}{n_1 n_2 n_3 \cdots n_k (n_1 + n_2 + \cdots + n_k)} \\
&= \frac{y}{n_1 n_2 n_3 \cdots n_k (n_1 + n_2 + \cdots + n_k)}
\end{aligned}$$

Now we only need to show  $y \geq 0$  since  $n_1 n_2 n_3 \cdots n_k (n_1 + n_2 + \cdots + n_k) \geq 1$ , here

$$\begin{aligned}
y &= n_2 n_3 \cdots n_k (n_1 + n_2 + \cdots + n_k) + \\
&\quad n_1 n_3 \cdots n_k (n_1 + n_2 + \cdots + n_k) + \\
&\quad n_1 n_2 \cdots n_k (n_1 + n_2 + \cdots + n_k) + \\
&\quad \cdots + \\
&\quad n_1 n_2 n_3 \cdots n_{k-1} (n_1 + n_2 + \cdots + n_k) - \\
&\quad k^2 (n_1 n_2 n_3 \cdots n_k) \\
&= n_1 n_2 n_3 \cdots n_k + n_2^2 n_3 \cdots n_k + \cdots + n_2 n_3 \cdots n_k^2 + \\
&\quad n_1^2 n_3 \cdots n_k + n_1 n_2 n_3 \cdots n_k + \cdots + n_1 n_3 \cdots n_k^2 + \\
&\quad n_1^2 n_2 \cdots n_k + n_1 n_2^2 \cdots n_k + \cdots + n_1 n_2 \cdots n_k^2 + \\
&\quad \cdots + \\
&\quad n_1^2 n_2 n_3 \cdots n_{k-1} + n_1 n_2^2 n_3 \cdots n_{k-1} + \cdots + n_1 n_2 n_3 \cdots n_{k-1} n_k - \\
&\quad k^2 (n_1 n_2 n_3 \cdots n_k)
\end{aligned}$$

$$\begin{aligned}
&= n_1 n_2 \dots n_k - n_1 n_2 \dots n_k + (n_1^2 n_3 n_4 \dots n_k - n_1 n_2 \dots n_k) + \dots + [n_2 n_3 n_4 \dots n_k^2 - n_1 n_2 \dots n_k] + \\
&\quad (n_1^2 n_3 n_4 \dots n_k - n_1 n_2 \dots n_k) + n_1 n_2 \dots n_k - n_1 n_2 \dots n_k + \dots + n_1 n_3 n_4 \dots n_k^2 - n_1 n_2 \dots n_k + \\
&\quad n_1^2 n_2 n_4 \dots n_k - n_1 n_2 \dots n_k + n_1 n_2^2 n_4 \dots n_k - n_1 n_2 \dots n_k + \dots + n_1 n_2 n_4 \dots n_k^2 - n_1 n_2 \dots n_k + \\
&\quad \dots + \\
&\quad [n_1^2 n_2 n_3 \dots n_{k-1} - n_1 n_2 \dots n_k] + n_1 n_2^2 n_3 \dots n_{k-1} - n_1 n_2 \dots n_k + \dots + n_1 n_2 \dots n_k - n_1 n_2 \dots n_k
\end{aligned}$$

Then, we re-arrange the equation to put together the corresponding terms on both sides of the diagonal line :

$$\begin{aligned}
y &= (n_1^2 n_3 n_4 \dots n_k - n_1 n_2 n_3 \dots n_k + n_2^2 n_3 n_4 \dots n_k - n_1 n_2 n_3 \dots n_k) + \\
&\quad \dots + \\
&\quad [n_1^2 n_2 n_3 \dots n_{k-1} - n_1 n_2 n_3 \dots n_k + n_2 n_3 n_4 \dots n_k^2 - n_1 n_2 n_3 \dots n_k] + \\
&\quad \dots \\
&= n_3 n_4 \dots n_k (n_1^2 - n_1 n_2 + n_2^2 - n_1 n_2) + \\
&\quad \dots + \\
&\quad n_2 n_3 \dots n_{k-1} [n_1^2 - n_1 n_k + n_k^2 - n_1 n_k] + \\
&\quad \dots \\
&= n_3 n_4 \dots n_k (n_1 - n_2)^2 \\
&\quad \dots + \\
&\quad n_2 n_3 \dots n_{k-1} [n_1 - n_k]^2 + \\
&\quad \dots
\end{aligned}$$

Since  $(n_1 - n_2)^2 \geq 0$ , ...,  $(n_1 - n_k)^2 \geq 0$  we know that  $y \geq 0$  and  $x \geq 0$ .

Therefore, we have shown that

$$\sum_{i=1}^k \left(1 - \frac{k}{|A|}\right) \geq \sum_{i=1}^k \frac{|A_i| - 1}{|A_i|}.$$

Then

$$k \left(1 - \frac{k}{|A|}\right) \geq \sum_{i=1}^k \frac{|A_i| - 1}{|A_i|}$$

$$(1 - \frac{k}{|A|}) \geq \frac{1}{k} \sum_{i=1}^k \frac{|A_i| - 1}{|A_i|}$$

Since  $\sum_{j=1}^k |r_j| \cdot |s_j| \geq 0$ , we know

$$LHS = (1 - \frac{k}{|A|}) \sum_{j=1}^k |r_j| \cdot |s_j| \geq (\frac{1}{k} \sum_{i=1}^k \frac{|A_i| - 1}{|A_i|}) \sum_{j=1}^k |r_j| \cdot |s_j| = RHS$$

That is

$$\sum_{j=1}^k |r_j| \cdot |s_j| \cdot \frac{\frac{|A|}{k} - 1}{\frac{|A|}{k}} \geq \sum_{i=1}^k (\frac{\sum_{j=1}^k |r_j| \cdot |s_j|}{k}) \cdot \frac{|A_i| - 1}{|A_i|} \quad \square$$

To illustrate the application of the *EP* method to partition relations into subsets such that  $|r_i| \cdot |s_i| = |r_j| \cdot |s_j|$ , we construct two carefully planned relations  $r$  and  $s$  in Table 3.3.3. Three subsets  $\{a, b, c\}$  are then partitioned in a way that satisfies the required conditions:

$$|r_a| \cdot |s_a| = 10 \cdot 12 = 120,$$

$$|r_b| \cdot |s_b| = 10 \cdot 12 = 120,$$

$$|r_c| \cdot |s_c| = 15 \cdot 8 = 120.$$

Note that the join domain is not equally divided, i.e.,  $|A_a| = 2$ ,  $|A_b| = 1$ , and  $|A_c| = 3$ . Nor is  $|r_a| = |r_b| = |r_c|$ . Then, the maximum estimation error can be computed as follows

$$\begin{aligned} MaxErr_{EP}(r \bowtie_A s) &= \sum_{i=1}^k |r_i| \cdot |s_i| \cdot \frac{|A_i| - 1}{|A_i|} \\ &= \sum_{i=1}^k (120) \cdot \frac{|A_i| - 1}{|A_i|} \\ &= (120) \cdot (\frac{1}{2} + \frac{0}{1} + \frac{2}{3}) = 140 \end{aligned}$$

On the other hand, for *ED* method we can easily partition relations  $r$  and  $s$  into  $r_1, r_2, r_3$  and  $s_1, s_2, s_3$  respectively such that  $|A_1| = |A_2| = |A_3| = 2$ . Since we

Join domain values			(a)		(b)		(c)		(1)		(2)		(3)	
	r	s	r <sub>a</sub>	s <sub>a</sub>	r <sub>b</sub>	s <sub>b</sub>	r <sub>c</sub>	s <sub>c</sub>	r <sub>1</sub>	s <sub>1</sub>	r <sub>2</sub>	s <sub>2</sub>	r <sub>3</sub>	s <sub>3</sub>
1	6	4	6	4					6	4				
2	4	8	4	8					4	8				
3	10	12			10	12					10	12		
4	2	3					2	3			2	3		
5	5	3					5	3					5	3
6	8	2					8	2					8	2
			$\overline{10}$	$\overline{12}$	$\overline{10}$	$\overline{12}$	$\overline{15}$	$\overline{8}$	$\overline{10}$	$\overline{12}$	$\overline{12}$	$\overline{15}$	$\overline{13}$	$\overline{5}$

Table 3.3.3 The *EP* and *ED* partitions

know  $|r_1| = 10$ ,  $|r_2| = 12$ ,  $|r_3| = 13$  and  $|s_1| = 12$ ,  $|s_2| = 15$ ,  $|s_3| = 5$ , then

$$\begin{aligned}
 \text{MaxErr}_{ED}(|r \bowtie_A s|) &= \sum_{j=1}^k |r_j| \cdot |s_j| \cdot \left(1 - \frac{k}{|A|}\right) \\
 &= \left(\frac{1}{2}\right) \cdot \sum_{j=1}^k |r_j| \cdot |s_j| \\
 &= \left(\frac{1}{2}\right) \cdot (10 \cdot 12 + 12 \cdot 15 + 13 \cdot 5) = 182.5 > 140
 \end{aligned}$$

That is,  $\text{MaxErr}_{ED}(|r \bowtie_A s|) > \text{MaxErr}_{EP}(|r \bowtie_A s|)$ .

In this chapter we have shown that

$$\text{Err}_{UP}(|r \bowtie_A s|) \leq \frac{|r| \cdot |s| (|A| - 1)}{|A|}$$

It appears certain that

$$\text{Err}_P(|r \bowtie_A s|) \leq \text{Err}_{UP}(|r \bowtie_A s|)$$

even though we are only able to prove

$$\text{MaxErr}_P(|r \bowtie_A s|) \leq \text{MaxErr}_{UP}(|r \bowtie_A s|).$$

In the next chapter the A-partition method will be used again in the schemes of

monitoring the statistical parameters in dynamic database systems.

## CHAPTER 4

### Applications in a Dynamic Database

One of the primary objectives of this research is to develop general models that can be applied to real world applications. In Chapter 2 we have presented a complete set of formulae for computing the exact and expected size of a join. The maximum error of estimating the join size was analyzed in Chapter 3 and a practical approach to minimizing the maximum estimation error, called A-partitioning with equal size join domains was proposed. However, in order to make significant contributions, our system needs to function with minimum degradation in a dynamic environment. In some systems the database changes occur rapidly and in large volume. Therefore, a system that requires a high precision on the join size computation will need efficient schemes to calculate the correction term  $|A|c\delta_1\delta_2$  whenever necessary. Recall that  $c$  denotes the correlation coefficient and  $\delta_i$  denotes the standard deviation of a relation  $r_i$ . For systems requiring less accuracy, the correction term still can be used to bound the estimation error.

However, we know that computing the correction term for just one join attribute of large relations is expensive. Using frequency distributions of join attribute values directly to compute all statistical variables at run time is prohibitive in even a small dynamic database. Computing the correction terms for all joinable attributes in all relations is not practical without a dedicated resource. Our solution uses approximation methods to calculate the required parameters. The techniques of sampling, partitioning, and numerical approximation are employed in our approach. In the following discussion we assume that  $|r|$  and  $|A|$  are kept dynamically. For each join attribute  $A_i$  in  $r$  we will maintain the standard deviation  $\delta_i$  and the correlation coefficient  $c_{ij}$  for each relation  $r_j$  having the same



join attribute  $A_i$ .

#### 4.1. Maintaining $c$ and $\delta$ in a dynamic environment

In the initial step, normally while the relation is being created, we compute the expected join sizes and their correction terms. Depending on the availability of storage space and the accuracy requirement we use frequency tables (or grouped frequency tables) to compute  $c\delta_1\delta_2$  directly from source relations or from their sample sets. Frequency tables and grouped frequency tables (based on frequency distributions) will be discussed in 4.2 and sampling methods will be investigated in 4.3. As mentioned before, frequency tables can be used only in the systems with small join domains, or relations, because the cost of retrieving relations and maintaining corresponding frequency tables is expensive. But query optimization in small databases is seldom important. By their very nature one expects database relations and their join domains to be relatively large. Using sample sets instead of entire relations is more practical in general applications. In the following discussion we assume all the relations are obtained by sampling except in Scheme 3 where an A-partition approach is applied on the entire relation.

The maintenance step is invoked when the database is dynamically changed to such an extent that a new estimation of the parameters  $c$  and  $\delta$ , used to compute the join size, is required. We can use one of the following three schemes and repeat this step as often as necessary.

##### (Scheme 1) Completely recalculate $c$ and $\delta$ for the changed database.

Basically, the same procedures of the initial step are repeated here. New values of the required parameters are recomputed from newly sampled tuples. Since complete recomputation is expensive, and we expect no drastic change in either the sign or magnitude of the correlation coefficient after only a small number of

updates, it is wise to simply monitor the current values of  $c$  and  $\delta$  using the sampling techniques in 4.3. If significant discrepancies appear, then recalculation can be initiated.

Except for sample errors (which will be discussed in 4.3), this approach yields excellent estimates; but the processing cost is very expensive. First, one has to record all the relation names of those tuples whose join attribute values have been changed. Then, all corresponding joining relations need to be identified. They must participate in the recalculation process even there is no change in their tuples since the correlation coefficient  $c$  must be recalculated. That means all the join attribute frequency distributions of the related relations have to be accessed. One also needs to coordinate the sampling operations between relations with joinable (common) attributes in order to avoid sampling the same relation more than once. To accomplish that end, temporary storage space is needed to store the shared data and the complexity of the procedure is also increased.

One way to reduce I/O operations as much as possible and simplify the process is to perform the recalculation process for the updated relations only and to avoid the use of frequency distributions. In the next scheme we only recompute the standard deviations for the relations whose tuples have been updated on the join attributes. A numerical approach for computing the standard deviation is used instead of a frequency table.

**(Scheme 2) Assume  $c$  remains the same and only recalculate  $\delta$  for the correction term.**

If we assume the updates on join attribute values of a dynamic database are distributed uniformly over the corresponding domains, then changes on the regression lines and correlation coefficients of frequency distributions can be ignored unless the updates are drastic. In the latter case, we go back to the initial step. Therefore, in general we can assume the correlation coefficient  $c$  remains the same

and recalculate only the standard deviations  $\delta$  for relations whose tuples have been updated on the join attributes. Although frequency tables and grouped frequency tables can be used to compute  $\delta$ , the processing cost will still be too expensive. In Section 4.4 we will present a numerical method to approximate the standard deviation. The major advantage of this approach is that it avoids the use of frequency distributions. With new values of  $\delta$ ,  $|r_1|$ ,  $|r_2|$ , and  $|A|$ , we can compute the join size and correction term. Less frequently, we would recompute all the parameters again by using the procedures of the initial step.

Since the procedures in this scheme do not involve the frequency distribution and tuple retrieval, its processing cost is much less. However, the precision of estimating the join size is not as good as in Scheme 1 because the new frequency distribution of the join attribute values can not be reflected as accurately by calculating new  $\delta$  and retaining the old  $c$ . To improve the estimation accuracy, the next scheme uses an approach which is more capable of representing the actual distribution of the join attribute values by partitioning the join domains and relations into subsets in the initial step. Then the same procedures of this scheme are applied to each partition.

### (Scheme 3) Combination of an A-partition approach and Scheme 2.

The main idea of this scheme is the combination of the A-partition with equal size join domains and Scheme 2. As we have shown in the previous chapter, the maximum estimation error *MaxErr* of a join size can be reduced significantly if we partition the join attribute domains and source relations into subsets. One practical approach we use is to first partition the domain of join attribute  $A$  into equal size subsets, and then let this induce a second partition on source relations. Intuitively, the more partitions we have the smaller *MaxErr* we get. To use this scheme we A-partition the entire relation (not the sample set) into  $k$  subsets for all source relations at the beginning of the initial step. Then, the procedures of the initial

step are performed to produce the parameters for each of the  $k$  partitions. Of course, the sum of these  $k$  join sizes is the total size of the join. Now, in the maintenance step we use exactly the same procedures of Scheme 2 to estimate the join size for each of the  $k$  partitions in which the partition state has been changed. Periodically, we recompute the parameters for all partitions by using the procedures of the initial step.

In addition to having all the advantages of the second scheme, Scheme 3 provides a better estimation of the join size by using an A-partition. The only overhead is the processing cost of partitioning relations in the initial step and maintaining  $k$  sets of frequency data while using the database. In practice, Scheme 3 is much better since the extra cost is in proportion to the number of partitions which is at our discretion.

While an A-partition approach minimizes the "maximum" estimation error, the degree of improvement in the actual estimation error strictly depends on the actual frequency distribution of the join attribute values. If the attribute values are distributed in a relatively uniform manner, then the actual estimation error will be minimized. On the other hand, if the distribution is extreme, then the estimation error may be relatively large in some unusual distributions. For example, Table 4.1.1 shows an extreme case where the estimated error based on a partition of the join domain and relations  $r_1$  and  $r_2$  into subsets (b) and (c) is larger than the unpartitioned one. That is,  $Err_P(|r_1 \bowtie_A r_2|) > Err_{UP}(|r_1 \bowtie_A r_2|)$  as shown in the following

$$Err_{UP}(|r_1 \bowtie_A r_2|) = |r_1 \bowtie_A r_2| - \frac{|r_1| \cdot |r_2|}{|A|} = 30 - \frac{15 \cdot 15}{4} = -26.25$$

$$Err_P(|r_1 \bowtie_A r_2|) = 30 - \left( \frac{10 \cdot 10}{2} + \frac{5 \cdot 5}{2} \right) = -32.5$$

Here  $\mu_i$  denotes the frequency of the join attribute values in a relation  $r_i$ . In the example we see that the expected join sizes are greater than the real join sizes

Join Domain Values	(a)			(b)		(c)	
	$\mu_1$	$\mu_2$	$\mu_1\mu_2$	$\mu_1$	$\mu_2$	$\mu_1$	$\mu_2$
1	1	9	9	1	9		
2	9	1	9	9	1		
3	2	3	6			2	3
4	3	2	6			3	2
	15	15	30	10	10	5	5

Table 4.1.1 An A-partition of a skewed distribution

because the value of the correlation coefficient is negative ( $c = \frac{30 \cdot 4 - 15 \cdot 15}{(4)^2 \delta_1 \delta_2} = \frac{-105}{(4)^2 \delta_1 \delta_2}$ ). When the join domain and relations are partitioned such that the extreme values of the frequencies  $\mu_1$  and  $\mu_2$  are in the same subset, a larger join size will be expected because the join domain size (the denominator of the expected join size formula) of the partition is smaller than the unpartitioned one. This example shows why we could only prove assertions about the estimate of "maximal" error in Chapter 3. Normally, of course, partitioning decreases the expected estimation error.

## 4.2. Frequency tables

In this section we describe several methods for computing the correlation coefficient  $c$  and standard deviation  $\delta$  by using frequency tables. First, we introduce the construction of a frequency table and formulae for calculating  $c$  and  $\delta$ . Then, a grouped frequency table approach is shown for computing the standard deviation. A correction factor is also presented for the error derived from the grouping or rounding.

From the notation of Chapter 2 we use  $\mu_i$  to denote the frequency of the attribute value  $a \in D_A$  in a relation  $r_i$ . By using the standard definition [Mey65] of the standard deviation  $\delta$  and the result of Theorem 2.1.1 we have the following formulae:

$$\begin{aligned}\delta_i &= \left[ \frac{\sum_A \mu_i^2}{|A|} - \left( \frac{\sum_A \mu_i}{|A|} \right)^2 \right]^{1/2} = \frac{[|A| \cdot \sum_A \mu_i^2 - (\sum_A \mu_i)^2]^{1/2}}{|A|} \\ c &= \frac{\exp(\mu_1 \cdot \mu_2) - \exp(\mu_1) \cdot \exp(\mu_2)}{\delta_1 \delta_2} = \frac{\frac{\sum_A \mu_1 \mu_2}{|A|} - \frac{\sum_A \mu_1}{|A|} \cdot \frac{\sum_A \mu_2}{|A|}}{\delta_1 \delta_2} \\ &= \frac{|A| \cdot \sum_A \mu_1 \mu_2 - \sum_A \mu_1 \cdot \sum_A \mu_2}{|A|^2 \delta_1 \delta_2}\end{aligned}$$

To use the above formulae for computing  $\delta_i$  and  $c$ , the values of  $\sum_A \mu_1 \mu_2$ ,  $\sum_A \mu_i$  and  $\sum_A \mu_i^2$  must be known (here  $i = 1, 2$ ). A frequency table, as shown in Table 4.2.1 is constructed to collect these required frequency data for all attribute values in the domain of a join attribute  $A$ . In the following example we use a small attribute domain  $D_A = \{1, 2, 3, \dots, 14, 15\}$ .

From Table 4.2.1 we have  $\sum_A \mu_1 = 47$ ,  $\sum_A \mu_1^2 = 249$ ,  $\sum_A \mu_2 = 49$ ,  $\sum_A \mu_2^2 = 259$  and  $|r_1 \bowtie_A r_2| = \sum_A \mu_1 \mu_2 = 124$ . Then

$$\begin{aligned}\delta_1 &= \frac{[|A| \cdot \sum_A \mu_1^2 - (\sum_A \mu_1)^2]^{1/2}}{|A|} = \frac{(15 \cdot 249 - 47 \cdot 47)^{1/2}}{15} \\ &= \frac{(3735 - 2209)^{1/2}}{15} = \frac{(1526)^{1/2}}{15} = 2.604 \\ \delta_2 &= \frac{[|A| \cdot \sum_A \mu_2^2 - (\sum_A \mu_2)^2]^{1/2}}{|A|} = \frac{(15 \cdot 259 - 49 \cdot 49)^{1/2}}{15} \\ &= \frac{(3885 - 2401)^{1/2}}{15} = \frac{(1484)^{1/2}}{15} = 2.568\end{aligned}$$

and

---

Attribute values	$\mu_1$	$\mu_1^2$	$\mu_2$	$\mu_2^2$	$\mu_1\mu_2$
1	2	4	10	100	20
2	4	16	3	9	12
3	0	0	2	4	0
4	3	9	4	16	12
5	5	25	0	0	0
6	10	100	1	1	10
7	6	36	5	25	30
8	3	9	2	4	6
9	2	4	5	25	10
10	0	0	6	36	0
11	2	4	5	25	10
12	4	16	1	1	4
13	5	25	2	4	10
14	0	0	3	9	0
15	1	1	0	0	0
Total	47	249	49	259	124

---

Table 4.2.1 A frequency table

---


$$c = \frac{|A| \cdot \sum_A \mu_1 \mu_2 - \sum_A \mu_1 \cdot \sum_A \mu_2}{|A|^2 \delta_1 \delta_2} = \frac{(15)(124) - (47)(49)}{(225)(2.604)(2.568)}$$

$$= \frac{-443}{1504.591} = -0.294$$

To illustrate the use of these correction terms, we compute the exact join size as follows

$$|r_1 \bowtie_A r_2| = \frac{|r_1| \cdot |r_2|}{|A|} + |A|c\delta_1\delta_2 = \frac{47 \cdot 49}{15} + (15)(-0.294)(2.604)(2.568)$$

$$= 153.533 - 29.489 = 124.044 \equiv 124$$

A less expensive method of estimating the standard deviation by using the group data, called a grouped frequency table, is shown in the next example. In Table 4.2.2 we partition the frequency  $\mu_1$  of the previous example into six groups

with a fixed group interval  $h = 2$ . For example, group 1 has four frequencies ( $f$ ) in the range of 0 to 1, i.e., three 0's and one 1. The midpoint ( $x$ ) of group 1 is 0.5. The origin ( $g$ ) is the midpoint of group 3 such that the deviation ( $d$ ), in group intervals, of that group is 0. Here  $d = \frac{x - g}{h}$ . Let  $n$  denote the total number of groups. Then

$$\begin{aligned}\delta &= \left[ \frac{\sum_n f(x - g)^2}{|A|} - \left( \frac{\sum_n f(x - g)}{|A|} \right)^2 \right]^{1/2} = \left[ \frac{\sum_n f(dh)^2}{|A|} - \left( \frac{\sum_n f(dh)}{|A|} \right)^2 \right]^{1/2} \\ &= h \cdot \left[ \frac{\sum_n f(d)^2}{|A|} - \left( \frac{\sum_n f(d)}{|A|} \right)^2 \right]^{1/2} = \frac{h \cdot [ |A| \cdot \sum_n f(d)^2 - (\sum_n f d)^2 ]^{1/2}}{|A|}\end{aligned}$$

From Table 4.2.2 we know  $\sum_n f(d)^2 = 31$  and  $\sum_n f d = 9$ . Then

$$\begin{aligned}\delta_1 &= \frac{h \cdot [ |A| \cdot \sum_n f(d)^2 - (\sum_n f d)^2 ]^{1/2}}{|A|} = \frac{2 \cdot (15 \cdot 31 - 9 \cdot 9)^{1/2}}{15} \\ &= \frac{2 \cdot (384)^{1/2}}{15} = 2.612 \equiv 2.604\end{aligned}$$

---

$\mu_1$	$f$	$x$	$d = \frac{x - g}{h}$	$d^2$	$f d$	$f d^2$
0-1	4	0.5	-2	4	-8	16
2-3	5	2.5	-1	1	-5	5
4-5	4	4.5	0	0	0	0
6-7	1	6.5	1	1	1	1
8-9	0	8.5	2	4	0	0
10-11	1	10.5	3	9	3	9
Total	15				9	31

---

Table 4.2.2 A grouped frequency table

---



Although in this example the estimation of the standard deviation from a grouped frequency table is very good, in general we must select a group interval  $h$  such that the estimated  $\delta \geq 4h$  and the total number of groups  $n \geq 20$  [Sne56, p.196] [Win75, p.289]. For a bell-shaped distribution (e.g., normal distribution), the assumption that the data are located at the midpoint of each group is not correct because most data in the group will be closer to the higher point. The error resulted from grouping or rounding can be corrected by Sheppard's correction [Bur70, p.78]

$$\delta^2 = (\delta_{group})^2 - \frac{h^2}{12}$$

#### 4.3. Random sampling approaches

Since the cost of examining the join attribute values for all tuples in all relations is too expensive, we use an approximation approach which processes only a randomly selected subset of tuples from each relation. The selected subset is called a "sample". A sampling method selecting each of its members with an equal probability is called a "random sampling". More detailed discussions about sampling will be presented in Section 5.1.4. By using a sample of tuples, instead of entire population of a relation, we lose the exact precision. However, according to Kolmogorov's non-parametric statistics [Wal53] a very good degree of accuracy can still be maintained by using appropriate sample sizes.

Let  $d$  denote the maximum deviation between the cumulative distributions of the population and of the sample. In [Dix69, p.550] it is shown that for a maximum deviation  $d$  less than 0.20, the required sample sizes for the confidence levels 99%, 95%, 90%, 85% and 80% are  $(\frac{1.63}{d})^2$ ,  $(\frac{1.36}{d})^2$ ,  $(\frac{1.22}{d})^2$ ,  $(\frac{1.14}{d})^2$  and  $(\frac{1.07}{d})^2$  respectively. For example, to get a maximum deviation of 0.05 with 95%

of confidence level we need a sample of size

$$\left(\frac{1.36}{d}\right)^2 = \left(\frac{1.36}{0.05}\right)^2 = 739.84 \equiv 740$$

That is, if we have a sample relation  $s$  with 740 tuples selected randomly from a population relation  $r$  ( $|r| > 740$ ), then with a confidence level of 95% we know the probability of a tuple with the attribute value being less than a given domain value  $a_i \in D_A$  in the relation  $s$  is within  $\pm 0.05$  of the probability for the relation  $r$ , or equivalently

$$p_r(A < a_i) = p_s(A < a_i) \pm 0.05$$

A sample of 740 tuples may appear large. But in a typical database files (relations) of more than 20,000 tuples are not unusual. It is in such "large" databases that the advantage of the random sampling approach can be fully exploited. A partial list of the required sample sizes for  $d \leq 0.20$  will be shown in Table 5.1.4.1. Since we are interested in exact match probabilities for join operations, we need to derive the maximum deviation between the probabilities of a tuple satisfying ( $A = a_i$ ) in the population relation  $r$  and in the sample relation  $s$ .

Figure 4.3.1 illustrates a cumulative distribution of attribute value frequencies for attribute  $A$  in a relation. Here, the probability that a tuple has attribute value for  $A$  less than a given value  $a_i$  is denoted by  $p(A < a_i)$ . Similarly, for the next domain value  $a_{i+1}$  the probability is  $p(A < a_{i+1})$ . Note that we assume the attribute values of the domain  $D_A$  are ordered.

Then we know

$$p(A = a_i) = p(A < a_{i+1}) - p(A < a_i)$$

Since Kolmogorov's non-parametric method shows that

$$p_r(A < a_i) = p_s(A < a_i) \pm d$$

and

$$p_r(A < a_{i+1}) = p_s(A < a_{i+1}) \pm d$$

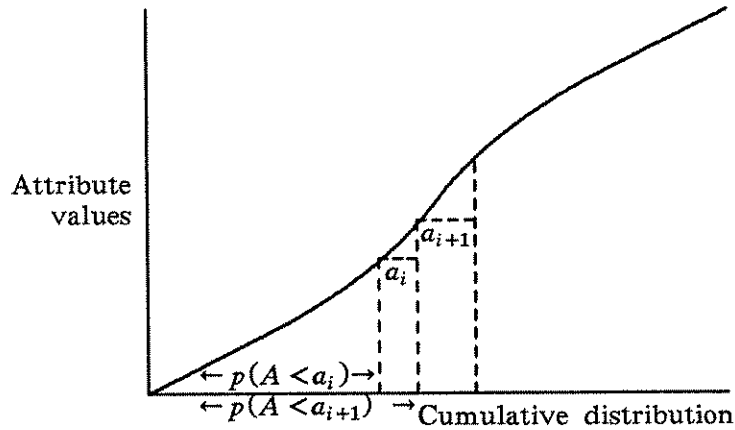


Figure 4.3.1

Readily,

$$p_r(A = a_i) = [p_s(A < a_{i+1}) - p_s(A < a_i)] \pm 2d$$

because the maximum value of  $p_r(A = a_i)$  is

$$[p_s(A < a_{i+1}) + d] - [p_s(A < a_i) - d] = [p_s(A < a_{i+1}) - p_s(A < a_i)] + 2d$$

and the minimum value of  $p_r(A = a_i)$  is

$$[p_s(A < a_{i+1}) - d] - [p_s(A < a_i) + d] = [p_s(A < a_{i+1}) - p_s(A < a_i)] - 2d.$$

Since the new maximum deviation is double for  $p(A = a_i)$ , the required sample sizes for the confidence levels 99%, 95%, 90%, 85% and 80% are  $(2 \cdot \frac{1.63}{d})^2$ ,  $(2 \cdot \frac{1.36}{d})^2$ ,  $(2 \cdot \frac{1.22}{d})^2$ ,  $(2 \cdot \frac{1.14}{d})^2$  and  $(2 \cdot \frac{1.07}{d})^2$  respectively. Again, we consider the previous example. To get a maximum deviation of 0.05 with 95% of confidence level, we need a sample of size

$$(2 \cdot \frac{1.36}{d})^2 = (\frac{1.36}{0.025})^2 = 2959.36 \equiv 2960$$

Therefore, from the above discussion we can determine the sample size according to the desired accuracy and available storage space. If the sample size is

too small, then a correction factor for estimating the standard deviation, called the Bessel correction [Lea74, p.28] can be used to make a better estimation. That is,

$$\delta_{population} = \delta_{sample} \left( \frac{n}{n-1} \right)^{1/2}$$

where  $n$  is the sample size. Note that as  $n$  increases this correction factor approaches unity. Normally, this correction can be ignored when the sample size  $n$  is greater than 30 [Lon70, p.68]. However, if we know that the attribute value frequencies are normally distributed, then a better correction factor can be used [Dix69, p.136]

$$\delta_{population} = \delta_{sample} \left[ 1 + \frac{1}{4 + (n-1)} \right]$$

Similarly, when the sample size  $n$  is small, we need a correction factor for the correlation coefficient as follows [Lon70, p.232]

$$c_{population} = c_{sample} \left( \frac{n}{n-2} \right)^{1/2}$$

This correction factor can be used for any frequency distribution.

#### 4.4. Estimating standard deviations

Since the processing cost of using frequency tables is very expensive, in this section we present numerical approximation methods for computing the standard deviation after the database state has been changed. Now we use  $\mu_k$  to denote the frequency of the attribute value  $a_k \in D_A$  in a relation. The main idea of this approach is to approximate the summation of the square of all the  $\mu$  without the use of frequency distributions. As in general approximation methods one can make a better estimate when the frequency distribution is approximately uniform. Our estimation error will be minimized if the attribute values are uniformly distributed. To simplify our notation for a clear presentation, the subscripts of the variables in a summation term are omitted.

We know that the standard deviation  $\delta$  is  $\left[ \frac{\sum_A \mu^2}{|A|} - \left( \frac{\sum_A \mu}{|A|} \right)^2 \right]^{1/2}$ . Therefore, to estimate the standard deviation for a new database state we approximate new values of  $\sum_A \mu$  and  $\sum_A \mu^2$  using knowledge of the dynamic updates to modify their existing values. Let  $n_i$  denote the number of new tuples (additions) with attribute value  $a_i$  and let  $m_j$  denote the number of deleted tuples with attribute value  $a_j$ . The total numbers of additions and deletions are denoted by  $N = \sum_A n$  and  $M = \sum_A m$  respectively. The change of attribute value for a tuple from  $a_j$  to  $a_i$  is represented by adding one to both  $m_j$  and  $n_i$ , i.e., a change is treated as a deletion followed by an addition. Since we know the total frequency  $\sum_A \mu$  is the relation size, the new value of  $\sum_A \mu$  (denoted by  $\sum_A \mu_{new}$ ) is the following

$$\sum_A \mu_{new} = \sum_A \mu + \sum_A n - \sum_A m = \sum_A \mu + N - M \quad (4.4.1)$$

Depending on the types of update and the desired accuracy, two methods can be used to estimate the new summation of the square of the frequencies  $\mu$ , i.e.,  $\sum_A (\mu_{new})^2$ . The first method is designed for the estimation where the numbers of updates  $n$  and  $m$  are not small and two or more updates may be on the same attribute value. Then

$$\begin{aligned} \sum_A (\mu_{new})^2 &= (\mu_1 + n_1 - m_1)^2 + \dots + (\mu_k + n_k - m_k)^2 + \dots + (\mu_{|A|} + n_{|A|} - m_{|A|})^2 \\ &= \mu_1^2 + 2\mu_1(n_1 - m_1) + (n_1 - m_1)^2 + \dots + \mu_k^2 + 2\mu_k(n_k - m_k) + (n_k - m_k)^2 + \dots \\ &= (\mu_1^2 + \mu_2^2 + \dots + \mu_k^2 + \dots) + \sum_A 2\mu(n - m) + \sum_A (n - m)^2 \\ &= \sum_A \mu^2 + 2 \sum_A \mu(n - m) + [\sum_A n^2 + \sum_A m^2 - 2 \sum_A (n \cdot m)] \end{aligned}$$

We use the expected value  $\bar{\mu} = \frac{\sum_A \mu}{|A|}$  to estimate each  $\mu$  of the second term, i.e.,

$$\begin{aligned} \sum_A (\mu_{new})^2 &= \sum_A \mu^2 + 2 \sum_A \bar{\mu}(n - m) + [\sum_A n^2 + \sum_A m^2 - 2 \sum_A (n \cdot m)] \\ &= \sum_A \mu^2 + 2\bar{\mu}(\sum_A n - \sum_A m) + [\sum_A n^2 + \sum_A m^2 - 2 \sum_A (n \cdot m)] \end{aligned}$$

$$= \sum_A \mu^2 + 2\bar{\mu}(N-M) + [\sum_A n^2 + \sum_A m^2 - 2\sum_A (n \cdot m)] \quad (4.4.2)$$

Although the frequency distribution is not required in the formula (4.4.2) for computing  $\sum_A (\mu_{new})^2$ , one still has to record all the frequencies of updated attributes  $n_i$  and  $m_j$  in order to compute  $\sum_A n^2$ ,  $\sum_A m^2$  and  $\sum_A (n \cdot m)$ . Therefore, the processing cost is proportion to the total numbers of updates  $N$  and  $M$ . And the required storage space depends on the number of unique attribute values updated.

In the second method, we minimize the processing cost and storage overhead by eliminating the need for recording the frequencies  $n_i$  and  $m_j$  for each updated attribute value  $a_i$ . The best application of this method is for the case when all the attribute values of the updated tuples are different. Since in our schemes the updates on the attribute values of the relation are presumed to be uniformly distributed over the attribute domain, it is reasonable to assume that no two updates have the same attribute value, i.e.,  $n_i$  and  $m_j$  are either 1 or 0. Then,  $\sum_A n^2 = \sum_A n$  and  $\sum_A m^2 = \sum_A m$  because  $(n_i)^2 = n_i$  and  $(m_j)^2 = m_j$ . Since  $n_i$  and  $m_i$  can not both have the same value 1, we know that  $n_i \cdot m_i = 0$  and  $\sum_A (n \cdot m) = 0$ . Then the new value of  $\sum_A \mu^2$  for the second method is the following

$$\begin{aligned} \sum_A (\mu_{new})^2 &= \sum_A \mu^2 + 2\bar{\mu}(N-M) + [\sum_A n^2 + \sum_A m^2 - 2\sum_A (n \cdot m)] \\ &= \sum_A \mu^2 + 2\bar{\mu}(N-M) + [\sum_A n + \sum_A m - 0] \\ &= \sum_A \mu^2 + 2\bar{\mu}(N-M) + N + M \end{aligned} \quad (4.4.3)$$

Obviously, the processing cost of the second method is much less since it need not record any frequency of the updated attribute value  $n_i$  or  $m_j$ . Instead, only the total number of updates  $N$  and  $M$  are required.

From these three expressions (4.4.1 to 4.4.3) we can compute the new values of  $\sum_A \mu^2$  and  $\sum_A \mu$ . Then, the standard deviation for the new database state is

$$\delta_{new} = \left[ \frac{\sum_A (\mu_{new})^2}{|A|} - \left( \frac{\sum_A \mu_{new}}{|A|} \right)^2 \right]^{1/2}$$

## CHAPTER 5

### Expected Size of Selection

#### 5.1. Probability of satisfying a selection function

To compute the expected size of selection is to estimate the fraction of tuples selected from a relation, i.e., the probability that an arbitrary tuple will satisfy a selection function. The basic form of a selection function can be represented as a term of the form  $(A \text{ comp } a)$ , where  $A$  denotes any attribute, " $a$ " is any value in its domain and  $\text{comp}$  is a comparison operator such as  $\{<, =, >, \leq, \neq, \geq\}$ . For example,  $(\text{age} > 20)$  is a selection function. In [Car75] it is called an atomic condition.

Although  $(A \text{ comp } B)$  is a more general form for selection functions, in theory it can be decomposed into the basic form  $(A \text{ comp } b_i)$  for all attribute values  $b_i$  in the domain of an attribute  $B$ ,  $i = 1, 2, \dots, n$ . For example, the selection function  $(A > B)$  can be decomposed into the following form:

$$(B = b_1 \wedge A > b_1) \vee \dots$$

$$(B = b_i \wedge A > b_i) \vee \dots$$

$$(B = b_n \wedge A > b_n)$$

Here  $\wedge$  (AND) and  $\vee$  (OR) are logical operators used to form compound selection functions. Later in this chapter we will formally define the selection function used in the relational database and examine the above compound selection forms. In this section we analyze the basic selection form only.

We use  $\text{prob}(A \text{ comp } a)$  to denote the probability that any given tuple in a relation  $r$  with scheme  $R$  and attribute  $A \in R$  satisfies the selection function  $(A \text{ comp } a)$ . In general,  $a$  can be any value. For notational convenience and clear



presentation we assume that the attribute domain  $D_A$  is ordered and  $a \in D_A$ . If the value of  $a$  is not in  $D_A$ , then an equivalent selection function ( $A \text{ comp}^* a^*$ ) can be derived such that  $\text{prob}(A \text{ comp } a) = \text{prob}(A \text{ comp}^* a^*)$  and  $a^* \in D_A$ . For example, consider an attribute domain  $D_A = \{a_1, a_2, \dots, a_i, a_{i+1}, \dots, a_n\}$  and  $a_i < a < a_{i+1}$ . We know that  $(A \leq a_i)$  is equivalent to  $(A < a)$  because  $\text{prob}(A \leq a_i) = \text{prob}(A < a)$ . There are two exceptions. If the comparison operator  $\text{comp}$  is  $\neq$  and  $a \notin D_A$ , then  $\text{prob}(A \neq a) = 1$ . Similarly, if the  $\text{comp}$  is  $=$  and  $a \notin D_A$ , then  $\text{prob}(A = a) = 0$ . Four different methods of computing  $\text{prob}(A \text{ comp } a_i)$  will be discussed in the following subsections: simple probabilistic approaches, attribute distribution tables, grouped frequency schemes, and random sampling methods.

### 5.1.1. Simple probabilistic approaches

Let  $D_A = \{a_1, a_2, \dots, a_n\}$  denote the domain of an attribute  $A$ , where  $a_1 < a_2 < \dots < a_n$ . In this "simple" approach we assume that the distribution of the attribute values is approximately "uniform" over the range  $[a_1, a_n]$ . Then

$$\begin{aligned}\text{prob}(A = a_i) &= \frac{1}{n} \\ \text{prob}(A < a_i) &= \frac{i - 1}{n} \\ \text{prob}(A > a_i) &= \frac{n - i}{n}\end{aligned}$$

Although this is a very simple method, the cost of storing and retrieving large attribute domains to locate the value of  $i$  may be very high. However, if the attribute value distribution is "linear" as well as "uniform", then we need not search the domain  $D_A$  to find an  $i$  value. Instead, we can compute the above probabilities by using  $n$ ,  $a_1$  and  $a_n$ . First, we compute the average interval between two consecutive attribute values  $\Delta = \frac{a_n - a_1}{n - 1}$ . Then

$$\text{prob}(A = a_i) = \frac{\Delta}{a_n - a_1 + \Delta} = \frac{1}{n}$$

$$\text{prob}(A < a_i) = \frac{a_i - a_1}{a_n - a_1 + \Delta}$$

$$\text{prob}(A > a_i) = \frac{a_n - a_i}{a_n - a_1 + \Delta}$$

In System R [Sel79, p.26] a similar approach is used by its OPTIMIZER to assign a selectivity factor for each selection function. Note that the second set of the above formulae can be applied for numerical attribute values only. In most applications the attribute distributions are neither uniform nor linear. Therefore, the estimation errors can be extremely large for certain attribute distributions. However, this is a very simple and low cost approach when the distribution of the attribute values is uniform and linear.

### 5.1.2. Attribute distribution tables

To have a better estimation of the selection size, the distribution of the selection attribute values must be taken into account. One approach is to record the number of tuples having the attribute value  $a_i$  for every  $a_i$  in the domain  $D_A$ . We call a table containing the frequencies of the attribute values in a relation an **attribute distribution table**. The basic form of this table includes the frequencies for each of the attribute values only.

For example, in Table 5.1.2.1 (a) we are given five entries of the frequencies  $f(a_i)$  for a small attribute domain  $D_A = \{a_1, a_2, a_3, a_4, a_5\}$ . Intuitively, for a relation  $r$  with the selection attribute domain  $D_A = \{a_1, \dots, a_i, \dots, a_n\}$  and  $a_1 < \dots < a_i < \dots < a_n$  we know that

$$\text{prob}(A = a_i) = \frac{f(a_i)}{|r|}$$

$$\text{prob}(A < a_i) = \frac{\sum_{j=1}^{i-1} f(a_j)}{|r|}$$

---

	(a)	(b)	(c)
Attribute Values	$f(a_i)$	$f^-(a_i)$	$f^+(a_i)$
	<hr/>	<hr/>	<hr/>
$a_1$	5	5	40
$a_2$	10	15	35
$a_3$	2	17	25
$a_4$	8	25	23
$a_5$	15	40	15

---

Table 5.1.2.1 An attribute distribution table

$$\text{prob}(A > a_i) = \frac{\sum_{j=i+1}^n f(a_j)}{|r|}$$

To avoid the repeated computation of the summation, we can include the cumulative frequencies in an attribute distribution table. In Table 5.1.2.1 (b) and (c) two cumulative columns in the top-down and bottom-up directions are displayed respectively. Here,  $f^-(a_i)$  denotes the top-down cumulation, i.e.,  $f^-(a_i) = \sum_{j=1}^i f(a_j)$  for  $1 \leq i \leq n$ . Similarly, the bottom-up cumulative column is denoted by  $f^+(a_i)$  and defined as  $f^+(a_i) = \sum_{j=i}^n f(a_j)$  for  $1 \leq i \leq n$ .

Since the storage requirement and retrieval cost for a large attribute distribution table are expensive, a complete table including all three columns  $f$ ,  $f^-$ , and  $f^+$  is impractical in general applications. Actually, one cumulative column is all we need. The other two columns can be derived easily from that one. From  $f^-(a_i)$  we know that

$$f(a_1) = f^-(a_1)$$

$$f(a_i) = \sum_{j=1}^i f(a_j) - \sum_{j=1}^{i-1} f(a_j) = f^-(a_i) - f^-(a_{i-1}) \quad 1 < i \leq n$$

$$f^+(a_1) = \sum_{j=1}^n f(a_j) = f^-(a_n)$$

$$f^+(a_i) = \sum_{j=1}^n f(a_j) - \sum_{j=1}^{i-1} f(a_j) = f^-(a_n) - f^-(a_{i-1}) \quad 1 < i \leq n$$

Similarly, we can derive  $f(a_i)$  and  $f^-(a_i)$  from  $f^+(a_i)$  :

$$f(a_n) = f^+(a_n)$$

$$f(a_i) = f^+(a_i) - f^+(a_{i+1}) \quad 1 \leq i < n$$

$$f^-(a_n) = f^+(a_1)$$

$$f^-(a_i) = f^+(a_1) - f^+(a_{i+1}) \quad 1 \leq i < n$$

### 5.1.3. Grouped frequency schemes

Due to the large space required to store the list of all attribute value frequencies in the selection domains (e.g., social security numbers, ages), we may divide the range of attribute values into  $n$  groups. In this method we record the frequencies of the attribute values in terms of groups. Therefore, for each selection attribute  $A$  only  $n$  frequency entries needed to be stored. The saving is substantial when  $|A|$  is much greater than  $n$ .

A well known graphical representation of a grouped frequency scheme is called a **histogram** [Sen56, p.16]. Normally, the attribute domain in a histogram is divided into  $n$  equal size subdomains. For example, Figure 5.1.3.1 shows an equal-width histogram on the attribute *Score* for a frequency distribution of 100 students. Here, the scores are divided equally into five intervals of size 20. The label on the  $x$  axis denotes the midpoint of each group.

Using a histogram to compute  $\text{prob}(A \text{ comp } a)$  is very cheap since in general applications  $|A| \gg n$ . However, the precision is not always good in an equal-width histogram because the estimation error of  $\text{prob}(A < a)$  may be as large as half the height of the group where  $a$  belongs to. For example,  $0 \leq \text{Err}[\text{prob}(\text{Score} < 75)] \leq \frac{0.40}{2}$ . In the worst cast where most of the attribute values in a relation fall into one group, then the maximum estimation error

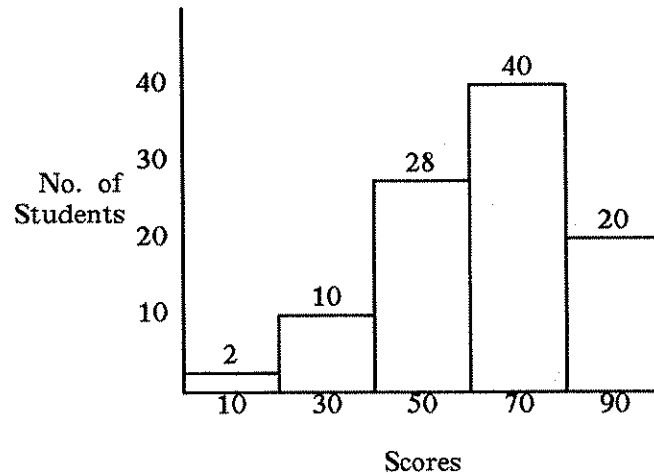


Figure 5.1.3.1 A histogram

*MaxErr* may reach 50%.

One way to reduce the maximum estimation error is to lower the height of each group by using an equal-height grouping instead of equal-width. From the previous example, we construct an equal-height histogram in Figure 5.1.3.2. Here, the attribute domain of *Score* is divided into five groups such that the number of students in each group is fixed (i.e., 20). For example, the range of scores for the first group of 20 students is (0, 45). Since the height of each group is the same, we have a guaranteed maximum estimation error for the selection function ( $A < a$ ) where  $a$  falls between the range of any group interval. In fact,  $MaxErr[prob(A < a)] = \frac{1}{2n}$ , where  $n$  is the total number of groups. From Figure 5.1.3.2 we know the maximum estimation error is  $\frac{1}{(2)(5)} = 10\%$ .

A complete procedure for creating an equal-height histogram, called Distribution Steps is presented in [Pia84]. We will give a brief description in the following paragraphs since it provides a highly accurate estimate of the selection size with a controllable upper bound of the estimation error.

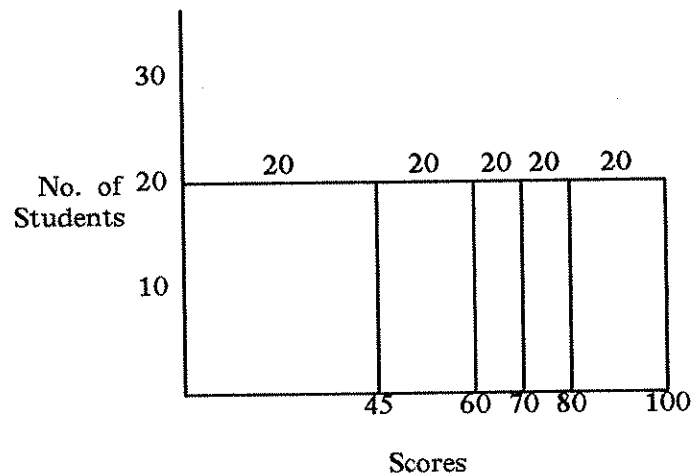


Figure 5.1.3.2 An equal-height Histogram

For a selection attribute  $A$  in a relation  $r$ , the distribution step  $Step(k)$  with  $k = 1, 2, \dots, s$  is defined as the attribute value in the ordered domain  $D_A = \{a_1, a_2, \dots, a_n\}$  such that  $prob(A < Step(k)) \leq \frac{k}{s}$ . This means the attribute domain  $D_A$  is divided into  $s$  groups (steps) in such a way that the number of actual attribute values in each group is the same. Therefore,  $Step(k)$  is the upper bound of the attribute values in the step  $k$ .

To compute distribution steps  $Step(1), \dots, Step(s)$ , we need to sort the attribute values of  $A$  from all the tuples in a relation  $r$  into ascending order. Next we determine the number of distribution steps  $s$  according to the desired precision. Then the sorted list of attribute values is divided into  $s$  groups such that each group has the same number of attribute values. Finally, we assign the upper bound of attribute values in each group to the corresponding distribution step  $Step(k)$  for  $k = 1, 2, \dots, s$ .

Using the data of the previous example, we construct Figure 5.1.3.3 to show a scheme of 10 distribution steps. Here, the values of distribution steps  $Step(k)$  are shown at the top of each of the dashed lines. For example, let us compute the

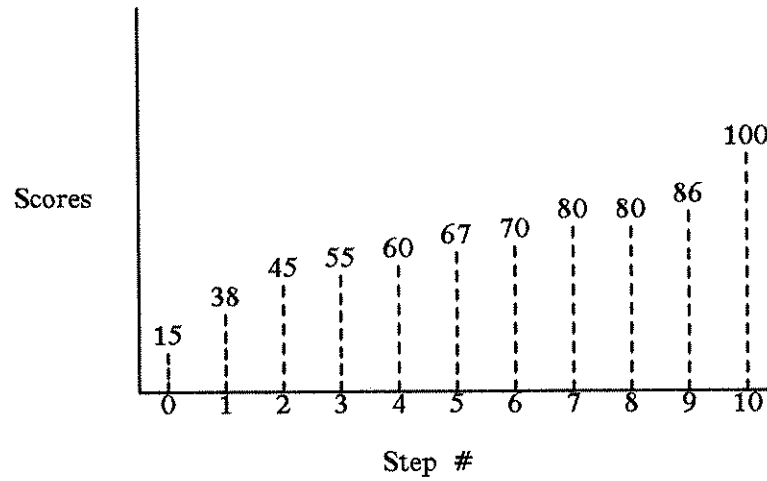


Figure 5.1.3.3 Distribution Steps

probability of a tuple satisfying the selection function ( $Score < 75$ ). We know that  $0.60 \leq \text{prob}(Score < 75) \leq 0.70$  because  $\text{prob}(Score < 70) \leq 0.60$  and  $\text{prob}(Score < 80) \leq 0.70$ . By using the average value 0.65 as the expected  $\text{prob}(Score < 75)$ , we know the maximum estimation error  $MaxErr$  is 5%. Note that the 50% reduction of  $MaxErr$  from 10% in Figure 5.1.3.2 down to 5% in Figure 5.1.3.3 is due to the increase of distribution steps (from 5 to 10). That means the maximum estimation error can be reduced to any desired small value by using a sufficiently large  $s$ .

A complete set of formulae that are optimized to reduce the maximum estimation error for the selection function ( $A \text{ comp } a$ ) can be found in [Pia84, pp.264-270].

#### 5.1.4. Random sampling methods

Our goal is to "accurately" estimate the selection size  $|\sigma_{A \text{ comp } a}(r)|$  for a relation  $r$  in a dynamic database and to do so "effectively". The methods discussed in the previous subsections are either distribution dependent or too expensive.

Simple probabilistic approaches are most suitable for uniform distributions of the attribute values and the cost of (grouped) frequency methods is rather high in terms of the I/O operations and storage space. One solution is to perform selection operations on a sample of the tuples in a relation, rather than on the entire relation. Although we lose guaranteed precision, a high degree of accuracy still can be achieved.

A **sample** consists of a small collection from some large population (or universe). Information obtained from a sample or a set of samples can reflect the characteristics and statistical parameters of the population rather faithfully. A sampling method in which each member of the population has an equal chance to be included in the sample is called a **random sampling method** [Lon70]. To analyze the estimation error due to the variation of sampling techniques, we use a distribution-free method called non-parametric statistics [Dix69]. Kolmogorov-Smirnov's test [Wa153] shows that the cumulative percentage distributions for two samples randomly selected from the same population should resemble each other, and should resemble the parent distribution.

For example, if we take 266 tuples from a relation  $r$  ( $|r| > 266$ ) to form a sample relation  $s$ , then with confidence 99% we know

$$prob_r(A < a) = prob_s(A < a) \pm 0.10$$

From Table 5.1.4.1 (excerpted from [Dix69, p.550]), we can see that the maximum deviation will drop to 0.05 (see row (3)) if the size of the sample relation is increased to 1064 tuples. If, instead, we decrease the sample size to 185 tuples, then the confidence level will drop to 95% (column (b)) with the same maximum deviation 0.10. Note that these estimates are independent of  $|r|$ .

Given the values of the maximum deviation and confidence level, we can find the sample size at the intersection of corresponding row and column of Table 5.1.4.1. Confidence levels and maximum deviations are functions of sample size,



---

Maximum deviation	Confidence levels		
	(a) 90%	(b) 95%	(c) 99%
(1) .20	35	45	67
(2) .10	149	185	266
(3) .05	596	740	1,064
(4) .025	2,348	2,960	4,256
(5) .01	14,880	18,500	26,570

---

Table 5.1.4.1 Sample sizes

but independent of population size. That means we can have a bounded estimation error with a specified confidence level no matter how large the size of a population relation is.

To illustrate the potential of random sampling, we use an exponential distribution  $f(t) = \alpha e^{-\alpha t}$  of a selection attribute "life length" for a randomly generated relation containing 9,467 components in a manufacturing company. The attribute domain is assumed from 1 to 69 time units. A sample relation is then created by randomly selecting 1064 tuples from the entire relation. In Table 5.1.4.2, we show 20 distribution steps for both the entire relation and sample relation. The attribute values of the corresponding steps in two relations are very close. The average deviation is only 0.75.

To show that the maximum deviation of our random sampling method is within 0.05, we use five selection functions  $prob(A < a_i)$  where  $a_1 = 5$ ,  $a_2 = 10$ ,  $a_3 = 15$ ,  $a_4 = 25$ , and  $a_5 = 45$ . The results for both the frequency distribution table and the distribution step are shown in Table 5.1.4.3. The largest deviation

Step no.	Entire relation	Sample relation	Absolute deviation	Step no.	Entire relation	Sample relation	Absolute deviation
1	1	1	0	11	8	8	0
2	2	2	0	12	10	9	1
3	2	2	0	13	11	10	1
4	3	3	0	14	12	12	0
5	3	3	0	15	14	13	1
6	4	4	0	16	16	15	1
7	5	5	0	17	19	18	1
8	6	5	1	18	23	22	1
9	6	6	0	19	30	28	2
10	7	7	0	20	69	63	6

Table 5.1.4.2 Distribution step values

from the actual frequency distribution table is 0.0205 while the largest one for 20 distribution steps is 0.05 as expected.

In a runtime system any of the preceding four methods can be used to estimate  $\text{prob}(A \text{ comp } a)$  and thus  $|\sigma_{A \text{ comp } a}(r)| = |r| \cdot \text{prob}(A \text{ comp } a)$ . In general,

Selection function	(Frequency table)			(Distribution steps)		
	Actual <i>prob</i>	Sample <i>prob</i>	Absolute deviation	Population <i>prob</i>	Sample <i>prob</i>	Absolute deviation
$A < 5$	0.3308	0.3440	0.0132	0.2250	0.2250	0.0000
$A < 10$	0.5954	0.6156	0.0202	0.5750	0.6250	0.0500
$A < 15$	0.7558	0.7763	0.0205	0.7666	0.7750	0.0084
$A < 25$	0.9118	0.9229	0.0111	0.9166	0.9166	0.0000
$A < 45$	0.9896	0.9972	0.0075	0.9666	0.9666	0.0000

Table 5.1.4.3 Maximum deviation  $\leq 0.05$

the choice depends on the cost of maintenance and the estimation accuracy. Although the estimation error may be extremely large for non-uniform attribute distributions, probabilistic methods offer a simple and low cost approach. Attribute distribution tables and (grouped) frequency schemes have large overheads on storage space and I/O operations. Despite the fact that estimation accuracy is good for these two methods, the cost of maintaining the frequency parameters in a dynamic database is rather high. Generally, random sampling is a much better approach since it can be used for any attribute distributions and processing cost as well as estimation accuracy is proportional to the sample size.

While various methods may be used to estimate the expected size of a basic selection term, little work has been done on the general formulae for computing the compound selection size. In the next section we present a generalized model on the expectation of compound selections.

## 5.2. Compound selections

In real applications, a general query normally involves more than one attribute. Later in the next chapter we will examine the performance of a join by decomposing the operation into two selection operations on the same attribute. In both cases, our interests are in the combination of basic selection terms, which we call the **compound selection**. In the following, we first give a complete description of the selection function in the relational database.

We assume the basic selection criterion is a well formed formula in the predicate calculus which is a conjunction of terms involving one or more attributes of a relation. Without loss of generality we can form a more general selection criterion by using logical operators  $\wedge$  (AND),  $\vee$  (OR), and  $\neg$  (NOT). To be specific, we define a **selection term** to be a single comparison of two values which

may be either

- (1) attributes  $A_i$  and/or  $A_j$  evaluated with respect to a tuple  $t$ , or
- (2) constant values.

Note item (2) can be any expression that evaluates to a constant. For example, see the following three selection terms,

$$\begin{aligned} A_i(t) &= \text{"Smith"} \\ A_k(t) &> 20 \\ A_i(t) &\neq A_j(t) \end{aligned}$$

A **clause** is a conjunction of terms. A **query** is a disjunction of clauses. Any query is called a **valid formula**. For example,  $(A_i(t) = \text{"Smith"} \wedge A_k(t) > 20) \vee (A_i(t) \neq A_j(t))$  is a valid formula. The formal definition of a valid formula will be discussed later. Observe that our definition of a formula in terms of "queries" as defined above is slightly restricted. For example, one could have conjunction of disjunctions and quantifiers, e.g.,

$$\begin{aligned} A_i(t) = 9 \wedge (\exists u)[A_j(t) \neq A_j(u)] \\ (A_i(t) = 9 \vee A_i(t) = 8) \wedge (A_k(t) \neq \text{"Tom"}) \end{aligned}$$

Nothing in our development precludes using these more general "queries", but since no attempt has been made to analyze them, we use the simpler form. Also note the comparison operators  $<$ ,  $=$ ,  $>$ ,  $\leq$ ,  $\neq$ , or  $\geq$  can be used in each term, but to simplify our discussion, we initially assume the perfect match retrieval (equality) in all selection terms.

By a "valid formula" we mean any expression in the predicate calculus with the following properties :

- (1) has a single **free** variable (i.e., without quantifier), implicitly  $t$ .
- (2) references only attributes of the relation  $r$ .
- (3) is **safe** [Ull82, p.159].

Distinction between a valid formula and a safe relational expression [Ull82, p.159]

is item (2) in which all attributes are from same relation.

The result of a selection using a valid formula  $\rho$  on a relation  $r$  with scheme  $R$  denoted by  $\sigma_\rho(r)$ , is a set of tuples  $t$  in  $r$  such that  $\rho(t)$  is true for every  $t$ . Of course,  $\sigma_\rho(r) \subseteq r$ . To distinguish different selection formulae, a subscripted notation  $\rho_1$ , or  $\rho_2$  is used. For notational convenience,  $\sigma_{\rho_1}(r)$  and  $\sigma_{\rho_2}(r)$  will be denoted simply by  $\sigma_1(r)$  and  $\sigma_2(r)$  respectively. Note  $\sigma_\rho$  and  $\rho$  are effectively synonymous, the former just denotes its application to a particular relation. A valid formula  $\rho$  can be viewed as a "restriction" of a relation  $r$ , that is  $\sigma_\rho(r) = \{t \mid t \in r, \rho(t) \text{ is true}\}$  is a subset of  $r$ .

Our interest is in the size or cardinality of the restriction of a relation  $r$  by a valid formula  $\rho_k$ . It is denoted by  $|\sigma_k(r)|$ . We know that the application of the  $\neg$  (NOT) operator is to negate the expression, e.g.,  $\neg(A = a \wedge B < b)$  can be expressed as  $(A \neq a \vee B \geq b)$ . Therefore, in our following discussion we use only  $\wedge$  (AND) and  $\vee$  (OR) to combine two or more selection formulae. Before showing the expected size expressions for a selection, we examine the relationship between two selection formulae.

Let  $Z$  denote the attribute set in a selection formula  $\rho$ , and

$$Z_1 = \{A_i \mid A_i \text{ appears in } \rho_1\}$$

$$Z_2 = \{A_j \mid A_j \text{ appears in } \rho_2\}$$

We say that  $Z_2$  is **statistically dependent** on  $Z_1$  if the probability that a tuple  $t$  will satisfy selection formula  $\rho_1$  is not independent of the probability for any tuple satisfying  $\rho_2$ . That is  $\text{prob}(t \text{ satisfying } \rho_2 \mid t \text{ satisfies } \rho_1) \neq \text{prob}(t \text{ satisfying } \rho_2)$ . Then,  $Z_2$  is **statistically independent** of  $Z_1$  if equality holds. Readily, if  $Z_2$  is "functionally" dependent on  $Z_1$ , then  $Z_2$  is statistically dependent on  $Z_1$ .

Assume  $\rho_i$  is a selection formula whose terms are denoted by  $\rho_{ij}$  such that  $\rho_i = (\rho_{i1} \wedge \rho_{i2} \wedge \dots)$ . We use  $Z_i$  to denote the attribute set appearing in  $\rho_i$ , i.e.,

$$Z_i = \{A_k \mid A_k \text{ appears in } \rho_i\}$$

Without loss of generality we let  $\rho_1 = \rho_{11} \wedge \rho_{12} \dots \wedge \rho_{1m}$  and  $Z_1 = Z_{11} \cup Z_{12} \cup \dots \cup Z_{1m}$  such that  $Z_{11}, Z_{12}, \dots, Z_{1m}$  are statistically independent to each other. Similarly, let  $\rho_2 = \rho_{21} \wedge \rho_{22} \dots \wedge \rho_{2n}$  and  $Z_2 = Z_{21} \cup Z_{22} \cup \dots \cup Z_{2n}$  such that  $Z_{21}, Z_{22}, \dots, Z_{2n}$  are statistically independent to each other. For example,  $\rho_1 = (A = 1) \wedge (B = 2)$  and  $\rho_2 = (C = 3) \wedge (D = 4)$ .

Let  $P_i$  and  $P_{ij}$  denote the probabilities of satisfying  $\rho_i$  and  $\rho_{ij}$  respectively, then from [Mey65, p.42] we know

$$P_1 = P_{11} \cdot P_{12} \cdot P_{13} \dots \cdot P_{1m} \text{ and}$$

$$P_2 = P_{21} \cdot P_{22} \cdot P_{23} \dots \cdot P_{2n}.$$

Now, we consider the following two cases:

(1). If  $Z_1 \cap Z_2 = \emptyset$  and all pairs of  $Z_{1i}$  and  $Z_{2j}$  are statistically independent, then

$$P_{\rho_1 \wedge \rho_2} = P_1 \cdot P_2 \quad (5.2.1)$$

(2). If  $Z_1 \cap Z_2 \neq \emptyset$  or some pairs of  $Z_{1i}$  and  $Z_{2j}$  are statistically dependent, then without loss of generality we can assume that  $(Z_{11}, Z_{21}), \dots, (Z_{1k}, Z_{2k})$  are the dependent pairs of attribute sets (including  $Z_{11} = Z_{21}, \dots, Z_{1k} = Z_{2k}$ ), where  $k \leq m$  and  $k \leq n$ . Here, some of the pairs  $(Z_{1i}, Z_{2i}), 1 \leq i \leq k$ , may be empty. The reason for choosing the terms from 1 to  $k$  is for notational convenience only. Then, the probabilities for satisfying these dependent terms are  $P_{11 \wedge 21}, P_{12 \wedge 22}, \dots, P_{1k \wedge 2k}$ .

So that we have three sets of independent probabilities  $(P_{1(k+1)}, \dots, P_{1m})$ ,  $(P_{2(k+1)}, \dots, P_{2n})$ , and  $(P_{11 \wedge 21}, \dots, P_{1k \wedge 2k})$ .

Therefore,

$$\begin{aligned}
P_{\rho_1 \wedge \rho_2} &= (P_{1(k+1)} \cdot \dots \cdot P_{1m}) \cdot (P_{2(k+1)} \cdot \dots \cdot P_{2n}) \cdot (P_{11 \wedge 21} \cdot \dots \cdot P_{1k \wedge 2k}) \\
&= (P_{11} \cdot \dots \cdot P_{1m}) \cdot (P_{21} \cdot \dots \cdot P_{2n}) \cdot \left( \frac{P_{11 \wedge 21}}{P_{11} \cdot P_{21}} \cdot \dots \cdot \frac{P_{1k \wedge 2k}}{P_{1k} \cdot P_{2k}} \right) \\
&= P_1 \cdot P_2 \cdot \left( \frac{P_{11 \wedge 21}}{P_{11} \cdot P_{21}} \cdot \dots \cdot \frac{P_{1k \wedge 2k}}{P_{1k} \cdot P_{2k}} \right) \tag{5.2.2}
\end{aligned}$$

This is the same as if we partition the selection terms into three independent subsets as shown in Figure 5.2.1.

As an example, consider the selection expressions  $\rho_1 = (A \leq 1) \wedge (B > 2)$  and  $\rho_2 = (A = 1) \wedge (C < 4)$ . By the preceding, we can rearrange them into three statistically independent sets:  $\rho_a = (A \leq 1) \wedge (A = 1)$ ,  $\rho_b = (B > 2)$  and  $\rho_c = (C < 4)$ .

To illustrate this process, we assume all three attributes  $A$ ,  $B$  and  $C$  have the same domain  $\{1, 2, 3, 4\}$ . Since  $\rho_{11} = (A \leq 1)$ ,  $\rho_{12} = (B > 2)$ ,  $\rho_{21} = (A = 1)$  and  $\rho_{22} = (C < 4)$ , we know  $P_{11} = \frac{1}{4}$ ,  $P_{12} = \frac{2}{4}$ ,  $P_{21} = \frac{1}{4}$ ,  $P_{22} = \frac{3}{4}$ , and  $P_{11 \wedge 21} = \frac{1}{4}$ . Then  $P_1 = \frac{1}{4} \cdot \frac{2}{4} = \frac{1}{8}$ ,  $P_2 = \frac{1}{4} \cdot \frac{3}{4} = \frac{3}{16}$ , and  $P_a = \frac{1}{4}$ ,  $P_b = \frac{2}{4}$ ,  $P_c = \frac{3}{4}$ . Thus,

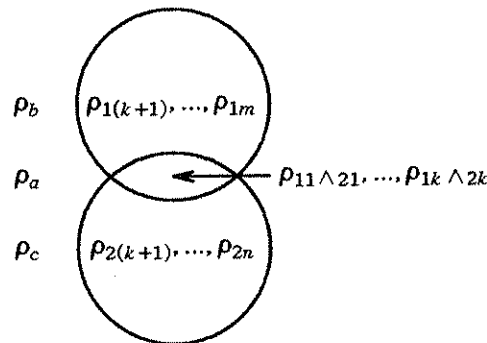


Figure 5.2.1

$$\begin{aligned}
P_{\rho_1 \wedge \rho_2} &= P_1 \cdot P_2 \cdot \left( \frac{P_{11 \wedge 21}}{P_{11} \cdot P_{21}} \right) \\
&= \frac{1}{8} \cdot \frac{3}{16} \cdot \frac{\frac{1}{4}}{\frac{1}{4} \cdot \frac{1}{4}} = \frac{6}{64}
\end{aligned}$$

Equivalently,  $P_{\rho_1 \wedge \rho_2} = P_{a \wedge b \wedge c} = \frac{1}{4} \cdot \frac{2}{4} \cdot \frac{3}{4} = \frac{6}{64}$ . We know the answer is correct, because there are only six out of 64 possible tuples can satisfy  $\rho_1 \wedge \rho_2$ : (1 3 1), (1 3 2), (1 3 3), (1 4 1), (1 4 2), and (1 4 3).

It is very easy to show that the expression (5.2.1) is a special case of the last expression (5.2.2). Since in Case (1) we know  $P_{11 \wedge 21} = P_{11} \cdot P_{21}$ ,  $P_{12 \wedge 22} = P_{12} \cdot P_{22}$ , ...,  $P_{1k \wedge 2k} = P_{1k} \cdot P_{2k}$ , then

$$\left( \frac{P_{11 \wedge 21}}{P_{11} \cdot P_{21}} \cdot \dots \cdot \frac{P_{1k \wedge 2k}}{P_{1k} \cdot P_{2k}} \right) = \left( \frac{P_{11} \cdot P_{21}}{P_{11} \cdot P_{21}} \cdot \dots \cdot \frac{P_{1k} \cdot P_{2k}}{P_{1k} \cdot P_{2k}} \right) = 1$$

In Theorem 5.2.1 we show the expected size expression for compound selections  $\sigma_{\rho_1 \wedge \rho_2}$  and  $\sigma_{\rho_1 \vee \rho_2}$ . Two special cases are also considered for the compound selection  $\sigma_{\rho_1 \wedge \rho_2}$  in the following corollaries. Corollary 5.2.1 assumes that the only (statistically) dependent terms in  $\rho_1$  and  $\rho_2$  are the selection terms common to both formulae. The case when all the selection terms in both  $\rho_1$  and  $\rho_2$  are statistically independent is presented in Corollary 5.2.2.



**Theorem 5.2.1** Let  $\rho_1$  and  $\rho_2$  be valid selection formulae on a relation  $r$  such that

$\rho_1 = \rho_{11} \wedge \rho_{12} \wedge \dots \wedge \rho_{1m}$  and  $\rho_2 = \rho_{21} \wedge \rho_{22} \wedge \dots \wedge \rho_{2n}$ . The probabilities of satisfying  $\rho_i$  and  $\rho_{ij}$  are denoted by  $P_i$  and  $P_{ij}$  respectively. And assume the probability of matching any attribute in a tuple is independent of all other attributes, i.e.,  $P_i = P_{i1} \cdot P_{i2} \cdot \dots \cdot P_{ih}$ . Here  $i = 1$  or  $2$ , and  $j, h = m$  or  $n$ . For notational convenience,  $\sigma_{\rho_k}(r)$  is simply denoted by  $\sigma_k(r)$ . Let  $|\sigma_1(r)|$  and  $|\sigma_2(r)|$  denote the corresponding cardinalities. Then

(1)  $\rho_1 \wedge \rho_2$  and  $\rho_1 \vee \rho_2$  are valid selection formulae.

$$(2) \exp|\sigma_{\rho_1 \wedge \rho_2}(r)| = \frac{\exp|\sigma_1(r)| \cdot \exp|\sigma_2(r)|}{|r|} \cdot \left( \frac{P_{11} \wedge 21}{P_{11} \cdot P_{21}} \cdot \dots \cdot \frac{P_{1k} \wedge 2k}{P_{1k} \cdot P_{2k}} \right)$$

where  $k \leq m$  and  $k \leq n$

$$(3) \exp|\sigma_{\rho_1 \vee \rho_2}(r)| = \exp|\sigma_1(r)| + \exp|\sigma_2(r)| - \exp|\sigma_{\rho_1 \wedge \rho_2}(r)|$$

**Proof :**

(1) If different variable names are used for the single free variable in  $\rho_1$  and  $\rho_2$ , then we can rename the variables to have a single free variable name in  $\rho_1 \wedge \rho_2$  and in  $\rho_1 \vee \rho_2$ . That is, for every  $\rho_1$  and  $\rho_2$ , we have a single free variable in  $\rho_1 \wedge \rho_2$  and in  $\rho_1 \vee \rho_2$ .

Let  $Z(\rho_k) = \{A_i | A_i \text{ appears in } \rho_k\}$  be the attribute set in  $\rho_k$ . Then, for any relation  $r$  with scheme  $R$ , we know that  $Z(\rho_k) \subseteq R$  implying  $Z(\rho_1 \wedge \rho_2) \subseteq R$  and  $Z(\rho_1 \vee \rho_2) \subseteq R$ . Then, from the definition of safety in [Ull82, pp.160] and the fact that  $\rho_1 \wedge \rho_2$  and  $\rho_1 \vee \rho_2$  reference only the attributes in  $R$ , we know that these two formulae are safe.

Therefore,  $\rho_1 \wedge \rho_2$  and  $\rho_1 \vee \rho_2$  are valid selection formulae.

(2) We know that  $P_i = \frac{\exp|\sigma_{\rho_i}(r)|}{|r|}$ , or equivalently,  $\exp|\sigma_{\rho_i}(r)| = P_i \cdot |r|$ , then

$$P_1 = \frac{\exp|\sigma_{\rho_1}(r)|}{|r|} \text{ and } P_2 = \frac{\exp|\sigma_{\rho_2}(r)|}{|r|}.$$

We have shown in the expression (5.2.2) that

$$P_{\rho_1 \wedge \rho_2} = P_1 \cdot P_2 \cdot \left( \frac{P_{11} \wedge 21}{P_{11} \cdot P_{21}} \cdot \dots \cdot \frac{P_{1k} \wedge 2k}{P_{1k} \cdot P_{2k}} \right)$$

Therefore,

$$\begin{aligned} \exp|\sigma_{\rho_1 \wedge \rho_2}(r)| &= P_{\rho_1 \wedge \rho_2} \cdot |r| \\ &= P_1 \cdot P_2 \cdot \left( \frac{P_{11} \wedge 21}{P_{11} \cdot P_{21}} \cdot \dots \cdot \frac{P_{1k} \wedge 2k}{P_{1k} \cdot P_{2k}} \right) \cdot |r| \\ &= \frac{\exp|\sigma_1(r)|}{|r|} \cdot \frac{\exp|\sigma_2(r)|}{|r|} \cdot \left( \frac{P_{11} \wedge 21}{P_{11} \cdot P_{21}} \cdot \dots \cdot \frac{P_{1k} \wedge 2k}{P_{1k} \cdot P_{2k}} \right) \cdot |r| \\ &= \frac{\exp|\sigma_1(r)| \cdot \exp|\sigma_2(r)|}{|r|} \cdot \left( \frac{P_{11} \wedge 21}{P_{11} \cdot P_{21}} \cdot \dots \cdot \frac{P_{1k} \wedge 2k}{P_{1k} \cdot P_{2k}} \right) \end{aligned}$$

(3) Let  $\rho_{22} = \rho_2 \wedge (\neg \rho_1)$  denote the conjunction of selection formula  $\rho_2$  and the negation of  $\rho_1$ . Then

$$\exp|\sigma_{\rho_1 \vee \rho_2}(r)| = \exp|\sigma_1(r)| + \exp|\sigma_{22}(r)| \quad (a)$$

$$\exp|\sigma_2(r)| = \exp|\sigma_{\rho_1 \wedge \rho_2}(r)| + \exp|\sigma_{22}(r)| \quad (b)$$

So that

$$\exp|\sigma_{\rho_1 \vee \rho_2}(r)| - \exp|\sigma_2(r)| = \exp|\sigma_1(r)| - \exp|\sigma_{\rho_1 \wedge \rho_2}(r)| \quad (a)-(b)$$

or equivalently

$$\exp|\sigma_{\rho_1 \vee \rho_2}(r)| = \exp|\sigma_1(r)| + \exp|\sigma_2(r)| - \exp|\sigma_{\rho_1 \wedge \rho_2}(r)| \quad \square$$

**Corollary 5.2.1** Let  $\rho_1$  and  $\rho_2$  be valid selection formulae on a relation  $r$  and  $\rho_1 \cap \rho_2$  denote all selection terms common to both  $\rho_1$  and  $\rho_2$  (assuming the rest of terms are statistically independent). Let  $|\sigma_1(r)|$ ,  $|\sigma_2(r)|$ , and  $|\sigma_{12}(r)|$  denote the corresponding cardinalities. And assume the probability of matching any attribute in a tuple is independent of all other attributes. From Theorem 5.2.1 (2) we have the following result :

$$\exp|\sigma_{\rho_1 \wedge \rho_2}(r)| = \frac{\exp|\sigma_1(r)| \cdot \exp|\sigma_2(r)|}{\exp|\sigma_{12}(r)|}$$

where  $\exp|\sigma_{12}(r)| = |r|$  if  $\rho_1 \cap \rho_2 = \emptyset$

**Proof :**

If  $\rho_1 \cap \rho_2 = \emptyset$ , then the result follows directly from Theorem 5.2.1. We therefore assume that there are  $k \geq 1$  common terms. We know that  $P_{\rho_1 \cap \rho_2} = \frac{\exp|\sigma_{\rho_1 \cap \rho_2}(r)|}{|r|} = \frac{\exp|\sigma_{12}(r)|}{|r|}$ . From the notation of Theorem 5.2.1 and the assumption that all the statistically dependent terms are common to both  $\rho_1$  and  $\rho_2$ , we have

$$\rho_{11} = \rho_{21}, \quad \rho_{12} = \rho_{22}, \quad \dots, \quad \rho_{1k} = \rho_{2k}, \quad 1 \leq k \leq m \quad \text{and} \quad 1 \leq k \leq n.$$

Then

$$P_{11 \wedge 21} = P_{11} = P_{21}, \quad P_{12 \wedge 22} = P_{12} = P_{22}, \quad \dots, \quad P_{1k \wedge 2k} = P_{1k} = P_{2k}.$$

Let  $\rho_1 \cap \rho_2 = \rho_{11} \wedge \rho_{12} \dots \wedge \rho_{1k} = \rho_{21} \wedge \rho_{22} \dots \wedge \rho_{2k}$ , then

$$P_{\rho_1 \cap \rho_2} = P_{11} \cdot P_{12} \cdot \dots \cdot P_{1k} = P_{21} \cdot P_{22} \cdot \dots \cdot P_{2k}.$$

Then, from Theorem 5.2.1 (2)

$$\exp|\sigma_{\rho_1 \wedge \rho_2}(r)| = \frac{\exp|\sigma_1(r)| \cdot \exp|\sigma_2(r)|}{|r|} \cdot \left( \frac{P_{21}}{P_{11} \cdot P_{21}} \cdot \dots \cdot \frac{P_{2k}}{P_{1k} \cdot P_{2k}} \right)$$

$$\begin{aligned}
&= \frac{\exp|\sigma_1(r)| \cdot \exp|\sigma_2(r)|}{|r|} \cdot \left( \frac{1}{P_{11}} \cdot \dots \cdot \frac{1}{P_{1k}} \right) \\
&= \frac{\exp|\sigma_1(r)| \cdot \exp|\sigma_2(r)|}{|r|} \cdot \left( \frac{1}{P_{\rho_1 \cap \rho_2}} \right) \\
&= \frac{\exp|\sigma_1(r)| \cdot \exp|\sigma_2(r)|}{|r|} \cdot \left( \frac{|r|}{\exp|\sigma_{12}(r)|} \right) \\
&= \frac{\exp|\sigma_1(r)| \cdot \exp|\sigma_2(r)|}{\exp|\sigma_{12}(r)|} \quad \square
\end{aligned}$$

**Corollary 5.2.2** If  $\rho_1$  and  $\rho_2$  are statistically independent selection formulae and  $\sigma_{\rho_k}(r)$  is denoted simply by  $\sigma_k(r)$ , then from Corollary 5.2.1 we have the following result :

$$\exp|\sigma_{\rho_1 \wedge \rho_2}(r)| = \frac{\exp|\sigma_1(r)| \cdot \exp|\sigma_2(r)|}{|r|}$$

**Proof :**

It is straight forward to show that the expression is true, since we know that  $\exp|\sigma_{12}(r)| = |r|$  when  $\rho_1$  and  $\rho_2$  do not have common selection term (or any statistically dependent term). By substituting  $|r|$  into Corollary 5.2.1 , we are done.

This is really an instance of Corollary 5.2.1, since without loss of generality we can augment  $\rho_1$  and  $\rho_2$  by a third selection formula  $\rho_3$  with selection terms neither in  $\rho_1$  nor in  $\rho_2$ . Then, from Theorem we know that  $\rho_1 = \rho_1 \wedge \rho_3$  and  $\rho_2 = \rho_2 \wedge \rho_3$  are valid selection formulae.  $\square$

### 5.3. Expected block accesses of selection

As mentioned before, the cost of processing a selection operation is normally measured by the expected number of disk block accesses. In the previous sections we have shown expressions for the expected size of selection,  $exp|\sigma_k(r)|$ . Now, we need to translate the expected number of tuples selected in a relation  $r$  into the number of blocks they occupy. Since no specific storage system has been considered, we assume that the tuples are randomly selected from an uniformly distributed relation.

Let  $n$  denote the expected number of selected tuples and  $B = \frac{|r|}{BlockingFactor}$  denote the total number of blocks in a relation  $r$ . We are concerned with the way that the  $n$  (distinct) tuples are distributed among  $B$  blocks, since the distribution of tuples determines the number of blocks they are stored. One approach presented by Cardenas [Car75] has the following expression for the expected block accesses

$$exp(\text{Block accesses}) = B [1 - (1 - \frac{1}{B})^n]$$

where  $(1 - \frac{1}{B})$  represents the probability that a particular block does not contain a particular tuple. The probability of a block not being occupied by any of the  $n$  selected tuples is  $(1 - \frac{1}{B})^n$  assuming the selections are made independently. Therefore,  $1 - (1 - \frac{1}{B})^n$  denotes the probability that at least one of the  $n$  tuples is in a particular block. Then the expression shows the probability for all  $B$  blocks.

However, Cardenas' assumption that the  $n$  tuples are selected independently means they may not be distinct. In [Yao77] Yao develops a formula for selecting  $n$  distinct tuples at one time and shows that the difference between two approaches is significant when the blocking factor is small (less than 10). Yao's formula

includes the relation size  $|r|$  as follows

$$\exp(\text{Block accesses}) = B \left[ 1 - \prod_{i=1}^n \frac{|r| \cdot (1 - \frac{1}{B}) - i + 1}{|r| - i + 1} \right]$$

In most real applications, blocking factors are much larger than 10 and generally  $|r| \gg n$ ,  $|r| \gg B$ . Therefore, the discrepancy is usually negligible. In an experiment, we considered two relations with different blocking factors. In Table 5.3.1 we show two sets of results according to these two parameters. The first four rows comes from a relation of size 50 tuples with a blocking factor 5. The relation size and blocking factor for the second set are 300 and 10 respectively. Note that the actual numbers of block accesses in the forth column are average counts from several independent runs. As expected Yao's estimate is somewhat better than Cardenas'. However, the maximal difference is only  $(9.20\% - 6.13\%) = 3.07\%$ .

The assumptions that the selection is random and the distribution of the selected tuples are uniformly distributed over  $B$  blocks is necessary to provide simple and low cost solutions to the problem. If more accurate estimations on

Relation size	Blocking factor	Selection size	Actual # of blocks	Cardenas' estimate	Error %	Yao's estimate	Error %
50	5	1	1.00	1.000	0.00	1.000	0.00
50	5	2	2.00	1.900	-5.00	1.918	-4.08
50	5	5	4.51	4.095	-9.20	4.234	-6.13
50	5	50	10.00	9.948	-0.52	10.000	0.00
300	10	1	1.00	1.000	0.00	1.000	0.00
300	10	2	2.00	1.967	-1.67	1.970	-1.51
300	10	20	14.98	14.772	-1.39	15.116	0.91
300	10	300	30.00	29.999	0.00	30.000	0.00

Table 5.3.1 Block accesses

certain distributions are required, then other information and (statistical) parameters are needed to develop complex formulae. For example, in [Che82] Cheung considers the case where the selected tuples may have duplications; Chan and Niamir in [Cha82] derive a general formula that allows the tuples to cross block boundaries and the number of blocks may not be of integral; and in [Zah83] Zahorjan presents an efficient computational technique for estimating block numbers when tuple access probabilities are non-uniform. For both operational and expository simplicity we have chosen to use only Cardenas' formula.

## CHAPTER 6

### Access Cost of Queries

#### 6.1. Issues in cost analysis

We know that the cost of performing a join or any other relational operator is determined primarily by the size of involved relations, that is the amount of data to be processed. If one assumes the relation size  $|r_i|$  and other associated variables such as the attribute domain size  $|A_i|$  and the distribution of values in  $D_{A_i}$  to be fixed, then the processing cost depends on the following three factors:

- (1) the physical organization of relations in secondary storage,
- (2) the access methods used in the implementation, and
- (3) the design of the algorithm implementing the operator.

In Chapter 2 and Chapter 5 we investigated the size of a derived relation which is independent of any of these factors, but it can be used to optimize the processing cost, particularly item (3). We will briefly describe each of these three aspects in turn.

First, we examine three major issues concerning the physical storage of relations in secondary storage:

1. Blocking factor ( $BF$ ), which denotes the number of tuples (records) in each disk block. Normally it is determined by the hardware block size and by the blocking method, e.g., fixed blocking, variable spanned blocking [Wie77, p.45].
2. Clustering. If  $n$  tuples must be retrieved to perform an operation, the processing cost is minimized when they are stored in the same block ( $n \leq BF$ ). The actual effect of clustering depends on the distribution of these  $n$  tuples in the



relation. In general, for  $n \ll |r|$

$$\text{Cost}(\text{perfectly clustered } n) \equiv \frac{1}{BF} \text{Cost}(\text{unclustered } n).$$

3. Physical assignment to tracks. The seek time and rotational delay are two big factors of the access time in a secondary device. If a relation occupies consecutive cylinders, then seek times can be effectively reduced. Choosing appropriate block sizes can make optimal use of the track capacity and reduce the access time. If the retrieval pattern can be predicted, a proper mapping between the tuples and their physical addresses in the tracks can significantly improve the I/O performance. However, if a tuple is pinned down [Ull82, p.39] to a fixed location, it is hard to move around in a relation or between different devices.

Since the physical organization of a relation can be device dependent, in our models we use general assumptions such as fixed blocking factors, clustering on the key attribute (if any), and using the maximum capacity of disk tracks.

Second, we look three typical access methods: sequential access, key access, and partial-match retrieval.

1. Sequential access method. When most of the tuples in a relation need to be processed, sequential access is the most effective retrieval method. However, to locate a specified tuple by a sequential search requires scanning at least half the relation on the average. Normally, a sequential access method is more productive in a batch processing when the relation has been put in the sequence of the desired attributes.

2. Key access, e.g., B-tree [Bay72]. A key access method is best suited for the random retrieval of a relatively small number of tuples from a relation. Various schemes can be used to implement the indices (pointers) on the designated attributes to speed up the access time. In general, hashed methods [Kno75] provide a faster access while indexed accesses allow retrieval in sorted order.

In common 3NF or BCNF [Cod74, Mai83] data base decompositions, most joins will be with respect to relation keys, so key access is practical for join attributes. But most queries also involve selections, where the selection is on non-key attribute (field). To optimize these queries, partial-match algorithms are desirable.

3. Partial-match retrieval, e.g., IDAM [Pfa80]. Since many queries involve more than one attribute, partial-match retrieval is frequently used in database processing. Although multiple secondary indices can be employed to find the specified tuples, a simple and more efficient access method called IDAM provides "tunable" performance for very large databases. Moreover, in IDAM retrieval the amount of work decreases when more attributes are specified in partial-match queries. Thus, we will begin with an IDAM system in the development of access cost formulae. A brief description of an IDAM system will be given later.

Last, we consider the goal of designing efficient algorithms to implement relational operators. There are several ways in which the efficiency of an algorithm can be measured. We divide the cost criteria into two categories: **storage overhead** and **computational (or processing) cost**. In many procedures, processing cost is optimized at the price of storage overhead, or vice versa. A practical algorithm should attempt to minimize both (if possible).

By storage overhead we mean the additional storage needed to implement the retrieval mechanism, such as the storage for pointers, or index files. We measure processing cost in terms of disk block accesses. By their very nature one expects data bases to be relatively large, and therefore represented in secondary storage (disk). If a data base is small enough that its relations can be largely stored in main memory, then the whole issue of retrieval cost becomes moot. Because secondary storage access time dominates the entire processing cost of retrieval, it is reasonable to measure processing cost in terms of the number of disk accesses. This measure of the processing cost of a relational operator is independent of any

particular storage structure and access method.

General query processing normally employs the selection operation to retrieve some specific tuples in a relation. Sometimes, we may use "selection", "retrieval", and "query" synonymously. In fact, the join operator itself must also employ a selection process. As we have discussed in the previous chapter, the specification of those tuples to be selected (or retrieved) is normally accomplished by specifying the desired values for one or more attributes (fields) in those tuples which will satisfy the query. Since many queries require multi-attribute retrieval, we begin with a discussion of partial-match retrieval in the next section.

## 6.2. Partial-match retrieval

When more than one selection attribute may be specified in the query, then we have multi-attribute access which is commonly known as **partial-match retrieval** [Riv76]. To satisfy such queries, all of the qualified records (tuples) must have the specified values as their selection attributes. When the acceptable values for a given attribute are more than a single value, then we have a **disjunctive retrieval**. It is called a **range search** if an interval of values is desired. However, if at most a single value is specified for any attribute, then it is called **perfect match** access. Note that the join operation can be viewed as perfect match retrieval on the join attribute(s), while selection operations are normally partial-match retrievals.

Several accessing methods and file structures have been studied and implemented for partial-match retrieval systems. [Car75] describes inverted files and their performance. In [Sev77], four file structures are discussed: multilist, cellular list, record inverted list, and cellular inverted list. Bit-string representations are reviewed in [Val76] and bit-slice search algorithms in [Rob79]. In [Ull82] it shows a partitioned hash method by using the full set of attributes as the key, in

which no indices (either primary or secondary) are used. The Indexed Descriptor Access Method is described in [Pfa80] and is used in [Ram83] along with a hashing technique. An intensive investigation of IDAM files can be found in [Fre82].

Since the join operation is basically a selection process on two relations, it is necessary to examine the behavior of selection in more detail. A cost expression for selection will be described and used later in the development of access cost for joins. First, we begin by a short introduction of the IDAM system.

### 6.3. Indexed Descriptor Access Method

The basic technology of the Indexed Descriptor Access Method was developed by Edgar Cagley in 1971 [Cag71]. A complete description and cost analysis can be found in [Pfa79,Pfa80,Pfa82,Fre82]. We will give a brief description here.

In IDAM the data records are represented in storage in a way that allows random access to any record. A collection of indexed descriptor files are used to provide the access paths. A descriptor is a string of  $w$  bits that encodes the values of the attributes of a record ( $D_R$ , record descriptor), or block of records ( $D_\beta$ , block descriptor). When descriptors are created by superimposed coding [Knu73, Rob79], each attribute will set one, or more, bits among all the  $w$  possible bits. By dividing the descriptor into  $k$  subfields, each of width  $w_i$  ( $1 \leq i \leq k, w = \sum_{i=1}^k w_i$ ), any value of attribute  $i$  will set precisely one of the  $w_i$  bits in field  $i$ . This approach is called disjoint coding in IDAM.

A block descriptor  $D_\beta$  is formed by simply OR-ing together all the record descriptors  $D_R$  within the block. The collection of all block descriptors, together with pointers to the corresponding data blocks form the first level of descriptor file. To avoid exhaustively searching this entire file sequentially, their records (block descriptors and pointers) are grouped into blocks. A new block descriptor is

then created the same way as the old one is formed. These new descriptors are collected into a second level of index file. The process continues until the highest level of index file is small enough to be stored in main memory. This top level file is the only file needs to be searched exhaustively in the retrieval process.

To retrieve a set of records for satisfying a query  $Q$ , we

- (1) Form the query descriptor  $D_Q$  the same way as  $D_R$  is created.
- (2) Compare  $D_Q$  exhaustively with all the descriptors in the top level file.
- (3) If they match, i.e.,  $D_Q \cap D_\beta$  is non-null in each field, then get the corresponding block of the next level file and
  - (a) If not level 0, every descriptor in the block is compared with  $D_Q$ .
  - Repeat (3) for each comparison if there is a match.
  - (b) Otherwise, each data record in that block is compared with  $Q$ .

All the matched records are added to the output file.

#### 6.4. Access cost of selection on IDAM files

In [Pfa80] the total number of blocks accessed in responding to a query  $Q$  is given by the following formula

$$\exp(\text{block accesses} | Q) = \sum_{i=0}^{h-1} [n_{i+1} \prod_{l \in Q} \frac{\overline{x}_{l,i+1}}{w_l}] \quad (6.4.1)$$

where  $h$  is the highest level number,  $n_{i+1}$  is the number of records in level  $(i+1)$  file,  $\overline{x}_{l,i+1}$  is the expected number of 1-bits in the  $l^{th}$  field of a descriptor at level  $(i+1)$ , and  $w_l$  is the number of bits in field  $l$ . We know that the number of records in file  $(i+1)$  is the same as the number of blocks in file  $i$ , and denoted by  $B_i$ . In this equation we assume the highest level file  $h$  is small enough to be stored in main memory. This is a reasonable and practical assumption in real applications unless we are dealing with a very huge relation. In [Pfa82, p.6] Pfaltz states that it has never used more than three index files to access any file of less

than a million records while the top level index file is in main memory.

We notice that the above expression also holds for a single attribute retrieval. It implies that the expression (6.4.1) can be used as the access cost equation for the selection with one or more attributes specified as well. Let  $\sigma_\rho(r)$  be any selection from a relation  $r$  using a selection criteria  $\rho$  and let  $\alpha(\sigma_\rho(r))$  denote the expected access cost of this selection, then from (6.4.1)

$$\alpha(\sigma_\rho(r)) = \sum_{i=0}^{h-1} \alpha_i(\sigma_\rho(r)) = \sum_{i=0}^{h-1} [B_i \prod_{l \in \rho(r)} \frac{\overline{w}_{l,i+1}}{w_l}]$$

where  $\alpha_i(\sigma_\rho(r))$  denotes the access cost of  $\sigma_\rho(r)$  in file level  $i$ . When  $\rho$  is an empty set, i.e.,  $\overline{w}_{l,i+1} = w_l$ , then  $\alpha(\sigma_\rho(r)) = \sum_{i=0}^{h-1} B_i$ . In order to develop a cost formula for performing a join operation, the following theorem is given to show the access cost of selection when the logical operator  $\wedge$  (AND) is used. Corollary 6.4.1 describes a general formula for the case where two selection formulae do not have any common term. Note that we assume the tuples satisfying selection formulae are distributed uniformly over the entire relation.

**Theorem 6.4.1** Let  $\rho_1$  and  $\rho_2$  be valid selection formulae on an IDAM file  $r$  and let  $\rho_{12}$  denote those selection terms common to both  $\rho_1$  and  $\rho_2$  (assuming the rest of terms are statistically independent). Short notation  $\sigma_k(r)$  is used to denote  $\sigma_{\rho_k}(r)$ . Also let  $\alpha(\sigma_1(r))$ ,  $\alpha(\sigma_2(r))$  denote the corresponding access costs (the number of blocks accessed). Then

$$\alpha(\sigma_{\rho_1 \wedge \rho_2}(r)) = \sum_{i=0}^{h-1} \frac{\alpha_i(\sigma_1(r)) \cdot \alpha_i(\sigma_2(r))}{\alpha_i(\sigma_{12}(r))}$$

and  $\alpha_i(\sigma_{12}(r)) = B_i$  if  $\sigma_{12} = \emptyset$

where  $\alpha_i$  is the access cost of level  $i$  index file ( $i > 0$ )

$\alpha_0$  is the access cost of data file ( $i = 0$ )

$h$  is the highest level of index file (in main memory)

$B_i$  is the number of blocks in level  $i$ .

**Proof :**

From [Pfa80,pp.526] we know that

$$\alpha(\sigma_k(r)) = \sum_{i=0}^{h-1} \alpha_i(\sigma_k(r)) = \sum_{i=0}^{h-1} [B_i \prod_{l \in \rho_k(r)} \frac{\overline{s}_{l,i+1}}{w_l}]$$

where  $\overline{s}_{l,i+1}$  is the expected number of bits set in field  $l$  of file  $i+1$  and  $w_l$  is the bit width of field  $l$ . Let  $p(i+1,l) = \frac{\overline{s}_{l,i+1}}{w_l}$ , then

$$\prod_{l \in \rho_k(r)} \frac{\overline{s}_{l,i+1}}{w_l} = \prod_{l \in \rho_k(r)} p(i+1,l) = \frac{\alpha_i(\sigma_k(r))}{B_i}$$

Then from the definition of selection formula and the result of Section 5.2 we have the following equation when  $\rho_1$  and  $\rho_2$  do not have common term, i.e.,  $\alpha_i(\sigma_{12}(r)) = B_i$  or equivalently  $\prod_{l \in \rho_{12}(r)} p(i+1,l) = 1$  :

$$\prod_{l \in (\rho_1 \wedge \rho_2)(r)} p(i+1,l) = [\prod_{l \in \rho_1(r)} p(i+1,l)] \cdot [\prod_{l \in \rho_2(r)} p(i+1,l)]$$

And in general, we add the denominator to include the case when  $\rho_{12}$  does exist. Thus

$$\begin{aligned} \prod_{l \in (\rho_1 \wedge \rho_2)(r)} p(i+1,l) &= \frac{[\prod_{l \in \rho_1(r)} p(i+1,l)] \cdot [\prod_{l \in \rho_2(r)} p(i+1,l)]}{[\prod_{l \in \rho_{12}(r)} p(i+1,l)]} \\ &= \frac{\frac{\alpha_i(\sigma_1(r))}{B_i} \cdot \frac{\alpha_i(\sigma_2(r))}{B_i}}{\frac{\alpha_i(\sigma_{12}(r))}{B_i}} \end{aligned}$$

$$= \frac{\alpha_i(\sigma_1(r)) \cdot \alpha_i(\sigma_2(r))}{B_i \cdot \alpha_i(\sigma_{12}(r))}$$

We know  $\alpha_i(\sigma_{12}(r))$  is always greater than 0, thus

$$\begin{aligned} \alpha(\sigma_{\rho_1 \wedge \rho_2}(r)) &= \sum_{i=0}^{h-1} \alpha_i(\sigma_{\rho_1 \wedge \rho_2}(r)) \\ &= \sum_{i=0}^{h-1} [B_i \prod_{l \in (\rho_1 \wedge \rho_2)(r)} \frac{\overline{s}_{l,i+1}}{w_l}] \\ &= \sum_{i=0}^{h-1} [B_i \prod_{l \in (\rho_1 \wedge \rho_2)(r)} p(i+1, l)] \\ &= \sum_{i=0}^{h-1} B_i \frac{\alpha_i(\sigma_1(r)) \cdot \alpha_i(\sigma_2(r))}{B_i \cdot \alpha_i(\sigma_{12}(r))} \\ &= \sum_{i=0}^{h-1} \frac{\alpha_i(\sigma_1(r)) \cdot \alpha_i(\sigma_2(r))}{\alpha_i(\sigma_{12}(r))} \quad \square \end{aligned}$$

**Corollary 6.4.1** If  $\rho_1$  and  $\rho_2$  have no common selection terms in relation  $r$  and  $\sigma_{\rho_k}(r)$  is denoted simply by  $\sigma_k(r)$ , then from Theorem 6.4.1 we have the following result :

$$\alpha(\sigma_{\rho_1 \wedge \rho_2}(r)) = \sum_{i=0}^{h-1} \frac{\alpha_i(\sigma_1(r)) \cdot \alpha_i(\sigma_2(r))}{B_i}$$

where  $B_i$  is the total number of blocks in level  $i$  file.

**Proof :**

It is straight forward to show that the expression is true, since we know that  $\alpha_i(\sigma_{12}(r)) = B_i$  when  $\rho_1$  and  $\rho_2$  do not have common selection term. By substituting  $B_i$  into Theorem 6.4.1, we are done.



This is really an instance of Theorem 6.4.1, since without loss of generality we can augment  $\rho_1$  and  $\rho_2$  by a third selection formula  $\rho_3$  with selection terms neither in  $\rho_1$  nor in  $\rho_2$ . Then, from Theorem we know that  $\rho_1 = \rho_1 \wedge \rho_3$  and  $\rho_2 = \rho_2 \wedge \rho_3$  are valid selection formulae.  $\square$

As shown in Table 6.4.1, our experimental result of estimating the expected access cost of  $\sigma_{\rho_1 \wedge \rho_2}$  is reasonably accurate. According to the number of attributes specified in  $\rho_1 \wedge \rho_2$ , four sets of outcome are displayed. For example, in set 1 we have an empty intersection of selection formula  $\rho_1 \wedge \rho_2 = \emptyset$  and in set 2 one attribute (or one selection term) is specified. In each set, column  $\alpha$  shows the actual access cost and  $\alpha^*$  represents the expected values from our formulae. Each of the values displayed in the table is an average figure computed from several runs. Error percentages are included only for the last two sets since we have perfect estimates on the first two sets as expected.

---

$\alpha(\sigma_{\rho_1 \wedge \rho_2})$										
Number of attributes specified in $\rho_1 \wedge \rho_2$										
File Size	0		1		2			3		
	$\alpha$	$\alpha^*$	$\alpha$	$\alpha^*$	$\alpha$	$\alpha^*$	err%	$\alpha$	$\alpha^*$	err%
100	25.00	25.00	8.16	8.16	5.66	6.19	9.36	3.33	2.68	-19.52
200	50.00	50.00	17.16	17.16	13.50	12.99	-3.78	6.16	5.33	-13.47
300	75.00	75.00	26.33	26.33	21.50	20.98	-2.42	8.33	7.50	-9.96
400	100.00	100.00	39.20	39.20	26.00	25.32	-2.62	8.80	9.44	7.27

---

Table 6.4.1 The access cost of  $\sigma_{\rho_1 \wedge \rho_2}$

It should be noticed that as more attributes specified in  $\rho_1 \wedge \rho_2$ , we get a higher error percentage. We also note that an imperfect uniform distribution of attribute values has greater impact on the small files than the large ones. The estimation error will be minimal with a perfectly uniform distribution.

### 6.5. Access cost of a query involving join and selection

We propose a simple and generalized expression for the cost of performing join and selection operations on any two relations  $r$  and  $s$  as follows :

$$\begin{aligned} \text{Cost } (\sigma_1(r) \bowtie_A \sigma_2(s)) = & \\ & (1) \text{ Cost to find all t-tuples in } \sigma_1(r) + \\ & (2) \text{ Cost to access all t-tuples in } \sigma_1(r) + \\ & (3) |\sigma_1(r)| \cdot ( \\ & (4) \quad \text{Cost to find all u-tuples in } \sigma_2(s) \\ & \quad \text{such that } A(t) = A(u) + \\ & (5) \quad \text{Cost to access these u-tuples} + \\ & (6) \quad \text{Cost to store } r \bowtie s ) \end{aligned} \tag{6.5.1}$$

Note that (1) and (4) are index dependent terms while (2) and (5) are clustering dependent terms. Term (3) indicates that we perform operations of terms (4)-(6) for every tuple  $t$  in  $\sigma_1(r)$ . (We assume  $|\sigma_1(r)| \leq |\sigma_2(r)|$ .) Here we simply present a straight forward algorithm for join operation. Later in this section an improved version will be described. As for term (6), we have already presented the expression for the expected size of the joined relation in Chapter 2. It is obvious that the cost of storing  $r \bowtie s$  depends on the blocking factor of the output. We treat this as a known cost and no further discussion on this item will be followed.

It can be seen that no specific file organization or access path is used in the expression (6.5.1). It should also be apparent that this is not an optimal algorithm

for performing a join operation. Actually, we do not intend to present any minimal access cost formula for the query processing in this study. Our goal is to develop a general expression for the access cost of a join. In doing so, we include selection operations in the formula since they are normally specified in a general query. If a selection formula is not specified, let  $\sigma_1(r) = r$  (or  $\sigma_2(s) = s$ ).

For a "worst case" benchmark, we use the simplest file organization and access method consisting of sequentially scanning two randomly organized sequential files (relations)  $r$  and  $s$ . Since there is no index used in the whole process, terms (1) and (4) of expression (6.5.1) become zeroes. Let  $BF$  denote the blocking factor for both  $r$  and  $s$ , then

$$\text{Access cost of } r \bowtie_A s = \frac{|r|}{BF} + |r| \cdot \frac{|s|}{BF} \quad (6.5.2)$$

where  $\frac{|r|}{BF}$  is the access cost of  $r$  (term (2)) and  $\frac{|s|}{BF}$  is the access cost of  $\sigma_{A=a}(s)$  which represents term (5). By  $\sigma_{A=a}(s)$  we mean the perfect match access on the join attribute  $A$  of relation  $s$  according to the value retrieved from  $r$ ;  $|r|$  represents term (3). If the selection formulae  $\rho_1$  and  $\rho_2$  are specified, the cost expression would be :

$$\alpha(\sigma_1(r) \bowtie_A \sigma_2(s)) = \frac{|r|}{BF} + |\sigma_1(r)| \cdot \frac{|s|}{BF}$$

Here,  $\alpha(\sigma_1(r))$  and  $\alpha(\sigma_{\rho_2 \wedge A=a}(s))$  have the same values as in the equation (6.5.2) since there is no index used in the algorithm. However, if we have indexes for the access paths as in IDAM files, then the equation would become more complex. For an IDAM system we use the access cost incurred in the levels from 1 to  $h-1$ , i.e., index files, to represent the terms (1) and (4) in the expression (6.5.1). The access cost in level 0, i.e., data file, corresponds to terms (2) and (5). A complete cost expression is shown in the next corollary.

**Corollary 6.5.1** If  $\rho_1$  and  $\rho_2$  are selection formulae on IDAM files  $r$  and  $s$  respectively and short notation  $\sigma_k(r)$  is used to denote  $\sigma_{\rho_k}(r)$ , then the access cost for the join of  $\sigma_1(r)$  and  $\sigma_2(s)$  on the common attribute  $A$ ,  $\alpha(\sigma_1(r) \bowtie_A \sigma_2(s))$  is

$$\sum_{i=0}^{h-1} [\alpha_i(\sigma_1(r)) + |\sigma_1(r)| \cdot \frac{\alpha_i(\sigma_2(s)) \cdot \alpha_i(\sigma_{A=a}(s))}{B_i}]$$

where  $A = a$  is a selection term with join attribute value  $a$  from  $\sigma_1(r)$ .

**Proof :**

We know the access cost for the join on IDAM files is

$$\alpha(\sigma_1(r) \bowtie_A \sigma_2(s)) = \alpha(\sigma_1(r)) + |\sigma_1(r)| \cdot \alpha(\sigma_{\rho_2 \wedge A=a}(s))$$

As we have stated in the previous corollary, it is assumed without loss of generality that  $\sigma_2(s)$  does not include the attribute  $A$  in the selection formula. Therefore, we have the following equation

$$\alpha(\sigma_{\rho_2 \wedge A=a}(s)) = \sum_{i=0}^{h-1} \frac{\alpha_i(\sigma_2(s)) \cdot \alpha_i(\sigma_{A=a}(s))}{B_i}$$

So that

$$\begin{aligned} \alpha(\sigma_1(r) \bowtie_A \sigma_2(s)) &= [\sum_{i=0}^{h-1} \alpha_i(\sigma_1(r))] + |\sigma_1(r)| \cdot \sum_{i=0}^{h-1} \frac{\alpha_i(\sigma_2(s)) \cdot \alpha_i(\sigma_{A=a}(s))}{B_i} \\ &= \sum_{i=0}^{h-1} [\alpha_i(\sigma_1(r)) + |\sigma_1(r)| \cdot \frac{\alpha_i(\sigma_2(s)) \cdot \alpha_i(\sigma_{A=a}(s))}{B_i}] \quad \square \end{aligned}$$

Now, for any arbitrary data file  $r$  with uniform distribution on the attribute values, we define the access cost of a selection as follows

$$\alpha_0(\sigma_1(r)) = B_0(r) \cdot G(\sigma_1(r))$$

Here,  $B_0(r)$  denotes the total number of data blocks in  $r$  and  $G(\sigma_1(r)) = \frac{|\sigma_1(r)|}{|r|}$  denotes the ratio between the numbers of blocks selected and the total number of blocks in  $r$ . It has been shown in the last theorem that with an IDAM organization

$$\alpha_0(\sigma_{\rho_1 \wedge \rho_2}(r)) = \frac{\alpha_0(\sigma_1(r)) \cdot \alpha_0(\sigma_2(r))}{\alpha_0(\sigma_{12}(r))}$$

It is also true that  $\alpha_0(\sigma_{12}(r)) = B_0(r)$  if  $\rho_{12} = \emptyset$ . From our observation, we propose the following access cost formula of a join over attribute  $A$  (with optional selection formula specified) for two randomly created data files (relations)  $r$  and  $s$  with uniform distribution of attribute values :

$$\alpha_0(\sigma_1(r) \bowtie_A \sigma_2(s)) = \alpha_0(\sigma_1(r)) + |\sigma_1(r)| \cdot \frac{\alpha_0(\sigma_2(s)) \cdot \alpha_0(\sigma_{A=a}(s))}{B_0(s)} \quad (6.5.3)$$

This general expression is independent of any specific file organization. Since it deals only with records in the data file, it is also independent of the access method. Moreover, we have already shown the expressions for computing these three terms and their empirical results:  $\alpha_0(\sigma_1(r))$  in Section 5.3,  $|\sigma_1(r)|$  in Section 5.1, and  $\frac{\alpha_0(\sigma_2(s)) \cdot \alpha_0(\sigma_{A=a}(s))}{B_0(s)}$  in Section 6.4. All the terms except  $B_0(s)$ , the number of blocks in  $s$ , can be tuned to minimize the total access cost by using certain optimal access paths and/or efficient file structures.

For a simple example, we could modify term (3) in the expression (6.5.1) by using a more efficient join algorithm. Instead of performing the operations in terms (4) and (5) for each tuple  $t$  in  $\sigma_1(r)$ , we could examine the current data block accessed from relation  $r$  and perform these operations once for each of the join

attribute values in the block. To simplify our example, we assume that the blocking factor of relation  $r$  is ten tuples per block and five of them are selected in  $\sigma_1(r)$ . The join attribute values for these five tuples are  $\{a_1, a_2, a_3, a_1, a_2\}$ . In the original algorithm, we have to perform terms (4) and (5) five times while the new algorithm only requires three times, i.e., one time for each of the attribute values  $\{a_1, a_2, a_3\}$ .

Therefore, we can replace  $|\sigma_1(r)|$  in term (3) of expression (6.5.1) and in the right hand side of expression (6.5.3) by

$$\mathbf{B}_0(\sigma_1(r)) \cdot \bar{d} \quad (6.5.4)$$

where  $\mathbf{B}_0(\sigma_1(r))$  is the number of data blocks in  $\sigma_1(r)$  and  $\bar{d}$  is the expected number of tuples with distinct join attribute values in each block of  $\sigma_1(r)$ . Note that the result of (6.5.4) is an expected value and it is very easy to show that  $\mathbf{B}_0(\sigma_1(r)) \cdot \bar{d} \leq |\sigma_1(r)|$ .

First, let  $\pi$  denote the expected number of tuples per block in  $\sigma_1(r)$ , i.e.,

$$\pi = \frac{|\sigma_1(r)|}{\mathbf{B}_0(\sigma_1(r))}$$

Then, from [Che82, p.486] we know the expected value  $\bar{d}$  is the following

$$\bar{d} = \frac{|A| \cdot \pi}{|A| + \pi - 1}$$

Since  $\pi \geq 1$ , we know  $|A| \leq |A| + \pi - 1$ , i.e.,  $\frac{|A|}{|A| + \pi - 1} \leq 1$ .

Therefore,  $\frac{|A| \cdot \pi}{|A| + \pi - 1} \leq \pi$ , or equivalently  $\bar{d} \leq \pi = \frac{|\sigma_1(r)|}{\mathbf{B}_0(\sigma_1(r))}$ .

That is

$$\mathbf{B}_0(\sigma_1(r)) \cdot \bar{d} \leq \mathbf{B}_0(\sigma_1(r)) \cdot \pi = |\sigma_1(r)| \quad \square$$

Although we have proved that  $\mathbf{B}_0(\sigma_1(r)) \cdot \bar{d} \leq |\sigma_1(r)|$ , the actual improvement depends on the distribution of the distinct join attribute values in the relation  $r$ .

Finally, to show that various techniques of computing the cost of join operation can be easily implemented on our general scheme of expression (6.5.1), we made a bit-wise join experiment on IDAM files. The main idea was to perform the join operation once for each bit of the join attribute field of the descriptor, instead of checking all the bits at one time. If the field-width of the descriptor on the join attribute is  $w$ , then the whole join operation will be performed  $w$  times and only one join bit needs to be matched each time.

The experimental result is shown in Table 6.5.1. Obviously, the access cost of a bit-wise join is higher than the cost of a normal join in each of the four sets of independent runs. However, we observe that a better performance may be achieved if the files can be clustered in the bit order sequence of the join attribute field of the descriptor and block descriptors have a lower 1-bit density on the join attribute.

---

	(1)	(2)	(3)	(4)
$ r ,  s $	50	50	50	50
$ A $	5	5	10	10
$BF_{data}$	4	4	5	5
$BF_{desc}$	3	3	3	3
$ \rho_r ,  \rho_s $	0	1	0	2
Normal $\alpha$	140.0	13.0	277.3	11.5
Bitwise $\alpha$	200.0	33.5	329.0	27.0

---

Table 6.5.1 The access cost of a bitwise join

## CHAPTER 7

### Conclusions

#### 7.1. Major contributions

For relational data bases, the selection and join operations play a heavy role in query processing. We have presented practical approaches to predict and evaluate the performance of these two operations in terms of the secondary storage accesses, in a manner that enables users to design and analyze selection and join algorithms effectively. One of the major contributions in this research has been the development of mathematical models for computing the cost of performing join and selection operations and the expected size of the resultant relations. In particular, our formulae are sufficiently general to provide analytic results and detailed enough to make accurate estimation in specific systems under realistic assumptions.

We have shown efficient methods to evaluate the behavior of the natural join operation as well as optional selections. Our empirical studies demonstrated the high correlation between the attribute value distribution and the outcome of join and selection operations. Parametric and non-parametric statistical methods were employed to derive mathematical formulae in our generalized models. Error analysis was performed for the estimation schemes to find the estimation errors and their maximum values. Further, we have also developed various algorithms to minimize the maximum estimation errors. Moreover, several methods of monitoring statistical parameters in a dynamic environment were proposed that permit run-time estimates with any required accuracy but with increasing processing cost.

In conclusion, it has been our goal to provide effective evaluation models for improving the development and analysis of query processing algorithms.



## 7.2. Summary

The background of this study was introduced in Chapter 1 by reviewing the relation model with our own terminology and notation and illustrating the role of join and selection operations in general queries. A join operation is more complex and computationally expensive than a selection because it involves two, or more, relations. In addition, we demonstrated the importance of a join operation in a database decomposition.

However, little work has been done on the analysis of join operations. The most related result was the expected join size formula derived by Rosenthal [Ros81]. His proof process required two stringent conditions to derive the result in terms of the cardinalities of the join domain and source relations. First, the distributions of the join attribute values in source relations had to be independent and second, one of the distributions had to be uniform. We know that the assumptions of uniformity and independence of attribute values in relations are seldom satisfied in actual database systems. Christodoulakis [Chr81, 84] shows that these assumptions used in most analytic work may result in large estimation errors. In fact, he has also proved that they often lead to pessimistic estimations of the database cost. One of the significant results of this study was showing that Rosenthal's expression is still valid under much more general conditions through the use of the exact join size formula we developed.

In Chapter 2 we focused on the development of the join size formulae. After illustrating the strong impact of different join attribute value distributions on the outcome of a join operation, we used correlation analysis to measure the strength of the relationship. The experimental results led us to form the exact join size formula by using the covariance (expressed by the correlation coefficient and standard deviations) of the bivariate data along with the set of parameters used in Rosenthal's expression. This formula was analytically proved and empirically

verified by using a large set of all possible join attribute distributions with respect to the given sizes of the join domain and source relations. Two important characteristics of the exact join size formula were discussed. First, if the frequencies of join attribute values in the source relations are independent variables, then our expression is exactly the same as Rosenthal's expected join size expression because the correlation coefficient is zero. Second, the same result occurs when the join attribute values of either relation have a perfectly uniform distribution. In this case, the standard deviation of that distribution is equal to zero.

In fact, the formula we developed for the exact join size refines Rosenthal's formula by adding a correction term reflecting the actual distribution of the join attribute values. For practical applications on real database systems where the computation of statistical parameters can be expensive, we have derived an "expected" join size formula. The proof techniques we employed to show the expectation do not require the conditions assumed in [Ros81]. We have also shown the formula for computing the expected join size when source relations have unequal number of unique join attribute values. Finally, the expected join size formula for the resultant relations after the selection operations have been performed was presented. This is most important in query analysis.

Although the expected join size formulae are good enough for general applications, we would like to bound the estimation errors to ensure that they are within acceptable ranges. To this end, error analysis was performed in Chapter 3 to show the calculation of the estimation errors and their maximum values. Since the processing cost of obtaining the estimation error is precisely the same cost of computing the exact join size, we considered minimizing only the maximum estimation error to avoid using so many statistical variables. Several algorithms for estimating the correction term (i.e., the estimation error) of the exact join size were discussed in the next chapter.

To minimize the maximum estimation error of the expected join size, we have developed an operation called A-partition, which divides the join domain and source relations into disjoint subsets. Three approaches to implementing the A-partition were presented and their effectiveness has been proved. In the equal-sized subrelation (*ER*) method, the reduction of the maximum estimation error can be a factor of  $k$ , where  $k$  is the total number of partitions. While in theory we have proved that the performance of the equal-sized partition for the product of two subrelation sizes (*EP* method) is better than the partition of equal-sized subdomains (*ED* method), we proposed the latter as a practical approach in general applications. For example, an *ED* approach can be implemented easily and effectively in systems that have loosely coupled multiprocessors with partitioned databases [Kim84a] because the mechanism of partitioning fits perfectly into the structure of the database partitions. Although the cost of the partitioning methods can be justified mostly for key attributes only, the saving from avoiding the use of the statistical parameters is considerable.

In Chapter 4 we presented a two-step process for computing the correction term of the exact join size formula in a dynamic environment. Three schemes were used in the process according to the accuracy requirement versus processing cost. Again, the A-partition approach offers reasonable accuracy with controllable processing cost. Then we introduced the basic methods of computing the correlation coefficient and standard deviation by using frequency tables. A grouped frequency table was used in a less expensive method of estimating the standard deviation. It has been realized that in practice the cost of processing all join attribute values for all tuples in all relations is too expensive. Therefore, we used a random sampling method to approximate the join size and other statistical parameters. The formula for determining the sample size was derived by modifying Kolmogorov's formula in non-parametric statistics [Wal53, Dix69]. We also examined the correction factors for estimating the correlation coefficient and standard deviation when the sample

size is too small. Finally, we developed two numerical approximation methods to estimate standard deviations without using statistical variables. The primary concern was to minimize the processing cost and storage overhead.

Since selection operations are very often used in general queries, we examined in depth the expectations of the basic and compound selections in Chapter 5. First, we showed four methods of computing the probabilities for a tuple to satisfy a single selection term in a database relation. Simple probabilistic methods offer the simplest and most inexpensive approaches for attribute distributions that are approximately uniform and linear. For other distributions, the estimation error may be extremely large. On the other hand, attribute distribution tables provide the best estimation (if the frequencies of all attribute values are recorded, then we have the exact selection size) for any distribution at the expense of large storage overhead and high processing cost. While grouped frequency schemes may have a reasonably accurate estimate of the selection size with a controllable upper bound error, in practice the random sampling method is the most effective approach in a dynamic database since the processing cost and estimation accuracy are proportional to the sample size.

After developing the formulae for computing the expected sizes of compound selections formed by using the logical operators  $\wedge$  or  $\vee$ , we considered the cost of performing a selection operation. Since the processing cost is measured in terms of the expected number of disk block accesses, we showed two methods of translating the expected selection size into the number of blocks they occupy. Even though Yao's formula [Yao77] provides a better estimate for the expected block accesses of a selection, we chose to use Cardenas' formula [Car75] for its operational and expository simplicity. As a matter of fact, the discrepancy between the two approaches is significant only when the blocking factor is less than 10.

In Chapter 6 we have explored the practical applications for the expectations of selection and join operations on the access cost of queries. Several important issues in cost analysis were discussed first, such as the physical organization of relations and access methods. Then a very competitive partial-match retrieval system, the Indexed Descriptor Access Method, was used as the main vehicle to develop the access cost expressions for selection and join operations. Among the advantages stated in [Pfa80, Pfa82, Ram83], IDAM provides a relatively efficient method of performing conjunctive retrieval and has index files separated from data file completely. By using certain tuning techniques, it is possible to optimize the query processing.

By using the cost expression from [Pfa80] we derived the access cost formulae for compound selection and join operations in an IDAM system. A simplified method based on a nested-iteration algorithm [Sel79] was used to illustrate the implementation of a join operation and the computation of its access cost. A generalized version of the cost formula for general file systems was our ultimate result.

The result of this study has shown that our generalized mathematical models allow users to accurately compute the access cost of performing selection and join operations, and the expected size of the resultant relations in real world applications.

### 7.3. Further work

There are further studies to be done in this area. The first open question is: How does one accurately and efficiently compute the actual domain size of a join attribute? Proper calculation of the join domain size is crucial to the accuracy of the system. Especially, one must be able to record the independent variables in highly dynamic databases. The problem is much more complicated if intermediate

(or temporary) relations are considered in a sequence of relational operations.

Other directions for future research are in algorithms for minimizing the (maximum) estimation errors and in evaluating the overhead (vs. the performance improvement) of computing the correlation coefficient and standard deviations for the distribution of join attribute values between two relations.

The A-partitioning method is a good approach to minimize the maximum estimation error for the join size. The questions are how to decide on the possible candidates (all relations or some specific relations?) and what is a suitable way of partitioning for all the relations with the same join attribute? The strategies will be more complex when a specialized query processor is used to perform the join operation, such as a parallel pipelined JOIN PIPE in [Kim84b]. To explore the full potential of an A-partition, further performance analysis is needed in a distributed system with partitioned databases. One needs to answer similar questions to determine for which pairs of relations computation of their correlation coefficients is worthwhile.

## References

- [Aho79] Aho, A.V. and Ullman, J.D. "Optimal partial-match retrieval when fields are independently specified", ACM TODS, v.4.2 June 1979 (pp. 168-179)
- [Aho79a] Aho, A.V., et al. "The theory of joins in relational databases", ACM TODS, v.4.3 Sept. 1979 (pp. 297-314)
- [Bay72] Bayer, R. and McCreight E. "Organization and maintenance of large ordered indices", Acta Informatica 1.3, 1972 (pp. 173-189)
- [Bla77] Blasgen, M.W. and Eswaran, K.P. "Storage and access in relational data bases", IBM Systems Journal, v.16.4 1977 (pp. 363-377)
- [Bur70] Burington, R.S. and May, D.C. "Handbook of probability and statistics with tables", McGraw-Hill Book Company, 1970
- [Car75] Cardenas, A.F. "Analysis and performance of inverted data base structures", Comm. ACM, v.18.5 May 1975 (pp. 253-263)
- [Cha82] Chan, Arvola and Niamir, Bahram "On estimating the cost of accessing records in blocked database organizations", The computer Journal, v.25.3 1982 (pp. 368-374)
- [Che69] Chen, C.S. "Statistics I and II", Taiwan Sun-Wu Publishing Co., Taiwan, 1969
- [Che82] Cheung, T.Y. "Estimating block accesses and number of records in file management", Comm. ACM, v.25.7 July 1982 (pp. 484-487)
- [Che82] Cheung, T.Y. "Estimating block accesses and number of records in file management", Comm. ACM, v.25.7 July 1982 (pp. 484-487)
- [Chr81] Christodoulakis, S. "Estimating selectivities in databases" Ph.D. dissertation, Rep. CSRG-136, Computer Science Dept, U. of Toronto, 1981
- [Chr84] Christodoulakis, S. "Implications of certain assumptions in database performance evaluation", ACM TODS, v.9.2 June 1984 (pp. 163-186)

- [Cod70] Codd, E.F. "A relational model for large shared data banks", Comm. ACM, v.13.6 June 1970 (pp. 377-387)
- [Cod71] Codd, E.F. "Further normalization of the data base relational model" Data Base Systems, edited by Randall Rustin, Prentice-Hall, 1971 (pp. 33-64)
- [Cod71a] Codd, E.F. "Relational completeness of data base sublanguages", in Data Base Systems, Courant Computer Science Symposium 6, Prentice-Hall, May 1971 (pp. 65-98)
- [Cod74] Codd, E.F. "Recent investigations in relational database systems", IFIP Conf. 1974 (pp. 1017-1021)
- [Dix69] Dixon, W.J. and Massey, F.J. "Introduction to Statistical Analysis", McGraw-Hill Book Company, New York, 1969
- [Fre82] French, J.C. "An investigation of IDAM files", Ph.D. thesis, Univ. of Virginia, August 1982
- [Got75] Gotlieb, L.R. "Computing joins of relations", ACM SIGMOD Int. Conf. on MOD, May 1975 (pp. 55-63)
- [Hoe62] Hoel, P.G. "Introduction to mathematical statistics", 4th edition, Wiley publications, New York, 1962
- [Kim84a] Kim, W. "Highly available systems for database applications", Computing Surveys, v.16.1, March 1984 (pp. 71-98)
- [Kim84b] Kim, W.; Gajski, D.; and Kuck D.J. "A parallel pipelined relational query processor", ACM TODS v.9.2, June 1984 (pp. 214-242)
- [Kno75] Knott, G. D. "Hashing functions", Computer Journal, 8; August 1975 (pp. 265-278)
- [Knu73] Knuth, D.E. "The art of computer programming", Vol. 3, Addison-Wesley, Reading, Mass., 1973
- [Lea74] Leaver, R.H. and Thomas, T.R. "Analysis and presentation of experimental results", The Macmillan Press LTD., 1974
- [Lon70] Longley-Cook, L.H. "Statistical Problems and how to solve them", Barnes &



Noble Books, Inc., 1970

- [Mai83] Maier, D. "The theory of relational databases", Computer Science Press, Inc., Rockville, Maryland, 1983
- [Mey65] Meyer, P.L. "Introductory probability and statistical applications", 2nd edition, Addison-Wesley Publishing Co., Inc., 1965
- [Pfa79] Pfaltz, J.L. "Efficient multi-attribute retrieval over very large geographical data files", Proc. AUTO-CARTO IV. Washington, DC, 1979 (pp. 54-62)
- [Pfa80] Pfaltz, J.L. "Partial-match retrieval using indexed descriptor files", Comm. ACM, v.23.9 Sept. 1980 (pp. 522-528)
- [Pfa82] Pfaltz, J.L. "Computational and storage costs associated with indexed descriptor access", 1982
- [Pia84] Piatetsky-Shapiro, G. and Connell, C. "Accurate estimation of the number of tuples satisfying a condition", ACM Proceedings of SIGMOD conf., 1984 (pp. 256-274)
- [Ram83] Ramamohanarao K., et al. "Partial-match retrieval using hashing and descriptors", ACM TODS, v.8.4 Dec. 1983 (pp. 552-576)
- [Ric81] Richard, P. "Evaluation of the size of a query expressed in relational algebra", ACM Proceedings of SIGMOD Conf., 1981 (pp. 155-163)
- [Riv76] Rivest, R.L. "Partial-match retrieval algorithms", SIAM J. Computing, v.5.1 March 1976 (pp. 19-50)
- [Rob79] Roberts, C.S. "Partial-match retrieval via the method of superimposed codes", Proc. IEEE, v.67.12, Dec. 1979 (pp. 1624-1642)
- [Ros81] Rosenthal, A.S. "Note on the expected size of a join", ACM SIGMOD Record, v.11.4 July 1981
- [Sel79] Selinger, P.G.; Astrahan, M.M.; Chamberlin, D.D.; Lorie, R.A. and Price, T.G. "Access path selection in a relational database system", IBM research report no. RJ2429 (1979.1) and Proc. ACM SIGMOD 1979 (pp. 23-34)
- [Sev77] Severance, D.G. and Carlis J.V. "A practical approach to selecting record

- access paths", Computing Surveys, v.9.4 Dec. 1977 (pp. 259-272)
- [Sne56] Snedecor, G.W. "Statistical Methods", fifth edition, The Iowa State College Press, Ames, Iowa, 1956
- [Ull82] Ullman, J.D. "Principles of database systems", Computer Science Press, Inc. 1982
- [Wal53] Walker, H.M. and Lev, J. "Statistical Inference", Holt, Rinehart and Winston Inc., 1953
- [Wal78] Walpole, R.E. and Myers, R.H. "Probability and statistics for engineers and scientists", 2nd edition, Macmillan Publishing Co., Inc. 1978 (p. 304)
- [Wie77] Wiederhold, Gio "Database design", McGraw-Hill Book Company, 1977
- [Win75] Winkler, R.L. and Hays, W.L. "Statistics: probability, inference, and decision", Holt, Rinehart and Winston Inc., New York, 1975
- [Yao77] Yao, S.B. "Approximating block accesses in database organizations", Comm. ACM, v.20.4 April 1977 (pp. 260-261)
- [Zah83] Zahorjan, J. "Estimating block transfers when record access probabilities are non-uniform", Information Processing Letters 16, 1983 (pp. 249-252)