

INFORMATION TO USERS

This material was produced from a microfilm copy of the original document. While the most advanced technological means to photograph and reproduce this document have been used, the quality is heavily dependent upon the quality of the original submitted.

The following explanation of techniques is provided to help you understand markings or patterns which may appear on this reproduction.

- 1. The sign or "target" for pages apparently lacking from the document photographed is "Missing Page(s)". If it was possible to obtain the missing page(s) or section, they are spliced into the film along with adjacent pages. This may have necessitated cutting thru an image and duplicating adjacent pages to insure you complete continuity.**
- 2. When an image on the film is obliterated with a large round black mark, it is an indication that the photographer suspected that the copy may have moved during exposure and thus cause a blurred image. You will find a good image of the page in the adjacent frame.**
- 3. When a map, drawing or chart, etc., was part of the material being photographed the photographer followed a definite method in "sectioning" the material. It is customary to begin photoing at the upper left hand corner of a large sheet and to continue photoing from left to right in equal sections with a small overlap. If necessary, sectioning is continued again -- beginning below the first row and continuing on until complete.**
- 4. The majority of users indicate that the textual content is of greatest value, however, a somewhat higher quality reproduction could be made from "photographs" if essential to the understanding of the dissertation. Silver prints of "photographs" may be ordered at additional charge by writing the Order Department, giving the catalog number, title, author and specific pages you wish reproduced.**
- 5. PLEASE NOTE: Some pages may have indistinct print. Filmec as received.**

University Microfilms International

300 North Zeeb Road

Ann Arbor, Michigan 48106 USA

St. John's Road, Tyler's Green

High Wycombe, Bucks, England HP10 8HR

77-17,950

BERELIAN, Elias, 1946-
A METHODOLOGY FOR DATA BASE DESIGN IN A
PAGING ENVIRONMENT.

The University of Michigan, Ph.D., 1977
Computer Science

Xerox University Microfilms , Ann Arbor, Michigan 48106

A METHODOLOGY FOR DATA BASE DESIGN
IN A PAGING ENVIRONMENT

by
Elias Berelian

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer, Information and Control Engineering)
in The University of Michigan
1977

Doctoral Committee:

Professor K.B. Irani, Chairman
Professor K. Chuang
Assistant Professor A. Rosenthal
Assistant Professor T.J. Teorey
Professor R.A. Volz

ACKNOWLEDGEMENTS

I would like to thank Professor Keki B. Irani, chairman of my doctoral committee, for the invaluable guidance and suggestions he provided throughout the research leading to this dissertation. His patience, understanding and friendship are equally appreciated.

I thank members of my committee, Professors Chuang, Rosenthal, Teorey and Volz for the time, interest and talent they devoted to the progress of this research. Professor Rosenthal's suggestions were especially helpful for Chapter 4.

I am also grateful to: Jamshed Mulla for his help in various stages of programming, particularly for plotting programs, Annette Sizemore for typing Chapter 4, and Karen Chapin for her professional efficiency in typing the rest of the dissertation.

This research was sponsored primarily by the Rome Air Development Center at Griffiss Air Force Base under Contract Number F30602-76-R-0029.

Finally, I thank my wife, Esther, and my family. Their patience and understanding were invaluable. The continuing faith, encouragement and support of my parents, Mr. and Mrs. Faraj Berelian, throughout my graduate studies, are deeply appreciated.

Elias Berelian
April 1977

TABLE OF CONTENTS

	<u>Page</u>
ACKNOWLEDGEMENTS	ii
LIST OF ILLUSTRATIONS	iv
LIST OF TABLES	vii
LIST OF SYMBOLS	viii
 <u>Chapter</u>	
I INTRODUCTION	1
1.1 Motivation and Objectives	1
1.2 DBTG Proposal	8
II USER REQUIREMENTS MODEL	16
2.1 Data Item Types and Data Item Values	16
2.2 Data Base Relations	19
2.3 User Activity	29
2.4 Storage Space Parameters	39
III IMPLEMENTATION ALTERNATIVES AND COSTS	43
3.1 Implementation Alternatives	45
3.2 Constraint Conditions	92
3.3 Storage Costs	113
3.4 Time Costs	123
IV DATA BASE DESIGN	171
4.1 Problem Set Up	171
4.2 Undominated Choices	187
4.3 Undominated Solutions	198
4.4 Data Base Design	216
V RESULTS	223
5.1 Sensitivity of Time Cost Functions	223
5.2 Discussion of Resulting Data Base Designs	234
5.3 Storage/Time Trade-Off	252
5.4 Effects of Some Parameter Variations	263
VI CONCLUSIONS AND FURTHER RESEARCH	277
6.1 Conclusions	277
6.2 Further Research	279
REFERENCES	284

LIST OF ILLUSTRATIONS

Figure	<u>Page</u>
2.1 An Example Set SI of 24 Data Items I_i	18
2.2 An Example Relation $R(A,B)$	22
2.3 An Example Set SR of 26 Data Base Relation R_j .	24
2.4 Directed Graph Representation of SI and SR. . .	27
2.5 An Example Set SU of 4 Run Units.	36
3.1 A Listing of the 24 Implementation	47
3.2 Fixed Duplication of B under A.	50
3.3 Fixed Duplication of A under B	50
3.4 Variable Duplication of B under A	52
3.5 Variable Duplication of A under B	52
3.6 Fixed Aggregation of B under A	54
3.7 A Combination of Two Aggregations	54
3.8 Variable Aggregation of B under A	59
3.9 Chain with Next Pointers Association of B under A	59
3.10 Chain with Next and Owner Pointers Association of B under A	67
3.11 Chain with Next and Prior Pointers Association of B under A	67
3.12 Chain with Next, Prior and Owner Pointers Association of B under A	77
3.13 Pointer Array Association of B under A	77
3.14 Pointer Array with Owner Pointers Association of B under A	81
3.15 Dummy Record Association of A and B	87
3.16 Single Linkage of B under A	93
3.17 Double Linkage of A and B	93

Figure		Page
3.18	Multi-Level Record Array and Binary Tree Set Structures.	167
5.1	Data Base Description for a Storage Limit of 1.35 M Bytes, PGSZ = 2000 Bytes and MSPG = 2. .	235
5.2	Data Base Description for a Storage Limit of 1.3 M Bytes, PGSZ = 2000 Bytes and MSPG = 2 . .	240
5.3	Data Base Description for a Storage Limit of 1207050 M Bytes, PGSZ = 2000 Bytes and MSPG = 2.	242
5.4	Data Base Description for a Storage Limit of 1227650 M Bytes, PGSZ = 2000 Bytes and MSPG = 2.	244
5.5	Data Base Description for a Storage Limit of 1348250 M Bytes, PGSZ = 2000 Bytes and MSPG = 2.	246
5.6	Data Base Description for a Storage Limit of 1485350 M Bytes, PGSZ = 2000 Bytes and MSPG = 2.	253
5.7	Storage/Time Trade-off Graph.	256
5.8	Data Base Description for a Storage Limit of 1124000 M Bytes, PGSZ = 2000 Bytes and MSPG = 2.	259
5.9	Trade-off Graph for PTRS = 3.	264
5.10	Trade-off Graphs for Different MSPG Values where PGSZ = 2000 Characters.	265
5.11	Trade-off Graphs for Different MSPG Values where PGSZ = 4000 Characters.	267
5.12	Trade-off Graphs for Different PGSZ Values where MSPG = 1.	269
5.13	Trade-off Graphs for Different PGSZ Values where MSPG = 2.	270
5.14	Trade-off Graphs for Different PGSZ Values where MSPG = 4.	271
5.15	Trade-off Graphs for Different PGSZ Values where MSPG = 5.	272

Figure		<u>Page</u>
5.16	Trade-off Graphs for Different MSPG and PGSZ Values.	273
5.17	Trade-off Graphs for Different PTRS Values where MSPG and PGSZ are fixed	274

LIST OF TABLES

Table		Page
4.1.1	Number of Acceptable Implementations for R_j . .	180
4.3.1	Maximal Number of Undominated Choices for Stage k	214
5.1	Time Cost Error Estimates for MSPG = 5, PGSZ = 4000	229
5.2	Time Cost Error Estimates for MSPG = 4, PGSZ = 4000	229
5.3	Time Cost Error Estimates for MSPG = 2, PGSZ = 4000	229
5.4	Time Cost Error Estimates for MSPG = 1, PGSZ = 4000	229
5.5	Time Cost Error Estimates for MSPG = 5, PGSZ = 2000	230
5.6	Time Cost Error Estimates for MSPG = 4, PGSZ = 2000	230
5.7	Time Cost Error Estimates for MSPG = 2, PGSZ = 2000	230
5.8	Time Cost Error Estimates for MSPG = 1, PGSZ = 2000	230
5.9	Time Cost Error Estimates for MSPG = 25, PGSZ = 4000	232

LIST OF SYMBOLS
(in the order of their appearance)

<u>Symbol</u>	<u>Description</u>
CHAR	A character set
I_i	A data item type
SI	The set of data item types of an application
NI	Cardinality of SI
ITNM(i)	Name of $I_i \in SI$
ITSZ(i)	Size (number of characters) of $I_i \in SI$
ITCR(i)	Cardinality of $I_i \in SI$
$R_j(A, B)$	A data base relation between two data item types $A \in SI$ and $B \in SI$
a, b	Values of data item types A, B respectively
R_j	A data base relation without explicit reference to component data item types
SR	The set of data base relations of an application
NR	Cardinality of SR
ORG(R_j)	Origin data item type of relation R_j ; e.g. if $R_j = R_j(A, B)$ then $ORG(R_j) = A$
DST(R_j)	Destination data item type of relation R_j ; e.g. if $R_j = R_j(A, B)$ then $DST(R_j) = B$
r_j	Cardinality of $R_j \in SR$ (number of ordered pairs $\langle a, b \rangle \in R_j(A, B)$, where $a \in A$ and $b \in B$)
$\langle m_j, \bar{m}_j, M_j \rangle$	An ordered triple representing the minimum, average and maximum number of B-values related to one A-value in R_j , respectively
$\langle n_j, \bar{n}_j, N_j \rangle$	An ordered triple representing the minimum, average and maximum number of A-values related to one B-value in R_j , respectively

<u>Symbol</u>	<u>Description</u>
α_j', β_j'	Cardinalities of data item types $A = \text{ORG}(R_j)$ and $B = \text{DST}(R_j)$, respectively
α_j'', β_j''	Sizes of data item types $A = \text{ORG}(R_j)$ and $B = \text{DST}(R_j)$, respectively
$\alpha_j(\beta_j)$	Number of A(B)-values related to at least one B(A)-value in R_j
$\text{RIM}_j(a)$	Right image of $a \in A$ in $R_j(A, B)$, i.e. $\{b \in \text{DST}(R_j) \mid \langle a, b \rangle \in R_j\}$
$\text{LIM}_j(b)$	Left image of $b \in B$ in $R_j(A, B)$, i.e. $\{a \in \text{ORG}(R_j) \mid \langle a, b \rangle \in R_j\}$
$\text{RM12}(j)$	$\langle m_j, \bar{m}_j, M_j \rangle$
$\text{RM21}(j)$	$\langle n_j, \bar{n}_j, N_j \rangle$
$\text{IORC}(i)$	Item origin count; number of $R_j \in \text{SR}$ such that $I_i = \text{ORG}(R_j)$
$\text{IDSC}(i)$	Item destination count; number of $R_j \in \text{SR}$ such that $I_i = \text{DST}(R_j)$
SINR	The set of sink relations; i.e. $\text{SINR} = \{R_j \in \text{SR} \mid \text{IDSC}(i) = 1 \text{ and } \text{IORC}(i) = 0 \text{ where } I_i = \text{DST}(R_j)\}$
SONR	The set of source relations; i.e. $\text{SONR} = \{R_j \in \text{SR} \mid \text{IORC}(i) = 1 \text{ and } \text{IDSC}(i) = 0 \text{ where } I_i = \text{ORG}(R_j)\}$
$0 = \langle \text{op}, B, A, R, f \rangle$	An operation; op is the operation code of 0, $B \in \text{SI}$ is the operand of 0, $A \in \text{SI}$ is the qualifier of 0, $R = R(A, B)$ or $R = R(B, A)$ and f is the frequency of 0
SOP	The set of all possible operation codes op
RU_k	Run unit number k; a sequence of operations
SU	The set of run units of an application
NU	Cardinality of SU
NO_k	Number of operations in RU_k
O_ℓ^k	ℓ -th operation of k-th run unit
RUUR_j	Number of run units that use relation R_j

<u>Symbol</u>	<u>Description</u>
PSI_k	The set of data items used by k-th run unit RU_k
PSR_k	The set of relations used by the k-th run unit RU_k
TMSB	Maximum storage bound
PGSZ	Size of a page
MSPG	Number of data pages in main storage
PTRS	Size of a pointer
C PTRS	Size of a counter
PLOKS	Size of a privacy lock
fd(A)	A fixed-length vector of duplicate A-values
vd(B)	A variable-length vector of duplicate B-values
fa(B)	A fixed-length vector of B-values
va(A)	A variable-length vector of A-values
mp, op	Member and owner pointers, respectively
np, pp	Next and prior pointers, respectively
MPL	An integer, $1 \leq MPL \leq 24$ representing an implementation number
CC(MPL, c)	Constraint condition number c for implementation MPL
AC(MPL, j)	A variable which is equal to one if implementation MPL is acceptable for relation R_j , zero otherwise
SC(MPL, j)	Storage cost of implementation MPL for relation R_j
MMSB	$= \sum_{i=1}^{NI} ITSZ(i) * ITCR(i)$
OMSB	$= TMSB - MMSB$
NATD, NATS	Numbers of access types to data items and data base sets, respectively

<u>Symbol</u>	<u>Description</u>
TLD_j, TLS_j	Total lock sizes for single linkage and association implementations of relation R_j , respectively
$TC(MPL, j)$	Time cost of implementation MPL for relation R_j
C	A configuration; $C = \{ \langle R_j, MPL_j \rangle \mid j = 1, 2, \dots, NR; R_j \in SR, 1 \leq j \leq 24 \}$
$CS(C), CT(C)$	Cumulative storage and time costs, respectively, of configuration C
$TC(MPL, j, O_\ell^k)$	Time cost of implementation MPL for relation R_j in (the ℓ -th operation of k -th run unit) O_ℓ^k
$p_1, \dots, p_4$	Page fault probabilities (p_1 for direct access, p_2 for next or prior access, p_3 for related record access and p_4 for a privacy lock access)
$PBSZ$	Size of the page buffer; $PBSZ = PGSZ * MSPG$
ARL	Average record length
$DRSZ$	Size of a dummy record
$IMP(j)$	$= \{MPL \mid AC(MPL, j) = 1\}$; the set of acceptable implementations for relation R_j
$PC(SSR)$	A partial configuration for a subset SSR of SR
$OVF_j(s)$	Objective (or optimal) value function; optimal j -stage solution to state s
$q = \langle MPL, S, T \rangle$	A choice for relation R
Q_j	The maximal set of undominated choices for R_j , ordered in increasing sequence of storage costs
$RL(k)$	The ordinal number in SR of the relation associated with stage k
Q^k	$\stackrel{d}{=} Q_{RL(k)}$
$NCHS(k)$	Number of elements of Q^k
$FMP(i, k)$	First coordinate of the i -th element q_i^k of Q^k

<u>Symbol</u>	<u>Description</u>
BASC, BATC	Sums of storage and time costs, respectively, of relations with only one undominated choice
FSR	$= \{R_j \mid R_j \in SR \text{ and } IMP(J) = 1\}$; the set of relations with only one undominated choice
FPC	The optimal partial configuration for FSR
VSR	The set of relations defined by the set theoretic difference: $VSR = SR - FSR$
MR	$= VSR $
VPC	A partial configuration for VSR
$z^k = \langle pc^k, S^k, T^k \rangle$	A k-stage solution
Z^k	The maximal set of undominated k-stage solutions z^k
NS(k)	Number of elements of the maximal set of undominated solutions to stage k
$\mu_1, \dots, \mu_4$	Coefficients of $p_1, \dots, p_4$ in time cost functions
δ_l	Percentage variation of time costs, TC, for a variation Δl in the average record length
γ	Average cardinality of data items in SI
δ_s	Percentage variation of time costs, TC, for a variation ΔS in the storage requirements

CHAPTER I

INTRODUCTION

1.1 Motivation and Objectives

The application of computer systems to store the operational data of large organizations has become an important part of such organizations' activities, in recent years. The benefits of using computerized data base systems have been widely recognized and well documented [Date 1975], [Beck 1976], [GUIDE-SHARE 1970], [Berg 1976]. The most important benefit of an integrated data base is that it provides centralized control of data. The advantages of central data control, among others, are that it reduces redundancies in data storage, facilitates sharing of data among multiple applications, enables the data base administrator to enforce standards in data representation, and provides means to apply security restrictions and maintain data integrity. Another important advantage of central data control is that it enables the data base administrator to structure the data base system so that the overall performance is best for the organization rather than for any individual application. In particular, the data base administrator can choose a structure for data storage that models the requirements of the organization in terms of data values and relationships and at the same time can optimize the processing performance of a selected set of applications whose combined performance is critical to

overall system performance.

The designer of the data base structure has knowledge of the basic entities about which information has to be stored in the data base. He also is aware of the associations that relate these basic entities. These associations, being as important to the organizations' informational needs as the basic entities they relate, must be represented in the data base. The realization of this representation may be accomplished by physical contiguity in storage, by pointers or by any other method. Different methods, however, result in different storage space and processing time requirements. It is clear that a large number of data base designs are available where each has advantages and disadvantages with respect to a given performance objective and that no absolutely optimal design exists for a particular set of user requirements.

The objective of this research is to develop a methodology for automatic design of the logical structure of a pagged data base. The methodology may be used, by a data base designer as a design tool, to investigate the space/time trade-offs of a large number of logical structures that can model the information content of a given set of user requirements. The approach is to map a high level description of user requirements in terms of individual data items and data base relations into a set of record structures, record relationships and storage structures.

Intra- and inter-record structures generated by this design method conform to the specifications published in the Data Base Task Group (DBTG) report of the Conference on Data Systems Languages [CODASYL 1971]. The data base is assumed to be accessed in a paging environment. The mapping will be such that the resulting data base structure is optimal (over a certain class of DBTG structures) in the sense that it produces a minimum expected number of page faults for a pre-specified set of user activities, subject to storage and security constraints.

The user requirements model presented in Chapter 2 provides the means for specifying the basic data items and relations between pairs of data items in a statistical description, namely one that requires information about names, sizes and number of occurrences rather than actual values. This model also provides a set of 12 data manipulation operations in terms of which expected frequencies and types of data retrieval and update (user activities) may be expressed. Other components of this model allow for the description of user's storage space requirements in terms of parameters such as: total allowable storage limit, and sizes of pointers, counters, privacy locks and memory pages. In the following we will refer to a given specification of all parameters of the user requirements model as an application.

In Chapter 3 we will define a set of 24 implementation

alternatives which can be used to implement data base relations, in such a way that the collection of exactly one implementation for each relation defined in the application determines the overall structure of the data base. The type of the relation implementation determines whether the relationship between component data item types will be realized in the data base by physical adjacency or by pointers through the use of data base sets or any other mechanism. Also in the case where data base sets are to be used the type of set implementation technique is determined by the type of the relation implementation.

A two-component cost function will be developed in Chapter 3 for relation implementations that are acceptable, namely those that satisfy certain constraint conditions. Constraint conditions act as a filter to prevent the selection of unconformable DBTG structures. The first cost component is the storage cost of a relation implementation and consists of the storage space necessary to store data item values, counter and pointer variables and security locks. The second cost component is called the time cost of a relation implementation measured in the number of page faults expected to occur when a certain set of data manipulation operations is executed. Time cost functions will be defined in terms of probabilities of a certain number of page fault categories. Probability of a page fault in each category will be computed from the variables

describing the application.

The task of the optimization algorithm is, then, to select exactly one implementation for each relation such that the total time cost is minimized, total storage cost is bounded by some given value and certain constraint conditions are satisfied. This algorithm is presented in Chapter 4 and it treats the optimization problem as a discrete, multi-stage decision process. For each relation defined in the application, acceptable implementations and their storage and time costs are first determined using the models of Chapter 3. Then a sequence of "undominated choices" for each relation is generated. If, for any relation, this sequence consists of only one choice then the relation is excluded from further consideration; the optimal choice of implementation for that relation is fixed and known. The next phase of the decision process, then, treats each remaining relation (relations with two or more undominated choices) as one decision "stage". In stage k of the second phase, a sequence of "undominated solutions" to stage $(k+1)$ is generated from the sequence of undominated solution to stage k and undominated choices of stage $k+1$. The sequence of undominated solutions to the first stage is identical to the sequence of undominated choices of stage 1 and the undominated solutions to the final stage gives a spectrum of solutions where each one is optimal over a certain range of storage space limits.

Results obtained by applying our data base design methodology to an example application introduced in Chapter 2 will be discussed in Chapter 5. This chapter includes the results of a sensitivity analysis for time cost functions with respect to certain parameters. A number of data base designs, each optimal under a particular set of conditions, will be discussed and their storage/time trade-offs will be shown. Finally, the effect of varying some of the important parameters of the model on the outcoming data base designs will also be discussed in Chapter 5.

Other methodologies for the design of DBTG data structures have been reported by Gerritsen [1975], Mitoma [1975], Bubenko [1976], etc. While the main goals of these research efforts are the same, namely the design of DBTG data structures, they have substantial differences in the modelling of the real problem, their approach in arriving at a solution and most importantly in their performance objectives. Gerritsen [1976] developed an automatic data base "designer" capable of designing a "satisfactory" data base structure for a set of anticipated queries. These queries constitute the input to the designer system. A series of assertions are produced from the input and then a set of records and record relationships is constructed such that all of the assertions are satisfied. The possibilities of occurrence of cases where several alternative designs may satisfy

a given set of assertions are not considered and, consequently, criteria for selection of one data structure design from among several candidates are not presented in [Gerritsen 1976].

Bubenko [1976] takes a different view of data base schema design and proposes the design of two extreme alternatives followed by successive refinements and evaluations to arrive at a compromise. At one extreme, data will be stored on initial (transaction) level and answers to queries are derived when necessary. This extreme minimizes storage space requirements at the expense of long response time. At the other extreme, information is organized such that the response to every anticipated query is explicitly stored and modified with every transaction that affects it. In the evaluations of alternative solutions factors such as secondary storage space requirements, number of data base accesses for updates and number of data base accesses for answering queries are considered. A certain degree of redundancy in data storage is to be expected in the data base designs produced by this methodology. The degree of this redundancy depends on the proximity of the final solution to the two extremes. This is because information in one record occurrence can "be derived from information in record occurrences to which it is an owner".

Mitoma [1975] gives a methodology for data base schema design. The performance objective employed in that work is the number of record accesses required over a period of

time to process an anticipated set of operations. Record and set definitions are designed so as to minimize the number of record accesses subject to storage and feasibility constraints. For data base set types generated by this design methodology, specifications for LINKED TO OWNER and PRIOR PROCESSABLE are determined by the design, although the specific techniques employed to implement data base sets are not considered. The effect of different alternative techniques for the implementation of data base sets on the performance of data base systems has been recognized, for example, in [Bachman 1974].

1.2 DBTG Proposal

The CODASYL Data Base Task Group specifications for data base management systems, published in [CODASYL 1971], constitute a major development in data base technology. Concepts presented by these specifications have been used in several commercially available data base management systems including DMS1100 [UNIVAC 1974], DBMS-10 [DEC], IDMS [Cullinane] and IDS [Honeywell], and are expected to influence future national standards in data base technology. The DBTG proposal is composed of specifications for three languages: schema Data Definition Language (schema DDL), a sub-schema Data Definition Language (sub-schema DDL) intended for use with COBOL and a Data Manipulation Language (DML). The COBOL and non-COBOL portions of the original DBTG report were later extended and modified by

Data Base Language Task Group (DBLTG) and Data Definition Language Committee (DDLc), respectively. CODASYL COBOL Data Base Facility Proposal [CODASYL 1973a] is DBLTG's report and consists of a detailed proposal for a COBOL DML and sub-schema DDL which is basically the same as DBTG proposal [CODASYL 1971] with some differences in syntactic details. DDLc's report appeared in June 1973 as CODASYL DDLc Journal of Development [CODASYL 1973b]. As stated in the report, "the DDLc limited its efforts primarily to clarification of, rather than extension to, the base document."

In this section we will briefly review those components of the DBTG data definition model that are relevant to our work. The reader is referred to [CODASYL 1971], [CODASYL 1973a] and [CODASYL 1973b] for a more complete description and to [Date 1975] and [Taylor 1976] for a tutorial treatment of the underlying concepts. The major DBTG data definition concepts that are relevant to our work are data items, data aggregates (vectors and repeating groups), records, data base sets and privacy locks.

A data item is the smallest unit of named data. A contiguous sequence of storage locations that contains the value of a data item is called an occurrence of that data item. All occurrences of a data item type occupy the same number of storage locations called the size of the data item type. For example, an occurrence of the data item

type named EMPBRT (employee birth) and of size 8 may be a sequence of 8 storage locations containing "06/29/56". Data items can also take on null values or pointer (data-base-key) values.

Data aggregates are collections of data items within a record that can be referenced as a group using the name of the data aggregate. There are two kinds of data aggregates: vectors and repeating groups.

A vector is a one-dimensional array of data items of the same data item type, similar to a FORTRAN array. Vectors appearing at the first level of a record definition can be of variable-length, where the number of elements of the vector may be specified by the value of a counter data item in the same record. A fixed-length vector may be defined at any level of record definition and the number of elements of the vector is fixed at a value specified at data base design time.

A repeating group is a collection of data occurring repeatedly in a record. A repeating group may contain data items, vectors or repeating groups. This provides the capability of defining nested repeating groups. The number of occurrences of a repeating group (in a record) can be specified by the value of a (counter) data item in the record if this number is variable. However, just as in the case of vectors, variably dimensioned repeating groups can occur only in level one of a record definition.

In other words, a variably dimensioned repeating group cannot be defined as part of another repeating group.

A record is the unit of access in a DBTG system. It can consist of data items, vectors and repeating groups of fixed- or variable-lengths. Records with identical structure are grouped into record types. Number of records of a given type need not be explicitly specified anywhere in the data base, in contrast to repeating groups for which the number of occurrences, in the containing repeating group or record, is given either at data definition time (fixed-length) or as the value of a data item in the containing record (variable-length). Records are uniquely identifiable by their data-base-keys. A record may be directly accessed if its data-base-key is known, whereas a repeating group is accessible only when the record containing it is available to the executing program. Record types may be related to each other, using data base sets, to implement complex network-type associations. However, within a record only hierarchical associations can be accommodated (using repeating groups).

A data base set is a collection of related records. One of the records in the set is called the owner, and the remaining (zero or more) records are called member records of the set. The existence of a set is established by the existence of its owner record and it is "empty" if it does not contain any member records. Owner and member records of the same set cannot be of the same type. Sets

of identical structure are grouped into set types. A record type may be declared as member record type in many set types and at the same time as owner record type in many other set types. However, a record (occurrence) cannot belong to (participate as owner or member in) more than one set (occurrence) of a given type. It is evident from the foregoing that a data base set type can only implement one-to-many relationships between occurrences of its owner and member record types. A set type may be declared with more than one member record type.

Protection of data is achieved through a system of locks and keys. Privacy locks may be declared in the schema for different types of accesses to sets, records, data items, etc. Run units seeking access to data should provide privacy keys (at execution time) and each key is simply compared with the corresponding lock for a match. Multiple locks may be specified for a given pair (access type, resource). A request for that pair is granted if the key provided by the run unit matches any one of the locks. Privacy locks and keys may be defined as procedures (or results of procedures) that provide the value of a key or take on some action.

A run unit is an execution of one or more programs, written in some procedural language (host language) augmented to support a set of Data Manipulation Language (DML) commands. A complete description of a data base in terms of DDL entries is the schema for the data base. It includes

definitions of all record types and their component data items and data aggregates, set types, privacy locks, etc. A sub-schema is a "consistent and logical" subset of the schema from which it is derived. The concept of sub-schema is employed to separate the description of the entire data base, as known to the data base administrator, from the description of portions of it, as he wishes to be known by individual programs. A sub-schema is invoked by a run unit using a DML command. This invocation causes the automatic definition of a User Working Area (UWA) for the run unit. A location is provided in UWA for each data item included in the sub-schema. Each such data item may be referenced by the program using its name as declared in the sub-schema—which may be different from the one declared in the schema.

In our data base design descriptions discussed in Chapter 5 we will not give the exact schema for the data base, namely a syntactically correct set of record and set declarations written in DDL statements. The data base design descriptions will, however, consist of the following information from which a designer can easily derive all schema definitions concerning the logical structure of records and sets.

All record types of the data base are determined and for each record type all—if any—data item types, vectors and repeating groups contained in it are determined. Size

of each data item type, number of occurrences of each repeating group and number of elements of each vector are given as a constant (for fixed-lengths) or as the value of another data item type (for variable-lengths).

All set types of the data base are determined and for each set type its owner record type and its member record type are given. For each set type, it is also specified how occurrences of sets of that type should be implemented. Set implementation techniques considered are: chain, chain with owner pointers, chain with prior pointers, chain with owner and prior pointers, pointer array and pointer array with owner pointers.

In this data definition design we are primarily concerned with the logical record and set structures. We are not attempting to model and design all of the features included in a DBTG data model. In particular, the following non-structural data model components will not be modelled and designed.

- i) Groupings of records in AREAS (REALMS).
- ii) Access method specifications such as SINGULAR SET declarations, INDEX and SEARCH KEY declarations and CALC KEY definitions.
- iii) Data sub-model (sub-schema) specifications.
- iv) LOCATION MODE specifications for records.
- v) Some data item characteristics such as VIRTUAL or ACTUAL RESULT or SOURCE specifications,

ENCODING/DECODING and validity- or error-checking (CHECK and ON clauses).

- vi) Some data base set characteristics such as membership classes (MANDATORY/OPTIONAL or AUTOMATIC/MANUAL) and SET SELECTION criteria.

Further limitations and simplifying assumptions made for this work will be explained where those assumptions are used, e.g., assumptions regarding the page fault probability model are given in 3.4.1. Some of the constraint conditions of Section 3.2 are only sufficient (rather than both necessary and sufficient). The necessity and effects of adopting those conditions are discussed in Section 3.2. Finally, some areas of extension to the implementation alternatives and reasons for not incorporating them into our model are discussed at the end of Chapter 3.

CHAPTER II

USER REQUIREMENTS MODEL

In this chapter we will formally define a model capable of describing the informational requirements of an application at a level where the ultimate record and set structures of the data base are not known. It is in terms of the parameters of this model that the data base designer specifies an application. Implementation models of Chapter 3 will then use this information to analyze different implementation alternatives by associating storage and time costs to each alternative. The optimization algorithm of Chapter 4, then makes all the record and set structuring decisions to develop an optimized data base design.

In Section 2.1 we will discuss the first component of the user requirements model, namely the concepts of data item types and data item values. Next in Section 2.2 we will discuss data base relations. In Section 2.3 we will discuss the component of the model through which data access forms and frequencies are specified in terms of a certain number of operations using data base relations. The last component of the user requirements model concerns the storage space requirements discussed in Section 2.4.

2.1 Data Item Types and Data Item Values

A data item value is a string of characters chosen from a given character set with a known ordering. The

character set may be, for example, the set of EBCDIC characters with the conventional ordering. The size (or length) of a data item value is the number of characters it contains. For example, if $\text{CHAR} = \{a_1, a_2, \dots, a_m\}$ is a character set ordered such that $a_j < a_{j+1}$, $j=1, \dots, m-1$ then $d_1 = a_{i_1} a_{i_2} \dots a_{i_s}$ where $a_{i_j} \in \text{CHAR}$, $j = 1, \dots, s$ is a data item value of size s . A named set I of data item values d_i of equal size is called a data item type. Cardinality of this set is the cardinality of the data item type.

$$I = \{d_i \mid d_i \text{ is a data item value of size } \alpha", \\ i = 1, 2, \dots, \alpha'\}$$

$\alpha"$ is the size of I

α' is the cardinality of I ($|I|$)

Data items d_i , $i = 1, \dots, \alpha'$ of the same type I are ordered in the conventional way, i.e., if $d_i = a_i^1 a_i^2 \dots a_i^{\alpha'}$ and $d_j = a_j^1 a_j^2 \dots a_j^{\alpha'}$ then $d_i < d_j$ iff for some $k \in \{1, 2, \dots, \alpha'\}$, $a_i^k < a_j^k$ and $a_i^p = a_j^p$ for all $p < k$.

The set of all data item types of an application is denoted by $SI = \{I_1, I_2, \dots, I_{NI}\}$. The following variables characterize each data item type $I_i \in SI$, $i = 1, 2, \dots, NI$.

```
ITNM(i)      /* Name of the data item type  $I_i$  */
ITSZ(i)      /* Size of  $I_i$  */
ITCR(i)      /* Cardinality of  $I_i$  */
```

Figure 2.1 illustrates an example set of $NI = 24$ data

<u>i</u>	<u>ITEM NAME</u>	<u>SIZE</u>	<u>CARDINALITY</u>
1	ORGCOD	10	50
2	ORGNAM	30	50
3	BUDGET	6	50
4	JOB COD	10	500
5	ATHQNT	6	500
6	ATHSAL	6	1000
7	ATHMIN	6	1000
8	ATHMAX	6	1000
9	DDUCTN	15	2000
10	JOB TTL	60	500
11	EMPNUM	25	5000
12	EMPNAM	30	5000
13	JHSYRS	2	30
14	BRTDAT	8	5000
15	EMPLVL	2	20
16	EMPADR	100	5000
17	EMPMST	2	10
18	OFR COD	25	100
19	OFR FMT	3	10
20	OFR DAT	8	50
21	OFR LOC	10	50
22	COUNUM	15	50
23	COUTTL	60	50
24	COUDSP	250	50

Figure 2.1 An Example Set SI of 24 Data Items I_i

item types of an application. The sixteenth data item type I_{16} is, for example, called EMPADR and consists of 5000 employee addresses each of size 100 characters. The tenth data item type I_{10} is called JOBTTL and consists of 500 job titles each of size 60 characters, and so on. The exact values of data items, for example the 500 job titles, have no significance in our optimization problem and therefore they need not be specified.

2.2 Data Base Relations

Associations between different pairs of values of data items are modelled by the concept of a data base relation. If A and B are two data item types in SI then a relation $R(A,B)$ over A and B is a named set of ordered pairs $\langle a, b \rangle$ where a and b are data item values of data item types A and B, respectively. We call A and B the origin and destination data item types of relation R, respectively, and denote them as follows.

$$A = \text{ORG}(R), \quad B = \text{DST}(R)$$

In set theoretic terms $R(A,B)$ is a subset of the cartesian product $A \times B$. For our purpose the specification of a data base relation R_j over data item types A and B is as follows:

$$R_j = R(\text{RNAME}, A, B, r_j, \langle m_j, \bar{m}_j, M_j \rangle, \langle n_j, \bar{n}_j, N_j \rangle)$$

where

$j \in \{1, 2, \dots, \text{NR}\}$ is the ordinal number of relation R_j in the set SR of all

	relations of an application and
	$NR = SR $
$RNAME = RNAME(j)$	is the name of the relation R_j
$A = ORG(R_j)$	is the origin data item type of R_j
$B = DST(R_j)$	is the destination data item type of R_j
r_j	is the number of ordered pairs in R_j and is called the cardinality of the relation R_j
$\langle m_j, \bar{m}_j, M_j \rangle$	is an ordered triple representing the minimum, average and maximum number of B-values related to one A-value in R_j , respectively, and the average is taken over the A-values related to at least one B-value in R_j
$\langle n_j, \bar{n}_j, N_j \rangle$	is an ordered triple representing the minimum, average and maximum number of A-values related to one B-value in R_j , respectively, and average is taken over B-values related to at least one A-value in R_j .

We will use the notation $R_j(A, B)$ (instead of simply R_j) to refer to the j -th relation in SR wherever it is

necessary to identify the origin and destination data item types of the relation. In such cases we will use the six sets of variables α_j , α'_j , α''_j , β_j , β'_j and β''_j with the following definitions.

α'_j, β'_j are the cardinalities of the data item types $A = \text{ORG}(R_j)$ and $B = \text{DST}(R_j)$, respectively.

$\alpha_j(\beta_j)$ is the number of $A(B)$ -values related to at least one $B(A)$ -value in R_j .

α''_j, β''_j are the sizes of data item types A and B , respectively.

It is clear from the above definitions that $\bar{m}_j$ and $\bar{n}_j$ are given by the following.

$$\bar{m}_j = \frac{r_j}{\alpha_j}, \quad \bar{n}_j = \frac{r_j}{\beta_j}$$

Furthermore, if $m_j \neq 0$ then every A -value is related to at least one B -value in R_j and therefore $\alpha_j = \alpha'_j$. Similarly $n_j \neq 0$ implies $\beta_j = \beta'_j$.

For every element $a \in \text{ORG}(R_j)$ the set of data item values $\{b \in \text{DST}(R_j) \mid \langle a, b \rangle \in R_j\}$ is called the right image of a in R_j denoted by $\text{RIM}_j(a)$. Elements of $\text{RIM}_j(a)$ are assumed to be ordered with the same ordering as defined for data item values of type B , for all $a \in A$, so that we can refer to the i -th B -value related to an A -value in relation $R_j(A, B)$. Similarly the left image of $b \in \text{DST}(R_j)$ in R_j is defined as:

$$\text{LIM}_j(b) = \{a \in \text{ORG}(R_j) \mid \langle a, b \rangle \in R_j\},$$

which is again an ordered set of A-values. Figure 2.2 illustrates some of the above definitions for a simple example relation $R(A, B) = \{ \langle a_1, b_1 \rangle, \langle a_2, b_1 \rangle, \langle a_2, b_2 \rangle, \langle a_2, b_3 \rangle, \langle a_3, b_3 \rangle, \langle a_3, b_4 \rangle, \langle a_4, b_3 \rangle, \langle a_5, b_3 \rangle \}$.

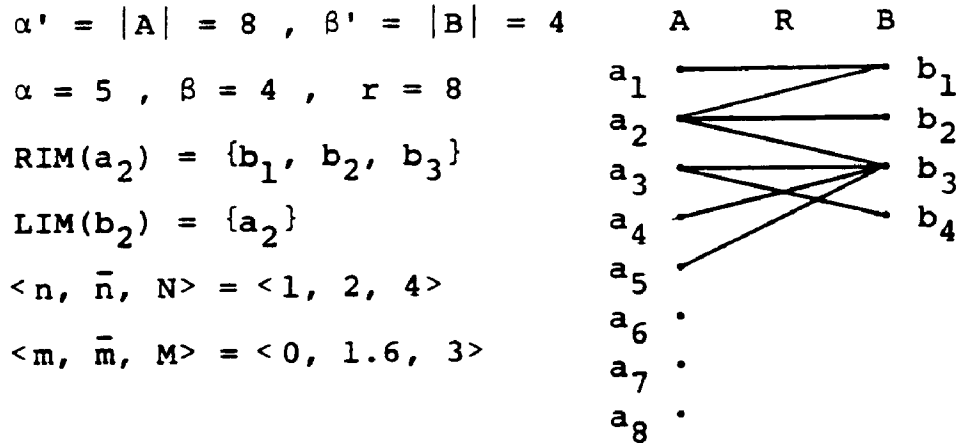


Figure 2.2 An Example Relation $R(A, B)$

The set of all data base relations for a data management application is denoted by SR and its cardinality is denoted by NR :

$$SR = \{R_1, R_2, \dots, R_{NR}\}.$$

Figure 2.3 shows an example of 26 data base relations defined over the 24 data item types shown in Figure 2.1. We make the assumption that each data item value of every data item type $I_i \in SI$ is related to at least one data item value in some relation $R_j \in SR$. In other words for all $I_i \in SI$ and for all $a \in I_i$ there exists $R_j \in SR$ and b such that either $\langle a, b \rangle \in R_j$ or $\langle b, a \rangle \in R_j$.

In Figure 2.3 the 26th relation R_{26} , for example, is called OFROFCOU. This relation is intended to associate offering codes OFRCOD of all offerings of each course to

its course number COUNUM. The origin and destination data item types of this relation are called COUNUM and OFRCOD, respectively. There are 100 ordered pairs $\langle \text{counum}, \text{ofrcod} \rangle$ in R_{26} where counum and ofrcod are data item values of data item types called COUNUM and OFRCOD, respectively. The first multiplicity vector $\langle m_{26}, \bar{m}_{26}, M_{26} \rangle$ is equal to $\langle 0, 2, 10 \rangle$ which means that a course has at least zero and at most 10 offerings and that on the average it has two offerings. As another example relation R_{23} called STUDENTS associates employee numbers of employees enrolled in an offering (of a course) to its offering code. There are 2100 pairs in R_{23} . At least 1, at most 50 and on the average 21 students (employees) are enrolled in each offering. Some employees are not enrolled in any offering ($n_{23} = 0$), however, among those enrolled each student is enrolled in an average of $\bar{n}_{23} = 2$ and a maximum of $N_{23} = 5$ offerings. This is an example of a many-to-many relation, while relation R_1 , called NAMOFORG is an example of a 1-to-1 relation.

In order to facilitate the development of our constraint conditions in Chapter 3 we will now classify relations $R_j \in SR$, $j = 1, 2, \dots, NR$ into eight types numbered 0 through 7, based on their multiplicity vectors

$$RM12(j) = \langle m_j, \bar{m}_j, M_j \rangle \text{ and } RM21(j) = \langle n_j, \bar{n}_j, N_j \rangle.$$

Relation $R_j(A, B) \in SR$ is of type zero if $RM12(j) = RM21(j) = \langle 1, 1, 1 \rangle$. This is a one to one relationship between all A-values and all B-values. In set theoretic terms R_j in this case is an invertible function

j	NAME	ORIGIN	DSTIN.	CARDIN.	$\langle m_j, \bar{m}_j, M_j \rangle$	$\langle n_j, \bar{n}_j, N_j \rangle$
1	NAMOFORG	ORGCOD	ORGNAM	50	$\langle 1, 1, 1 \rangle$	$\langle 1, 1, 1 \rangle$
2	BGTOFORG	ORGCOD	BUDGET	50	$\langle 1, 1, 1 \rangle$	$\langle 1, 1, 1 \rangle$
3	JOBFOFORG	ORGCOD	JOB COD	500	$\langle 1, 10, 20 \rangle$	$\langle 0, 1, 1 \rangle$
4	QNTOFJOB	JOB COD	ATHQNT	500	$\langle 1, 1, 1 \rangle$	$\langle 1, 1, 1 \rangle$
5	TTLOFJOB	JOB COD	JOB TTL	500	$\langle 1, 1, 1 \rangle$	$\langle 1, 1, 1 \rangle$
6	SALOFJOB	JOB COD	ATHSAL	1000	$\langle 1, 2, 4 \rangle$	$\langle 1, 1, 1 \rangle$
7	MAXOFSAL	ATHSAL	ATHMAX	500	$\langle 1, 1, 1 \rangle$	$\langle 1, 1, 1 \rangle$
8	MINOFSAL	ATHSAL	ATHMIN	500	$\langle 1, 1, 1 \rangle$	$\langle 1, 1, 1 \rangle$
9	DDCOFSAL	ATHSAL	DDUCTN	2000	$\langle 2, 2, 2 \rangle$	$\langle 1, 1, 1 \rangle$
10	EMPOFORG	ORGCOD	EMPNUM	5000	$\langle 1, 100, 250 \rangle$	$\langle 1, 1, 1 \rangle$
11	JHSOFEMP	EMPNUM	JOB COD	10000	$\langle 0, 2, 10 \rangle$	$\langle 0, 25, 30 \rangle$
12	JOB OFEMP	EMPNUM	JOB COD	5000	$\langle 1, 1, 1 \rangle$	$\langle 1, 10, 30 \rangle$
13	YRSOFEMP	EMPNUM	JHSYRS	5000	$\langle 1, 1, 1 \rangle$	$\langle 50, 50, 50 \rangle$
14	NAMOFEMP	EMPNUM	EMPNUM	5000	$\langle 1, 1, 1 \rangle$	$\langle 1, 1, 1 \rangle$
15	MSTOFEMP	EMPNUM	EMPMST	5000	$\langle 1, 1, 1 \rangle$	$\langle 1, 500, 2000 \rangle$
16	BRTOFEMP	EMPNUM	BRTDAT	5000	$\langle 1, 1, 1 \rangle$	$\langle 1, 1, 1 \rangle$
17	LVLOFEMP	EMPNUM	EMPLVL	5000	$\langle 1, 1, 1 \rangle$	$\langle 1, 250, 2000 \rangle$
18	ADROFEMP	EMPNUM	EMPADR	5000	$\langle 1, 1, 1 \rangle$	$\langle 1, 1, 1 \rangle$
19	LOCOFOFR	OFR COD	OFRLOC	100	$\langle 1, 1, 1 \rangle$	$\langle 1, 2, 4 \rangle$
20	FMTFOFR	OFR COD	OFRFMT	100	$\langle 1, 1, 1 \rangle$	$\langle 10, 10, 10 \rangle$
21	DATOF COD	OFR COD	OFRDAT	100	$\langle 1, 1, 1 \rangle$	$\langle 1, 2, 4 \rangle$
22	TEACHERS	OFR COD	EMPNUM	2200	$\langle 1, 22, 30 \rangle$	$\langle 0, 1, 1 \rangle$
23	STUDENTS	OFR COD	EMPNUM	2100	$\langle 1, 21, 50 \rangle$	$\langle 0, 2, 5 \rangle$
24	TTLOFCOU	COUNUM	COUTTTL	50	$\langle 1, 1, 1 \rangle$	$\langle 1, 1, 1 \rangle$
25	DSPOFCOU	COUNUM	COUDSP	50	$\langle 1, 1, 1 \rangle$	$\langle 1, 1, 1 \rangle$
26	OFR OFCOU	COUNUM	OFR COD	100	$\langle 0, 2, 10 \rangle$	$\langle 0, 1, 1 \rangle$

Figure 2.3 An Example Set SR of 26 Data Base Relations R_j

from A to B. It is obvious that if R_j is a type zero relation then $r_j = \alpha_j = \beta_j = \beta'_j = \alpha'_j$.

Relation $R_j(A, B)$ is of type 1 if $RM21(j) = \langle 1, 1, 1 \rangle$ and $m_j = M_j \neq 1$. This type of relation over A and B is one in which each and every element of A is related to exactly M_j , B-values while at the same time each and every B-value is related to exactly one A-value. It is clear that $\bar{m}_j = M_j$ because $m_j = M_j$ and also that a relation cannot be both of type 1 and of type 0 because $M_j \neq 1$ is assumed. A type 2 relation is defined somewhat similarly to a type 1 relation as follows. Relation $R_j(A, B)$ is of type 2 if $RM12(j) = \langle 1, 1, 1 \rangle$ and $\bar{n}_j = N_j \neq 1$. Type 1 and type 2 relations are special cases of functions from B to A and from A to B, respectively, in set theoretic terminology.

Relation $R_j(A, B)$ is of type 3 if $RM21(j) = \langle 1, 1, 1 \rangle$ and $m_j \neq M_j$. The difference between a type 3 and a type 1 relation is that in a type 1 relation each A-value is related to exactly M_j B-values whereas in a type 3 relation it may be related to any number of B-values from m_j to M_j where m_j may even be zero. Again by requiring $m_j \neq M_j$ for type 3 we are preventing a type 1 (and type zero) relation to be also classified as a type 3 relation. Type 4 relations are similarly defined as follows. Relation $R_j(A, B)$ is of type 4 if $RM12(j) = \langle 1, 1, 1 \rangle$ and $n_j \neq N_j$. Again, type 3 and type 4 relations are functions from B to A and from A to B, respectively.

Relation $R_j(A, B)$ is of type 5 if $RM21(j) = \langle 0, 1, 1 \rangle$ and $m_j \neq M_j$. Relations of this type are similar to relations of type 3 except that in this case some B-values are related to no A-value in R_j because $n_j = 0$ is assumed. A-values may be related to a variable number (including zero) of B-values. Type 6 relations are similarly defined as follows. Relation $R_j(A, B)$ is of type 6 if $RM12(j) = \langle 0, 1, 1 \rangle$ and $n_j \neq N_j$. Finally, if the type of relation $R_j(A, B)$ is not zero through 6 then it is defined to be of type 7. A relation of type 7 is a many-to-many relation and either or both of m_j and n_j may be zero. In the example set of data base relations of Figure 2.3, we observe that relation R_2 is of type zero, R_9 is of type 1, R_{20} is of type 2, R_6 is of type 3, R_{12} is of type 4, R_3 is of type 5 and R_{11} is of type 7.

Figure 2.4 shows a directed graph representation of the example sets of data item types and data base relations of Figures 2.1 and 2.3. In this graph a node corresponds to a data item type in SI and is labeled with the name of that data item type. An arc is directed from node A to node B if there exists a relation $R_j \in SR$ such that $A = \text{ORG}(R_j)$ and $B = \text{DST}(R_j)$ and the arc is labeled with the name of the relation R_j .

This diagram shows the logical associations among elements of data for a typical data base. The task of data base design is to group data item types into records and relate records by data base sets (or inter-record pointers)

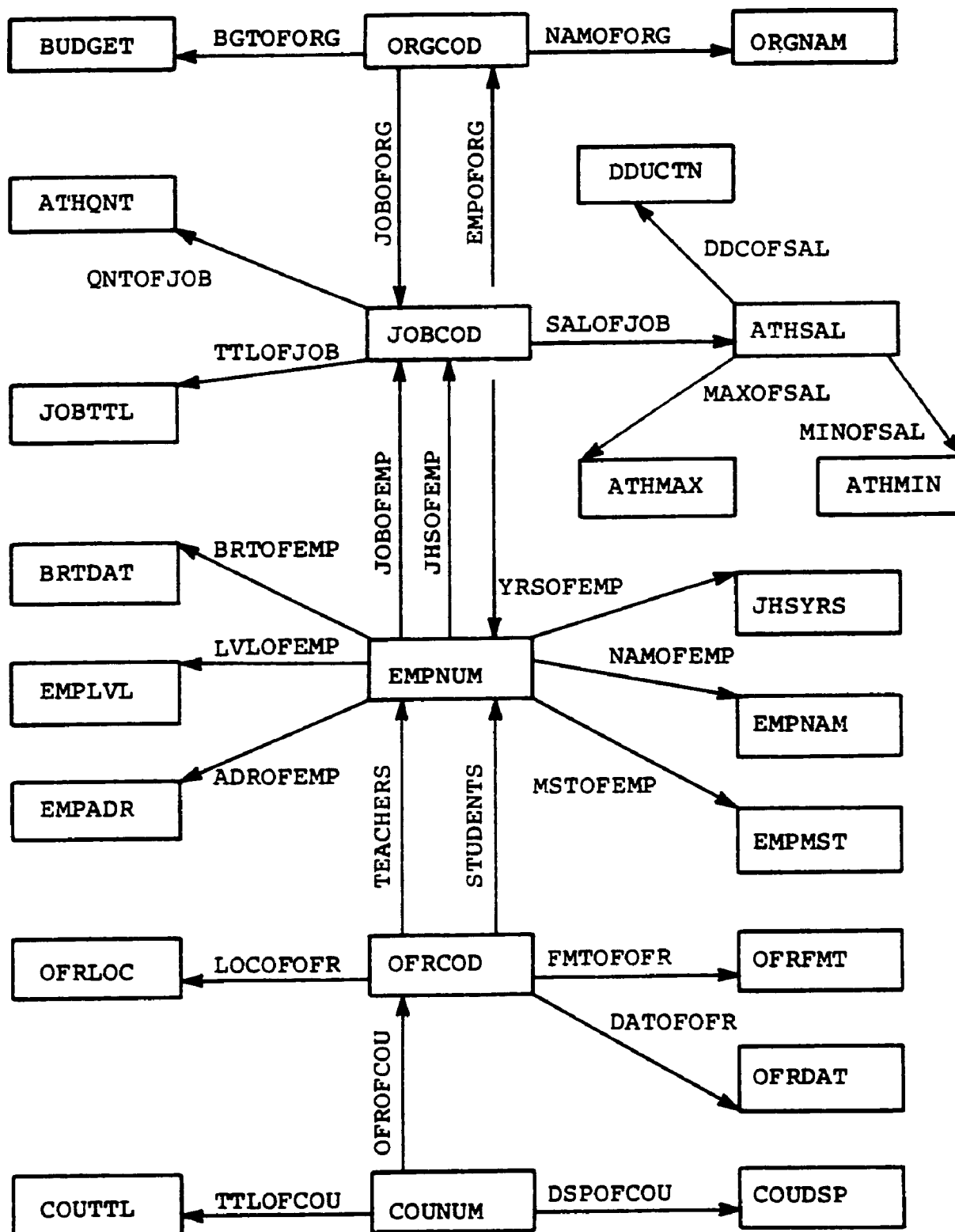


Figure 2.4 Directed Graph Representation of SI and SR

such that intra- and inter-record structures are compatible with DBTG specifications, the resulting structures are (in some predefined sense) optimal and yet the overall design can be carried out automatically and with reasonable efficiency.

The graphical representation of data items and relations is helpful in the following few definitions which we will use in connection with constraint conditions in Chapter 3. For each data item type $I_i \in SI$, $i = 1, 2, \dots, NI$ two counters $IORC(i)$ and $IDSC(i)$ represent the number of data base relations $R_j \in SR$ such that $I_i = ORG(R_j)$ and $I_i = DST(R_j)$, respectively. In the graphic representation, for each node I_i , $IORC(i)$ (Item Origin Count) is the number of arcs emanating from I_i and $IDSC(i)$ (Item Destination Count) is the number of arcs incident to I_i .

A node $I_i \in SI$ is said to be a sink (source) node if $IDSC(i) = 1$ and $IORC(i) = 0$ ($IORC(i) = 1$ and $IDSC(i) = 0$).

Note that our definitions of a sink and a source node are slightly different from the conventional ones in the sense that we require $IDSC(i)$ and $IORC(i)$ to be exactly one for sink and source nodes, respectively. The set $SINR$ of relations, which is a subset of SR is defined as follows:

$$SINR = \{R_j \in SR \mid DST(R_j) \text{ is a sink node; i.e., } IDSC(i) = 1 \text{ and } IORC(i) = 0 \text{ where } I_i = DST(R_j)\}.$$

Similarly, the set $SONR$ of relations, again a subset of SR

and mutually exclusive with SINR, is defined as follows:

$$\text{SONR} = \{R_j \in \text{SR} \mid \text{ORG}(R_j) \text{ is a source node; i.e.,}$$

$$\text{IORC}(i) = 1 \text{ and } \text{IDSC}(i) = 0 \text{ where } I_i = \text{ORG}(R_j) \}.$$

For example, in Figure 2.4, ORGNAM is a sink node and ORGCOD is not a source node.

Given a set of data item types SI and a set of relations SR whose elements are relations over pairs of data item types in SI, the assumption that each data item value is related to at least one other data item value through a relation in SR implies that each data item type in SI participates (as origin or as destination data item type) in at least one relation in SR. Alternatively we can state that $\text{IDSC}(i)$ and $\text{IORC}(i)$ cannot be simultaneously equal to zero for any $i \in \{1, 2, \dots, \text{NI}\}$. In graph representation this means that there are no isolated single nodes in the graph. The destination data item type of each relation R in SINR participates in exactly one relation in SR, namely R, and similarly the origin data item type of each relation R' in SONR participates in exactly one relation in SR, namely R'.

2.3 User Activity

In the preceding two sections, those components of the user requirements model dealing with the static view of data, namely data items and data base relations, were discussed. In this section we define a set of 12 data manipulation operations in terms of which sequences of data

retrieval, storage and update can be expressed. These operations are defined over the data items and data base relations with no assumption regarding record and set structures, and in that sense they are different from DBTG DML commands. Named sequences of data manipulation operations are called run units. We define the 12 data manipulation operations in terms of a typical relation $R_j(A, B)$ over data item types A and B, with A as the origin and B as the destination data item type, respectively. These operations will be concerned with the retrieval, update or storage of some value (or collection of values) of one data item type related, through a specified relation, to a "given value" of the other data item type where actual values will be provided at execution time. For example, an operation that requires the retrieval of the i -th B-value related to a $a \in A$ in relation $R_j(A, B)$ will specify i and a at execution time.

With each operation a number f , called the operation's frequency, is associated. The frequency of an operation is the number of times that the operation is expected to be used in some unit of time, say a week. An operation is defined as a 5-tuple such as $O = \langle op, B, A, R_j, f \rangle$ where $op \in SOP$ is the operation code (SOP is defined below), R_j is a relation in SR, B is called the operand of O, A is called the qualifier of O and f is the frequency of the operation O. The relation R_j must be defined over A and B either with A as origin and B as destination data item type

$(R_j(A, B))$ or vice versa $(R_j(B, A))$. SOP is the set of operation codes that we consider in our work and it is defined as follows:

SOP = {FIND FIRST, FIND LAST, FIND ITH, FIND NEXT,
FIND PREV., FIND ALL, FIND KEY, STORE FIRST,
STORE LAST, STORE KEY, MODIFY, ERASE}

Given the above set SOP of 12 operation codes we now define 12 operations using relation $R_j(A, B)$.

1) $\langle \text{FIND FIRST}, B, A, R_j, f \rangle :$

Find the first $b \in B$ related to a given $a \in A$ in $R_j(A, B)$, with frequency f . This is one of our retrieval operations requiring the retrieval of the first element of the set $\text{RIM}_j(a)$ where $a \in A$ will be provided at execution time. For example, an operation such as $\langle \text{FIND FIRST}, \text{OFRCOD}, \text{COUNUM}, \text{OFROFCOU}, 20 \rangle$ requires the retrieval of the offering code of the first offering of a course whose course number is given and the two values of OFRCOD and COUNUM are related through OFROFCOU relation. The operation is expected to be executed 20 times over a certain time period (e.g. a week).

2) $\langle \text{FIND LAST}, B, A, R_j, f \rangle :$

Find the last $b \in B$ related to a given $a \in A$ in $R_j(A, B)$, with frequency f . This operation is also a retrieval operation and it requires the retrieval of the last element of $\text{RIM}_j(a)$ for a given a , specified at execution time.

3) $\langle \text{FIND ITH}, B, A, R_j, f \rangle :$

Find the i -th $b \in B$ related to a given $a \in A$ in $R_j(A, B)$, with frequency f . In this operation the i -th B -value related to a specific A -value, say a , is to be retrieved. The B -value is actually the i -th element of $\text{RIM}_j(a)$. The values of i and a will be provided at execution time. An example of this kind of operation would be $\langle \text{FIND ITH}, \text{EMPNUM}, \text{ORGCOD}, \text{EMPOFORG}, 400 \rangle$. This operation requires the retrieval of the i -th employee number (EMPNUM) related to a given organization code (ORGCOD) in the relation EMPOFORG, a total of 400 times.

4) $\langle \text{FIND NEXT}, B, A, R_j, f \rangle :$

Find the next $b \in B$ related to a given $a \in A$ in $R_j(A, B)$, with frequency f . This operation assumes that a "current" B -value, say $b' \in B$, has been established which is related to $a \in A$ in $R_j(A, B)$. The operation then requires the retrieval of the next, with respect to the current, B -value related to that given A -value in $R_j(A, B)$, and it is expected to be used a total of f times.

5) $\langle \text{FIND PREV.}, B, A, R_j, f \rangle :$

Find previous $b \in B$ related to a given $a \in A$ in $R_j(A, B)$, with frequency f . Similar to FIND NEXT operation, this operation assumes that a current B -value related in R_j to a given A -value, say a , has been established and now the prior B -value related in

R_j to a is to be retrieved.

- 6) $\langle \text{FIND ALL}, B, A, R_j, f \rangle :$

Find all B -values related to a given $a \in A$ in $R_j(A, B)$, with frequency f . Unlike the previous 5 operations where a single B -value would be retrieved, in this operation a collection of B -values, namely the set $\text{RIM}_j(a)$ of all $b \in B$ related to $a \in A$ in $R_j(A, B)$, should be retrieved. The retrieved values may or may not be delivered to the run units (put in the user working area) at the same time.

- 7) $\langle \text{FIND KEY}, B, A, R_j, f \rangle :$

Find a given $b \in B$ related to a given $a \in A$ in $R_j(A, B)$, with frequency f . It is clear from its definition that in the FIND KEY operation both A - and B -values, namely a and b respectively, are given and the purpose is to verify whether $\langle a, b \rangle$ is in $R_j(A, B)$. This operation is the last retrieval operation and the following five operations are update operations.

- 8) $\langle \text{STORE FIRST}, B, A, R_j, f \rangle :$

Store a given B -value, b , as the first B -value related in R_j to a given A -value, a . As a result of this operation the B -value, to be specified at execution time, will become the first element of the set $\text{RIM}_j(a)$. This operation establishes the membership of $\langle a, b \rangle$ in R_j in the above manner which may or may not necessitate the storage of individual a and b occurrences.

- 9) $\langle \text{STORE LAST}, B, A, R_j, f \rangle :$

Store a given B-value, b , as the last B-value related in R_j to a given A-value, a , with frequency f . Except for the fact that STORE LAST operation establishes b as the last member of $\text{RIM}_j(a)$, this operation is similar to STORE FIRST operation. In both of these operations, however, the assumption is that the insertion of b in $\text{RIM}_j(a)$ will not impair its ordering.

- 10) $\langle \text{STORE KEY}, B, A, R_j, f \rangle :$

Store a given B-value, b , in the usual order of B-values related to a given A-value, a , with frequency f . This operation establishes the membership of $\langle a, b \rangle$ in R_j by inserting b in $\text{RIM}_j(a)$ such that its order is not impaired.

- 11) $\langle \text{MODIFY}, B, A, R_j, f \rangle :$

Modify a given $b \in B$ related to a given $a \in A$ in R_j to another $b' \in B$ with frequency f . Similar to previous operations the actual values of a , b and b' will be provided at execution time. Note that as the result of this operation the former B-value, b , may be deleted from the data base altogether and that the new B-value, b' , is placed in a position in $\text{RIM}_j(a)$ where its ordering is not impaired.

- 12) $\langle \text{ERASE}, B, A, R_j, f \rangle :$

Erase (delete) a given B-value, b , related to a given $a \in A$ in R_j , with frequency f . This operation requires the removal of the pair $\langle a, b \rangle$ from R_j which in turn

may or may not necessitate the deletion of either or both a and b occurrences from the data base.

As mentioned earlier in this section, named sequences of data manipulation operations are called run units and are denoted by RU_k , $k = 1, 2, \dots, NU$ where NU is the number of run units defined for a specific application. The set of all run units defined for an application is denoted by SU : $SU = \{RU_1, RU_2, \dots, RU_{NU}\}$. In this set each RU_k , $k = 1, \dots, NU$ is a named sequence of NO_k operations O_{ℓ}^k , $\ell = 1, 2, \dots, NO_k$. Figure 2.5 illustrates an example set of $NU = 4$ run units $RU_1, \dots, RU_4$ called PAYROLL, PERSONNEL, ADMINISTRATION and EDUCATION, respectively. Run units RU_1, RU_2, RU_3 and RU_4 of Figure 2.5 have the following number of operations: $NO_1 = 12$, $NO_2 = 28$, $NO_3 = 15$ and $NO_4 = 26$.

Note that each of the 12 operations defined above on relation $R_j(A, B)$ can also be specified in the reverse order of data item types A and B with similar definitions, i.e., A may be specified as the operand and B as the qualifier data item type of the operation. For example, an operation such as $\langle \text{FIND ALL}, A, B, R_j, f \rangle$ means: find all A-values related to a given $b \in B$ in $R_j(A, B)$ with frequency f . The retrieved set of A-values in this case is $LIM_j(b)$ instead of $RIM_j(a)$ in the case of operation number 6 above.

Having defined data manipulations operations and run

<u>OPERATION</u>	<u>OPERAND</u>	<u>QUALIFIER</u>	<u>RELATION</u>	<u>FREQUENCY</u>
1 PAYROLL				
FIND FIRST	EMPNAM	EMPNUM	NAMOFEMP	5000
FIND FIRST	EMPADR	EMPNUM	ADROFEMP	5000
FIND FIRST	JOB COD	EMPNUM	JOBOFEMP	5000
MODIFY	ATHSAL	JOB COD	SALOFJOB	1000
FIND NEXT	ATHSAL	JOB COD	SALOFJOB	2000
FIND FIRST	ATHSAL	JOB COD	SALOFJOB	1200
FIND FIRST	JOB COD	ATHSAL	SALOFJOB	20
FIND ALL	ATHSAL	JOB COD	SALOFJOB	1000
FIND FIRST	ATHMIN	ATHSAL	MINOFSAL	100
FIND FIRST	ATHMAX	ATHSAL	MAXOFSAL	50
MODIFY	DDCTNS	ATHSAL	DDCOFSAL	20
FIND FIRST	DDCTNS	ATHSAL	DDCOFSAL	100
2 PERSONNEL				
FIND ALL	EMPNUM	ORGCOD	EMPOFORG	40
FIND NEXT	EMPNUM	ORGCOD	EMPOFORG	800
STORE KEY	EMPNUM	ORGCOD	EMPOFORG	20
FIND ITH	EMPNUM	ORGCOD	EMPOFORG	400
MODIFY	EMPNUM	ORGCOD	EMPOFORG	500
FIND FIRST	EMPADR	EMPNUM	ADROFEMP	1000
STORE FIRST	EMPADR	EMPNUM	ADROFEMP	20
FIND FIRST	EMPNAM	EMPNUM	NAMOFEMP	1000
STORE FIRST	EMPNAM	EMPNUM	NAMOFEMP	20
FIND FIRST	EMPLVL	EMPNUM	LVLOFEMP	1000
STORE FIRST	EMPLVL	EMPNUM	LVLOFEMP	20
STORE FIRST	BRTDAT	EMPNUM	BRTOFEMP	20
FIND ALL	JOB COD	EMPNUM	JHSOFEMP	1000
FIND NEXT	JOB COD	EMPNUM	JHSOFEMP	3000
MODIFY	JHSYRS	EMPNUM	YRSOFEMP	2
FIND KEY	JOB COD	EMPNUM	JHSOFEMP	700
MODIFY	JOB COD	EMPNUM	JHSOFEMP	500
FIND NEXT	EMPNUM	JOB COD	JHSOFEMP	800
MODIFY	EMPNUM	ORGCOD	EMPOFORG	10
FIND FIRST	JOB COD	EMPNUM	JOBOFEMP	1000
FIND ALL	JOB COD	EMPNUM	JOBOFEMP	2000
FIND ALL	EMPNUM	JOB COD	JOBOFEMP	500
MODIFY	JOB COD	EMPNUM	JOBOFEMP	2000
FIND NEXT	JOB COD	EMPNUM	JOBOFEMP	3000
STORE FIRST	EMPMST	EMPNUM	MSTOFEMP	20
MODIFY	EMPMST	EMPNUM	MSTOFEMP	10
FIND FIRST	EMPMST	EMPNUM	MSTOFEMP	1000
FIND ALL	EMPNUM	BRTDAT	BRTOFEMP	2

Figure 2.5 An Example Set SU of 4 Run Units

<u>OPERATION</u>	<u>OPERAND</u>	<u>QUALIFIER</u>	<u>RELATION</u>	<u>FREQUENCY</u>
3 ADMINISTRATION				
FIND FIRST	JOBTTL	JOBCOD	TTLOFJOB	20
MODIFY	JOBTTL	JOBCOD	TTLOFJOB	10
FIND ALL	EMPNUM	ORGCOD	EMPOFORG	50
FIND FIRST	BUDGET	ORGCOD	BGTOFORG	50
FIND FIRST	BUDGET	ORGCOD	BGTOFORG	50
FIND FIRST	ORGNAM	ORGCOD	NAMOFORG	50
MODIFY	JOBCOD	ORGCOD	JOB OFORG	35
FIND ALL	JOBCOD	ORGCOD	JOB OFORG	40
FIND ALL	ORGCOD	JOBCOD	JOB OFORG	400
FIND LAST	JOBCOD	ORGCOD	JOB OFORG	30
FIND NEXT	JOBCOD	ORGCOD	JOB OFORG	40
MODIFY	ATHQNT	JOBCOD	QNT OFJOB	90
MODIFY	BUDGET	ORGCOD	BGTOFORG	20
MODIFY	ATHMIN	ATHSAL	MIN OFSAL	45
MODIFY	ATHMAX	ATHSAL	MAX OFSAL	50

4 EDUCATION

FIND ALL	OFRCOD	EMPNUM	TEACHERS	100
FIND NEXT	OFRCOD	EMPNUM	TEACHERS	1000
FIND PREV.	OFRCOD	EMPNUM	TEACHERS	1000
FIND ALL	EMPNUM	OFRCOD	TEACHERS	1000
FIND NEXT	EMPNUM	OFRCOD	TEACHERS	1000
FIND PREV.	EMPNUM	OFRCOD	TEACHERS	1000
STORE LAST	OFRCOD	EMPNUM	TEACHERS	10
ERASE	OFRCOD	EMPNUM	TEACHERS	5
MODIFY	EMPNUM	OFRCOD	TEACHERS	20
STORE KEY	EMPNUM	OFRCOD	STUDENTS	15
FIND KEY	EMPNUM	OFRCOD	STUDENTS	100
FIND KEY	EMPNUM	OFRCOD	TEACHERS	40
FIND FIRST	EMPNUM	OFRCOD	STUDENTS	1000
FIND PREV.	EMPNUM	OFRCOD	STUDENTS	200
FIND NEXT	EMPNUM	OFRCOD	STUDENTS	200
FIND ALL	OFRCOD	EMPNUM	STUDENTS	1000
FIND FIRST	COUNUM	OFRCOD	OFROFCOU	200
FIND LAST	OFRCOD	COUNUM	OFROFCOU	20
MODIFY	OFRCOD	COUNUM	OFROFCOU	800
FIND PREV.	OFRCOD	COUNUM	OFROFCOU	70
FIND FIRST	OFRFMT	OFRCOD	FMT OFOFR	200
FIND FIRST	OFRDAT	OFRCOD	DAT OFOFR	200
FIND FIRST	OFRLOC	OFRCOD	LOC OFOFR	200
FIND FIRST	OFRCOD	OFRLOC	LOC OFOFR	200
FIND FIRST	COUTTL	COUNUM	TTLOFCOU	100
FIND FIRST	COUDSP	COUNUM	DSPOFCOU	100

Figure 2.5 (Cont'd.)

units we will now define an additional variable for each relation $R_j \in SR$, which is the count of run units that use R_j in at least one of their operations. The set of variables $RUUR_j$, $j = 1, 2, \dots, NR$, is then defined as follows:

$$RUUR_j = |SUR_j|$$

where $SUR_j = \{RU_k \in SU \mid \text{there exists } 1 \leq \ell \leq NO_k \text{ such that } R_j \text{ is used in } O_\ell^k\}$.

These variables will be necessary in Chapter 3 where we will define storage costs.

Furthermore, for each run unit $RU_k \in SU$, $k = 1, \dots, NU$, we define the following two sets.

$$PSI_k = \{I \in SI \mid \text{for some } 1 \leq \ell \leq NO_k, I \text{ is either the qualifier or the operand data item type of } O_\ell^k\}$$

$$PSR_k = \{R \in SR \mid \text{for some } 1 \leq \ell \leq NO_k, R \text{ is the relation used in } O_\ell^k\}.$$

We recall that O_ℓ^k is by definition the ℓ -th operation of k -th run unit. For each RU_k , $k = 1, \dots, NU$, PSI_k is the set of data items that run unit RU_k uses and PSR_k is the set of relations used by RU_k . These two collections of sets will be used in connection with the constraint conditions defined in Chapter 3. PSI_k , $k = 1, 2, \dots, NU$ are not necessarily mutually exclusive sets. The same proposition holds for PSR_k , $k = 1, 2, \dots, NU$. Furthermore $\bigcup_{k=1}^{NU} PSR_k$ is not necessarily equal to the set SR . In the case where the union $\bigcup_{k=1}^{NU} PSR_k$ is a proper subset of SR , the time cost of relations in SR that are not members of the union will be

zero.

In the first run unit RU_1 shown in Figure 2.5, for example, $PSR_1 = \{R_6, R_7, R_8, R_9, R_{12}, R_{14}, R_{18}\}$ and $PSI_1 = \{I_4, I_6, I_7, I_8, I_9, I_{11}, I_{12}, I_{16}\}$.

The sequence of operations that comprise a run unit is intended only to model the systems view of data retrieval and update expected to occur over the data base by the run unit. In that sense, those operations do not model the exact DBTG data manipulation commands nor do they correspond to such commands in a one-to-one relationship. Furthermore, different operations of a run unit (or different executions of one such operation) may not necessarily occur in the order of their appearance in the run unit. The operations included in run units in SU will be used to compute time costs using time cost functions of Chapter 3. The order of operations in run units is immaterial in the resulting time costs. Furthermore, operations with zero frequencies do not contribute to time costs. Such operations may be used however to expand PSR_k and PSI_k sets if necessary.

2.4 Storage Space Parameters

In this section we define the parameters that model the storage requirements of an application. This is the last component of the user requirements model. Values for all of the storage space parameters have to be specified by the data base designer who is using this methodology. The storage space referred to here consists of space needed

to store data characters of the data base, pointer and counter variables, and the space needed to store certain privacy locks. It does not include some physical space requirements such as access method tables that are not modelled here.

A complete description of the storage space requirements for an application consists of specifying values for the following six parameters.

- 1) TMSB : Maximum Storage Bound ,
- 2) PGSZ : Page Size ,
- 3) MSPG : Main Storage Pages ,
- 4) PTRS : Pointer Size ,
- 5) CNTRS: Counter Size ,
- 6) PLOKS: Privacy Lock Size ,

The storage space can be viewed as a sequence of storage locations each capable of holding one character (one byte) of information. Associated with each storage location is a unique string of fixed length called the address of that storage location. Address strings are ordered with the same ordering defined in Section 2.1 for strings of fixed length. The maximum size of the storage space is given by TMSB. This is the absolute maximum storage limit that the designer specifies at the beginning of the optimization process. The optimization algorithm of Chapter 4 finds a spectrum of data base designs each of which is optimal over a range of storage space limits

below TMSB. TMSB is measured in units of one character (bytes).

The storage space in which the data base will be stored resides on a secondary storage medium and is divided into sub-divisions of equal size given by PGSZ, again measured in bytes. Accesses to secondary storage are made in units of one page and at most a given number MSPG of data pages can be in main storage at a given time. PTRS is the length of an address string (or a data base key in DBTG terminology) measured in units of one byte. An address occurrence is a contiguous sequence of PTRS storage locations that contain either an address string or a null value. CNTRS is the size of a counter data item, measured in units of one byte, used to hold the length of a variably dimensioned vector or repeating group. Finally, PLOKS is the size, in bytes, of a privacy lock.

A typical storage space requirements specification is the following.

TMSB	= 1,600,000	bytes
PGSZ	= 2000	bytes
MSPG	= 2	pages
PTRS	= 4	bytes
CNTRS	= 6	bytes
PLOKS	= 20	bytes

In this chapter we have presented the four components of the user requirements model, namely a set of data item types SI, a set of data base relations SR defined over

elements of SI, a set of run units SU consisting of operations that use elements of SR and a set of six storage space parameters. In Chapter 3 we define a number of implementation alternatives for a data base relation. The information provided in SI, SR, SU and storage space parameters will enable us to determine acceptability, storage cost and time cost of relation implementations. In turn, the information provided by costing functions of Chapter 3 will be used to set up and solve an optimization problem, in Chapter 4, which will determine the optimal data base design for the application described by SI, SR, SU and storage space parameters. The user requirements model may, then, be viewed as the input model to our data base design methodology. Also in this chapter we have given an example of a completely specified application.

CHAPTER III

IMPLEMENTATION ALTERNATIVES AND COSTS

The implementation of a data base relation $R(A, B)$ determines the way the relation between data item types A and B will actually be realized in the data base. In that sense the choice of exactly one implementation for each and every data base relation in SR (set of all relations for a specific application) will determine the overall data structure of the data base, upto the relative ordering of data items in a record. There may be more than one possible implementation for a relation and it is up to the optimization algorithm to decide which alternative should be selected for each relation so that the overall cost is minimum.

In Section 3.1 we will define a set of 24 implementations of a data base relation. It is our intention to include a sufficient number of implementation alternatives in the model so that as much use as possible can be made of the data structuring capability of the network (DBTG) data model. In Section 3.2 we will present a set of constraint conditions for each relation implementation alternative. Basically these constraint conditions are a set of rules which guarantee that data structures which are illegal in a DBTG data model would not result from the optimization process. These rules will also guarantee that the security requirements of run units will always be satisfied. We will present the formulas for computing storage costs

of relation implementations in Section 3.3. The page fault probability model used in this work is discussed in Section 3.4 and it is used as a starting point for the derivation of time costs in Section 3.4.

Throughout this chapter, we will use a typical relation $R_j \in SR$ characterized by the following variables in our definitions. Subscript j will be suppressed where no ambiguity can arise.

$$R_j = R(RNAME, A, B, r_j, \langle m_j, \bar{m}_j, M_j \rangle, \langle n_j, \bar{n}_j, N_j \rangle)$$

where

$j \in \{1, \dots, NR\}$ is the ordinal number of $R_j \in SR$ and
 $NR = |SR|$

$RNAME = RNAME(j)$ is the name of the relation $R_j \in SR$

$A = ORG(R_j)$ is the origin data item type of R_j

$B = DST(R_j)$ is the destination data item type of
 R_j

r_j is the number of ordered pairs in R_j
and is called the cardinality of the
relation R_j

$\langle m_j, \bar{m}_j, M_j \rangle$ is an ordered 3-tuple representing
the minimum, average and maximum
number of B-values related to one
A-value in R_j , respectively

$\langle n_j, \bar{n}_j, N_j \rangle$ is an ordered 3-tuple representing
the minimum, average and maximum
number of A-values related to one
B-value in R_j , respectively.

The variables associated with data item types A and B that will be referred to in this chapter are the following.

$\alpha'_j = A $	is the total number of A-values called the cardinality of data item type A, with β'_j similarly defined for B
α_j	is the number of A-values related to at least one B-value in R_j , with β_j defined similarly
α''_j	size of data item A, with β''_j defined similarly for B.

3.1 Implementation Alternatives

The implementation alternatives discussed in this section fall into two major categories. The first category contains implementations that group data items together to form records. The second category contains implementations that associate records by means of data base sets or explicit pointer data items. The motivation for this type of categorization is the following. In DBTG data structures, relationships may be implicit or explicit. An example of an implicit relationship is the relationship between the JOBCOD and EMPNUM data items when these two data items are defined as parts of the same record. An explicit relationship, on the other hand, is one that associates related values of the two data item types (residing in two different records) using data base sets.

Figure 3.1 shows a listing of the 24 implementations included in the model. In the following we will give the definitions of these implementation alternatives. The subscript j will be omitted from references to relation R_j and variables that characterize R_j throughout this section.

3.1.1 Fixed Duplications

Implementation 1 for $R(A, B)$ is called the fixed duplication of B under A . It consists of a set of α' ordered pairs such as

$$\{ \langle a_1, fd_1(B) \rangle, \langle a_2, fd_2(B) \rangle, \dots, \langle a_{\alpha'}, fd_{\alpha'}(B) \rangle \}.$$

In the above set a_i , $i = 1, \dots, \alpha'$ are unique A -values and $fd_i(B)$ is a vector of M elements (M is the maximum number of B -values that may be related to one A -value in $R(A, B)$). The vector $fd_i(B)$ contains one copy of each B -value that is related to a_i in $R(A, B)$, for $i = 1, \dots, \alpha'$. There are α ordered pairs $\langle a_i, fd_i(B) \rangle$ for which $fd_i(B)$ contains at least one non-null B -value and for the rest of the ordered pairs in the set $fd_i(B)$ consists of M null B -values. On the average for each pair $\langle a_i, fd_i(B) \rangle$ in the set r/α' non-null values (copies of B -values) exist in $fd_i(B)$. For all $i = 1, \dots, \alpha'$ the vector $fd_i(B)$ is stored in the same record in which a_i is stored (Figure 3.2).

Implementation number 2 for $R(A, B)$ is called the fixed duplication of A under B . The definition of this implementation is similar to that of implementation 1, and it consists of a set of ordered pairs such as

- (1) Fixed Duplication of B under A
- (2) Fixed Duplication of A under B
- (3) Variable Duplication of B under A
- (4) Variable Duplication of A under B
- (5) Fixed Aggregation of B under A
- (6) Fixed Aggregation of A under B
- (7) Variable Aggregation of B under A
- (8) Variable Aggregation of A under B
- (9) Chain with Next Pointers Association of B under A
- (10) Chain with Next Pointers Association of A under B
- (11) Chain with Next and Owner Pointers Association of B
under A
- (12) Chain with Next and Owner Pointers Association of A
under B
- (13) Chain with Next and Prior Pointers Association of B
under A
- (14) Chain with Next and Prior Pointers Association of A
under B
- (15) Chain with Next, Prior and Owner Pointers Association
of B under A
- (16) Chain with Next, Prior and Owner Pointers Association
of A under B
- (17) Pointer Array Association of B under A
- (18) Pointer Array Association of A under B
- (19) Pointer Array with Owner Pointers Association of B
under A

Figure 3.1 A Listing of the 24 Implementations.

- (20) Pointer Array with Owner Pointers Association of A under B
- (21) Dummy Record association of A and B
- (22) Single Linkage of B under A
- (23) Single Linkage of A under B
- (24) Double Linkage of A and B

Figure 3.1 (Cont'd).

$$\{ \langle b_1, fd_1(A) \rangle, \langle b_2, fd_2(A) \rangle, \dots, \langle b_{\beta'}, fd_{\beta'}(A) \rangle \}.$$

In the above set $b_1, \dots, b_{\beta'}$ are unique B-values and $fd_i(A)$, $i = 1, \dots, \beta'$ is a vector of size N (maximum number of A-values related to one B-value in R). Each $fd_i(A)$ vector contains an average of r/β' non-null values, namely copies of A-values, that are related to b_i in R. For all $i = 1, \dots, \beta'$ the vector $fd_i(A)$ is stored in the same record in which b_i is stored (Figure 3.3).

3.1.2 Variable Duplications

The next two implementations (numbered 3 and 4) use variable length vectors of duplicate values and they are defined below.

Implementation 3 for $R(A, B)$ is called the variable duplication of B under A and it consists of the following set of ordered 3-tuples.

$$\{ \langle a_1, c_1, vd_1(B) \rangle, \langle a_2, c_2, vd_2(B) \rangle, \dots, \langle a_{\alpha'}, c_{\alpha'}, vd_{\alpha'}(B) \rangle \}.$$

In this set $a_1, \dots, a_{\alpha'}$ are unique A-values, $vd_i(B)$ is a variable-length vector of B-values and c_i is a counter data item occurrence of size CNTRS, for $i = 1, \dots, \alpha'$.

For each $i = 1, \dots, \alpha'$, the vector $vd_i(B)$ and the counter c_i are stored in the same record in which a_i is stored; $vd_i(B)$ contains one copy of each B-value related to a_i in R and c_i contains the number of these B-values. The average length of $vd_i(B)$ vectors is r/α' and its range is from zero to M. These vectors do not contain any null

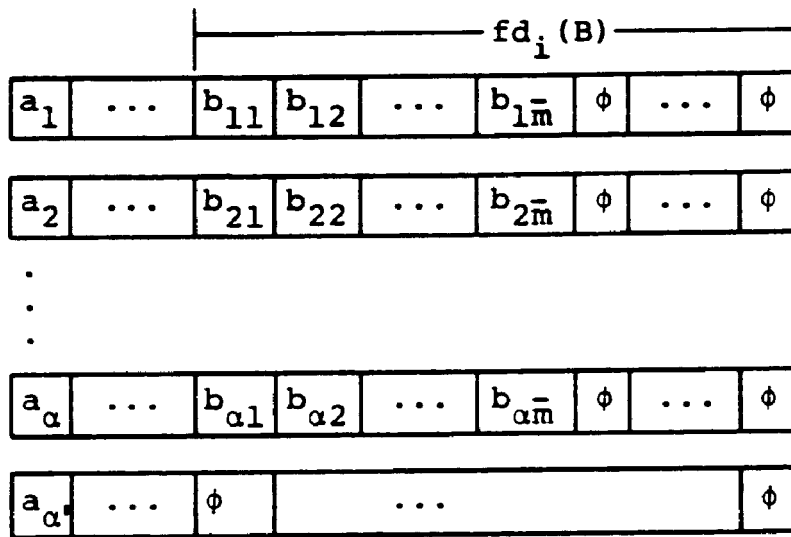


Figure 3.2 Fixed Duplication of B under A

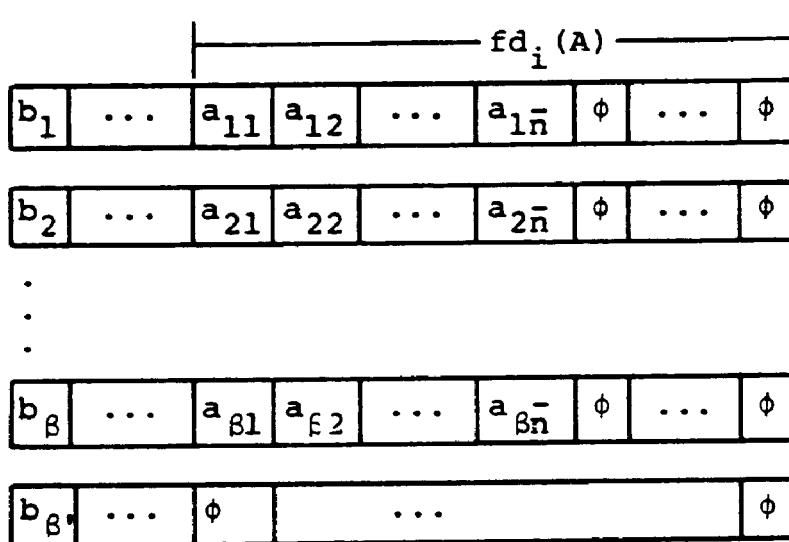


Figure 3.3 Fixed Duplication of A under B

values (Figure 3.4).

Implementation 4 for $R(A, B)$ (the converse of implementation 3) is called the variable duplication of A under B , and it consists of the following set of ordered 3-tuples.

$$\{ \langle b_1, c_1, vd_1(A) \rangle, \langle b_2, c_2, vd_2(A) \rangle, \dots, \langle b_{\beta}, c_{\beta}, vd_{\beta}(A) \rangle \}.$$

In this set $b_1, b_2, \dots, b_{\beta}$, are unique B -values, $vd_i(A)$ is a variable-length vector of duplicate A -values related to b_i in R and c_i is a counter data item occurrence (of size CNTRS) that contains the number of A -values related to b_i (length of $vd_i(A)$), $i = 1, \dots, \beta'$. For all $i = 1, \dots, \beta'$ the vector $vd_i(A)$ and the counter c_i are stored in the same record in which b_i is stored (Figure 3.5).

3.1.3 Fixed Aggregations

Implementation 5 for $R(A, B)$ is called the fixed aggregation of B under A and it consists of a set of ordered pairs such as:

$$\{ \langle a_1, fa_1(B) \rangle, \langle a_2, fa_2(B) \rangle, \dots, \langle a_{\alpha}, fa_{\alpha}(B) \rangle \}.$$

In this set $a_1, a_2, \dots, a_{\alpha}$, are unique A -values, $fa_i(B)$ is a vector of fixed length M (maximum number of B -values related to one A -value in R) which contains B -values related to a_i in R and $fa_i(B)$ is stored in the same record in which a_i is stored. The B -values in vectors of this implementation are not duplicate copies but rather they can be used to implement other relations defined over B . In particular if another data item type D is aggregated

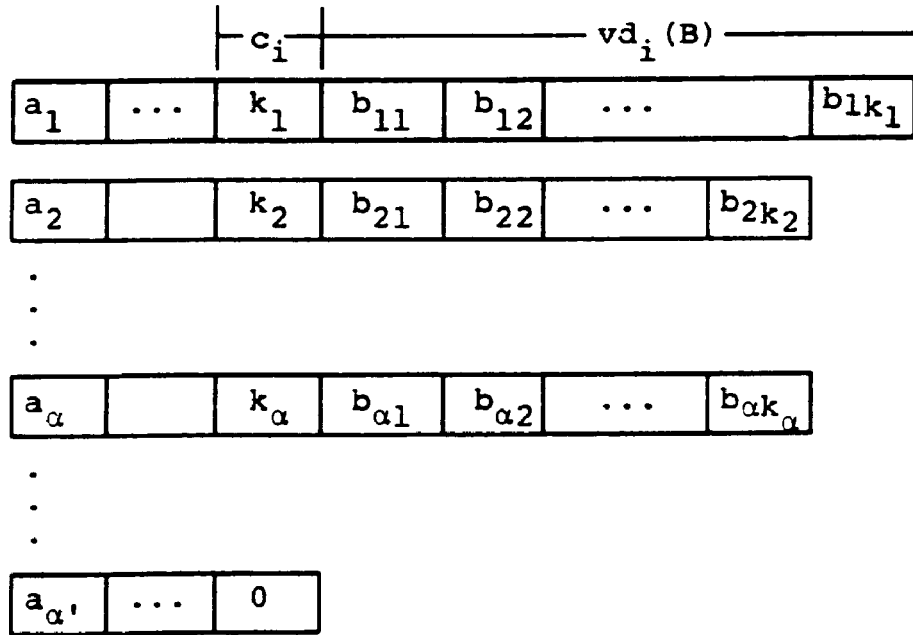


Figure 3.4 Variable Duplication of B under A

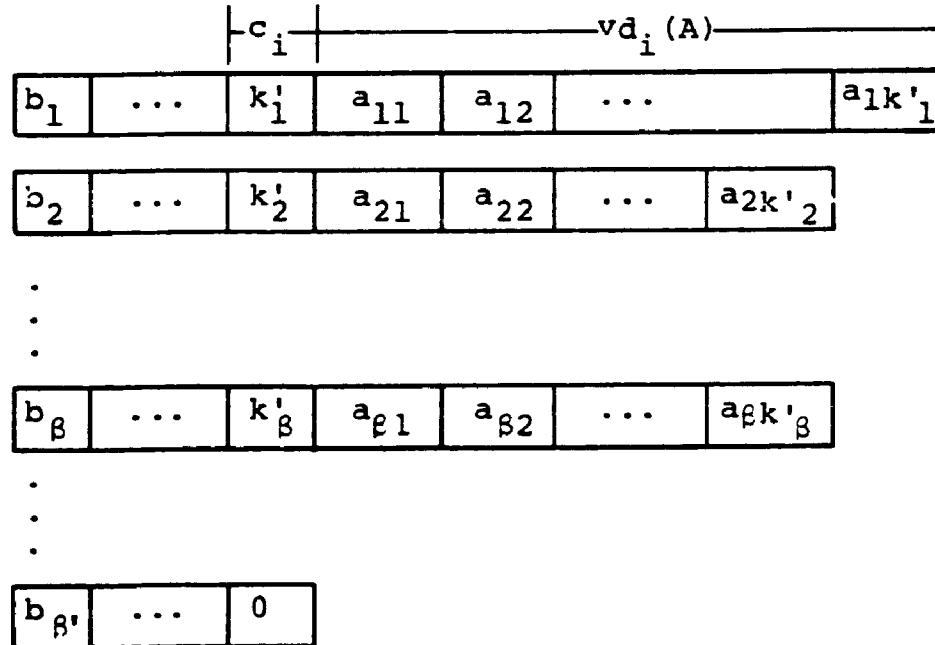


Figure 3.5 Variable Duplication of A under B

under B (to implement a relation such as $R'(B, D)$ for example) the vectors of D-values must be stored with each B-value in $fa_i(B)$ and since data items D and B are not required to be of the same size, this can generate sequences of repeating groups. There are, of course, limitations to the use of this implementation. We will discuss these limitations when we present the constraint conditions in Section 3.2. Figure 3.6 shows the fixed aggregation of B under A and Figure 3.7 shows the combination of implementation 5 for $R(A, B)$ and $R'(B, D)$.

Implementation 6 for $R(A, B)$ is called the fixed aggregation of A under B. This implementation is the converse of implementation 5 and it consists of a set of ordered pairs such as:

$$\{ \langle b_1, fa_1(A) \rangle, \langle b_2, fa_2(A) \rangle, \dots, \langle b_{\beta}, fa_{\beta}(A) \rangle \}.$$

Again in this set $b_1, \dots, b_{\beta}$, are unique B-values, $fa_i(A)$ is a vector of A-values related to b_i in R and it is stored in the same record in which b_i is stored, $i = 1, \dots, \beta'$. The fixed size of $fa_i(A)$ vectors is N, which is the maximum number of A-values related to one B-value in R. A-values in the vectors of this implementation are shared occurrences in the sense that they must be used in the implementations of all other relations over A in which A-values are not duplicated.

3.1.4 Variable Aggregations

The next two implementations (numbered 7 and 8) are

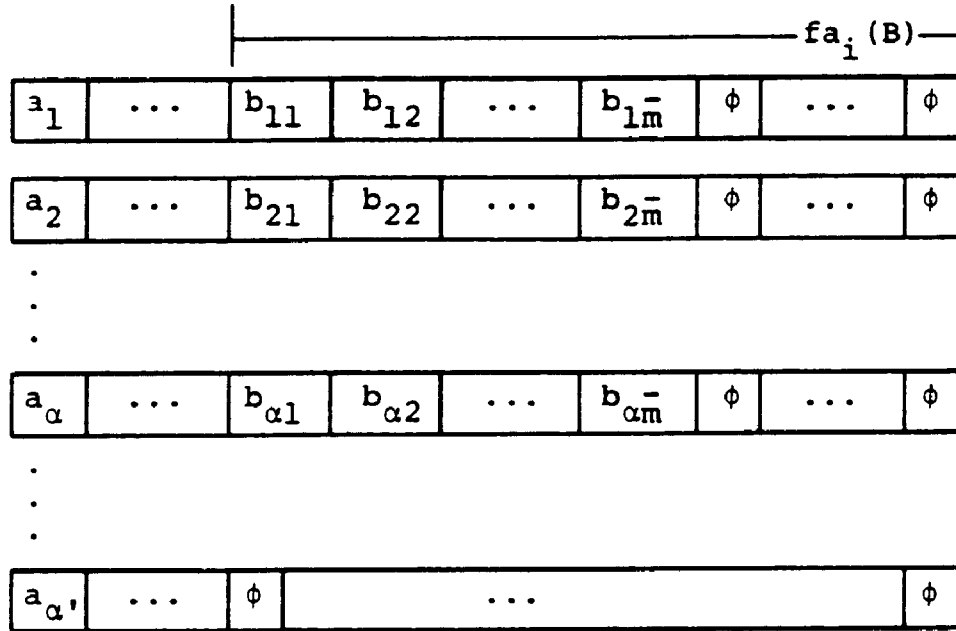


Figure 3.6 Fixed Aggregation of B under A

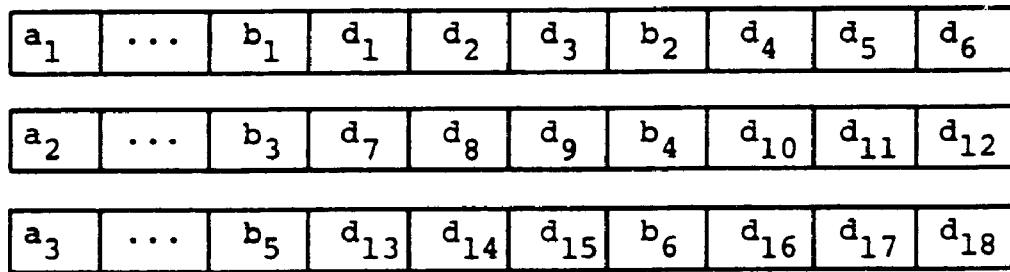
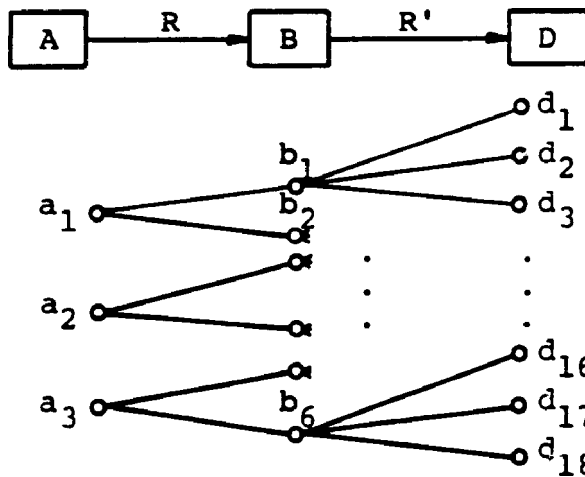


Figure 3.7 A combination of two fixed aggregations

similar to fixed aggregations except that in this case they use variable length vectors or repeating groups. Implementation 7 for $R(A, B)$ is called the variable aggregation of B under A and it consists of a set of ordered 3-tuples such as:

$$\{ \langle a_1, c_1, va_1(B) \rangle, \langle a_2, c_2, va_2(B) \rangle, \dots, \langle a_{\alpha}, c_{\alpha}, va_{\alpha}(B) \rangle \}.$$

In the above set $a_1, \dots, a_{\alpha}$ are unique A -values, $va_i(B)$ is a variable length vector (or repeating group) containing B -values related to a_i in R and c_i is a counter data item occurrence (of size CNTRS) that contains the length of the vector $va_i(B)$, for $i = 1, \dots, \alpha$. The vector $va_i(B)$ and data item occurrence c_i are stored in the same record in which a_i is stored. B -values in the vectors of this implementation are shared occurrences in the sense that the implementations of all other relations over B which do not use duplicate B -values should use the same occurrences of data item type B .

Implementation 8 for $R(A, B)$, called variable aggregation of A under B , is the converse of implementation 7 and it consists of the set of ordered 3-types:

$$\{ \langle b_1, c_1, va_1(A) \rangle, \langle b_2, c_2, va_2(A) \rangle, \dots, \langle b_{\beta}, c_{\beta}, va_{\beta}(A) \rangle \}.$$

In this set $b_1, \dots, b_{\beta}$ are unique B -values, $va_i(A)$ is a vector of A -values related to b_i in R , c_i is a counter data item occurrence that contains the length of $va_i(A)$ and finally c_i and $va_i(A)$ are stored in the same record in

which b_i is stored, $i = 1, \dots, \beta'$. Again B-values stored in the variable-length vectors of this implementation are shared occurrences.

Implementations 1 through 8 constitute the first category of implementations, namely those that require the two data item types A and B of the relation $R(A, B)$ to be defined in one record type. These implementations determine the intra-record structure of the data base whereas the second category of implementations (to be defined in the following parts of this section) determine the inter-record structure of the data base. Implementations 1 through 8 are similar to those defined in [Mitoma 1975].

3.1.5 Chain with Next Pointers Associations

Implementation 9 for $R(A, B)$ is called the chain with next pointers association of B under A. Implementation 9 consists of the following two sets of ordered pairs.

$$\begin{aligned} \text{OWNR} &= \{ \langle a_1, mp_1 \rangle, \langle a_2, mp_2 \rangle, \dots, \langle a_\alpha, mp_\alpha \rangle \}, \\ \text{MMBR} &= \{ \langle b_1, np_1 \rangle, \langle b_2, np_2 \rangle, \dots, \langle b_\beta, np_\beta \rangle \} \end{aligned}$$

where

- i) $a_1, \dots, a_\alpha$, and $b_1, \dots, b_\beta$, are unique A- and B-values, respectively
- ii) mp_i is an address occurrence (pointer) that contains:
 - 1) a null address value if a_i is related to no B-value in R

- 2) the address of the record that contains the first B-value related to a_i in R, otherwise
 - iii) np_k is an address occurrence that contains:
 - 1) a null address value, if b_k is related to no A-value in R,
 - 2) the address of the record that contains the A-value a_i related to b_k in R, if b_k is the last B-value related to a_i in R,
 - 3) the address of the next B-value related to a_i in R, otherwise
 - iv) mp_i and np_k are stored in the same records in which a_i and b_k are stored, respectively.
- $i = 1, \dots, \alpha', k = 1, \dots, \beta'$

This implementation establishes the relation $R(A, B)$ in the data base through data base sets. Each data base set of this implementation contains exactly one owner record in which a unique A-value, a_i , is stored and zero or more member records in each of which a unique B-value, related to a_i in R, is stored. Data base sets of implementation 9 are stored as chains of records in the sense that in each set, the owner record contains a pointer mp_i in which the address of the first member record is stored and each member record of the set contains a pointer np_k in which the address of the next member record is stored. The next pointer, np_k , of the last member record contains the address of the owner record. The set of all set occurrences of this implementation constitutes a set type whose owner

record type is the set of all owner records. The set of all member records of these set occurrences is a subset of the member record type of the set type. The next member record pointers, np_k , in the records that belong to the member record type but do not belong to any data base set (of this implementation) contain null address occurrences (Figure 3.9).

There are, of course, some restrictions to the use of implementation 9 for a relation which will be discussed later. One important restriction, however, is that a record cannot belong to (participate as owner or member in) more than one occurrence of the same set type. In other words DBTG data base sets, by definition, can only implement one-to-many relationships. This is why we could talk about the A-value related to a B-value in R, in the definition of implementation 9.

Implementation 10 for $R(A, B)$ is the converse of implementation 9, it is called the chain with next pointers association of A under B and consists of the two sets of ordered pairs:

$$\begin{aligned} \text{OWNR} &= \{ \langle b_1, mp_1 \rangle, \langle b_2, mp_2 \rangle, \dots, \langle b_\beta, mp_\beta \rangle \}, \\ \text{MMBR} &= \{ \langle a_1, np_1 \rangle, \langle a_2, np_2 \rangle, \dots, \langle a_\alpha, np_\alpha \rangle \}. \end{aligned}$$

where

- i) $a_1, \dots, a_\alpha$, and $b_1, \dots, b_\beta$, are unique A- and B-values respectively,
- ii) mp_i is an address occurrence (pointer) that contains:

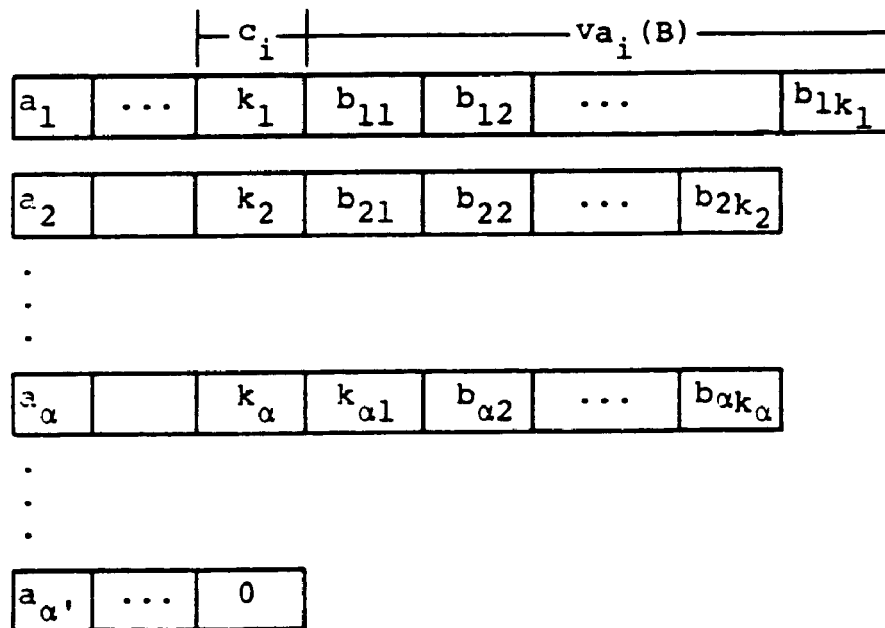


Figure 3.8 Variable Aggregation of B under A

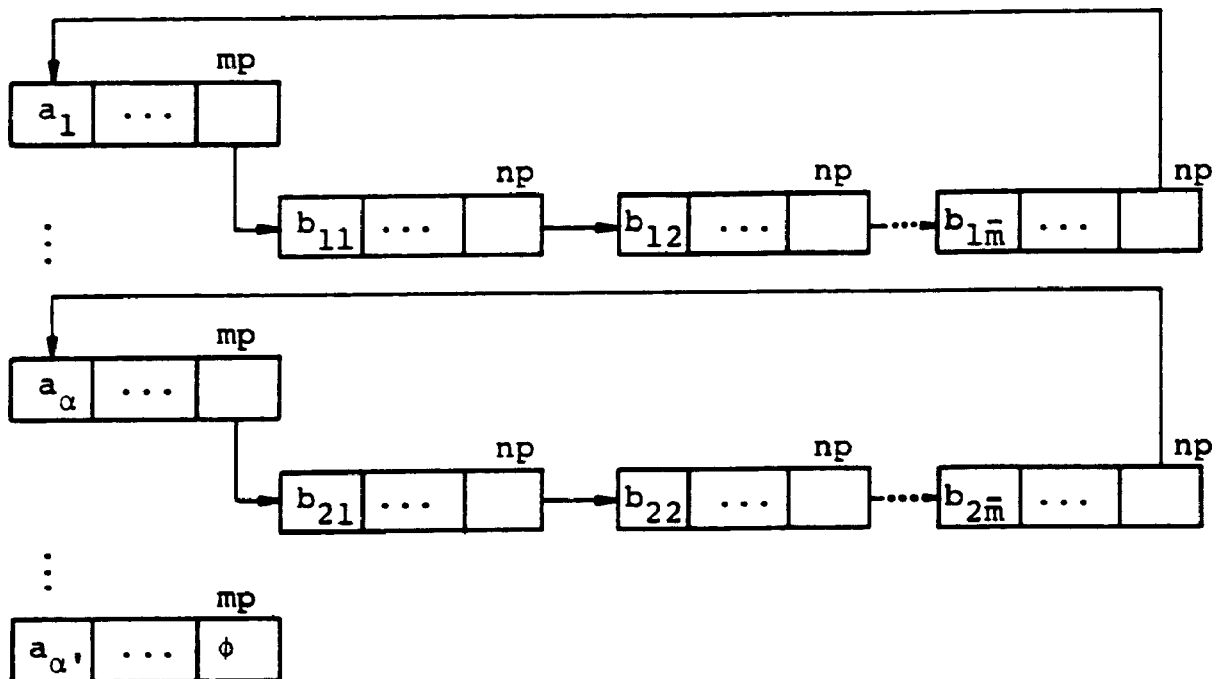


Figure 3.9 Chain with Next Pointers Association of B under A

- 1) a null address value if b_i is related to no A-value in R,
 - 2) the address of the record that contains the first A-value related to b_i in R, otherwise,
- iii) np_k is an address occurrence that contains:
- 1) a null address value, if a_k is related to no B-value in R,
 - 2) the address of the record that contains the B-value b_i related to a_k in R, if a_k is the last A-value related to b_i in R
 - 3) the address of the next A-value related to b_i in R, otherwise
- iv) mp_i and np_j are stored in the same records in which a_i and b_j are stored, respectively.
- $i = 1, \dots, \beta', k = 1, \dots, \alpha'.$

This implementation uses the same concepts as implementation 9 except in this case the owner records of the data base sets that establish the relation $R(A, B)$ in the data base contain unique B-values and the member records contain A-values. We also note that this implementation for $R(A, B)$ can only be used if at most one B-value is related to every A-value in R.

3.1.6 Chain with Next and Owner Pointers Associations

Implementation 11 for $R(A, B)$ is called the chain with next and owner pointers association of B under A and it consists of the two following sets.

$$\begin{aligned} \text{OWNR} &= \{ \langle a_1, mp_1 \rangle, \langle a_2, mp_2 \rangle, \dots, \langle a_\alpha, mp_\alpha \rangle \}, \\ \text{MMBR} &= \{ \langle b_1, np_1, op_1 \rangle, \langle b_2, np_2, op_2 \rangle, \dots, \\ &\quad \langle b_\beta, np_\beta, op_\beta \rangle \}. \end{aligned}$$

where

- i) $a_1, \dots, a_\alpha$, and $b_1, \dots, b_\beta$, are unique A- and B-values respectively,
- ii) mp_i is an address occurrence (pointer) that contains:
 - 1) a null address value, if a_i is related to no B-value in R,
 - 2) the address of the record in which the first B-value related to a_i in R is stored, otherwise
- iii) np_k is an address occurrence that contains:
 - 1) a null address value, if b_k is related to no A-value in R,
 - 2) the address of the record in which the A-value, a_i , related to b_j in R is stored, if b_k is the last B-value related to a_i in R,
 - 3) the address of the record in which the next B-value related to a_i in R is stored, otherwise,
- iv) op_k is an address occurrence that contains:
 - 1) a null address value, if b_k is related to no A-value in R,
 - 2) the address of the record in which the A-value related to b_k in R is stored,

otherwise,

- v) mp_i and $\langle np_k, op_k \rangle$ are stored in the same records in which a_i and b_k are stored, respectively.

$$i = 1, \dots, \alpha' \quad \text{and} \quad k = 1, \dots, \beta'.$$

This implementation is similar to implementation 9, it establishes the relation $R(A, B)$ in the data base, through data base sets, where the owner record of each set contains a unique A-value, a_i , and (zero or more) members records of the set contain unique B-values related to a_i in R . Data base sets of this implementation are stored as chains of records, as discussed in the definition of implementation 9. However, in this case the (member) record in which b_k is stored $k = 1, \dots, \beta'$ contains, in addition to the next pointer np_j , another pointer, op_k , that holds a null address value if b_k is not related to any A-value in R , otherwise it holds the address of the record in which the A-value is stored. Again this implies that only one A-value (at most) can be related to each B-value and therefore if $R(A, B)$ is to be implemented by implementation 11 it must satisfy this requirement (Figure 3.10).

Implementation 12 for $R(A, B)$ is the converse of implementation 11, it is called the chain with next and owner pointers association of A under B and it consists of the following two sets.

$$OWNR = \{ \langle b_1, mp_1 \rangle, \langle b_2, mp_2 \rangle, \dots, \langle b_{\beta'}, mp_{\beta'} \rangle \},$$

$$\text{MMBR} = \{ \langle a_1, np_1, op_1 \rangle, \langle a_2, np_2, op_2 \rangle, \dots, \langle a_\alpha, np_\alpha, op_\alpha \rangle \}$$

where

- i) $a_1, \dots, a_\alpha$, and $b_1, \dots, b_\beta$, are unique A- and B-values, respectively,
- ii) mp_i is an address occurrence (pointer) that contains:
 - 1) a null address value if b_i is related to no A-value in R,
 - 2) the address of the record in which the first A-value related to b_i in R is stored, otherwise
- iii) np_k is an address occurrence that contains:
 - 1) a null address value, if a_k is related to no B-value in R,
 - 2) the address of the record in which the B-value (b_i) related to a_k in R is stored, if a_k is the last A-value related to b_i in R,
 - 3) the address of the next A-value related to b_i in R, otherwise,
- iv) op_k is an address occurrence that contains:
 - 1) a null address value if a_k is related to no B-value in R,
 - 2) the address of the record in which the B-value related to a_k in R is stored, otherwise,
- v) mp_i and $\langle np_k, op_k \rangle$ are stored in the same records

in which b_i and a_k are stored, respectively,
 $i = 1, \dots, \beta'$ and $k = 1, \dots, \alpha'$.

This implementation is the converse of implementation 11 in the sense that in this case unique B-values are stored in the owner records of the data base sets of implementation 12 and unique A-values are stored in the member records. We also note that as in the case of implementation 10, implementation 12 for $R(A, B)$ can only be used if at most one B-value is related to every A-value in R .

3.1.7 Chain with Next and Prior Pointers Associations

Implementation 13 for $R(A, B)$ is called the chain with next and prior pointers association of B under A and it consists of the two following sets of 3-tuples.

$$\begin{aligned} \text{OWNR} &= \{ \langle a_1, mp_1, lp_1 \rangle, \langle a_2, mp_2, lp_2 \rangle, \dots, \\ &\quad \langle a_\alpha, mp_\alpha, lp_\alpha \rangle \}, \\ \text{MMBR} &= \{ \langle b_1, np_1, pp_1 \rangle, \langle b_2, np_2, pp_2 \rangle, \dots, \\ &\quad \langle b_\beta, np_\beta, pp_\beta \rangle \} \end{aligned}$$

where

- i) $a_1, \dots, a_\alpha$ and $b_1, \dots, b_\beta$ are unique A- and B-values, respectively,
- ii) mp_i is an address occurrence that contains:
 - 1) a null address value, if a_i is related to no B-value in R ,
 - 2) the address of the record in which the first B-value related to a_i in R is stored, otherwise,

- iii) lp_i is an address occurrence that contains:
 - 1) a null address value, if a_i is related to no B-value in R,
 - 2) the address of the record in which the last B-value related to a_i in R is stored, otherwise,
- iv) np_k is an address occurrence that contains:
 - 1) a null address value, if b_k is related to no A-value in R,
 - 2) the address of the record in which the A-value, a_i , related to b_k in R is stored, if b_k is the last B-value related to a_i in R,
 - 3) the address of the record in which the next B-value related to a_i in R is stored, otherwise,
- v) pp_k is an address occurrence that contains:
 - 1) a null address value, if b_k is related to no A-value in R,
 - 2) the address of the record in which the A-value, a_i , related to b_k in R is stored, if b_k is the first B-value related to a_i in R,
 - 3) the address of the record in which the prior B-value related to a_i in R is stored, otherwise,
- vi) $\langle mp_i, lp_i \rangle$ and $\langle np_k, pp_k \rangle$ are stored in the same records in which a_i and b_k are stored, respec-

tively,

$i = 1, \dots, \alpha'$ and $k = 1, \dots, \beta'$

This implementation is similar to implementation 9, it establishes the relation $R(A, B)$, in the data base, through data base sets where the owner record of each set contains a unique A-value, a_i , and (zero or more) member records of the set contain unique B-values related to a_i in R . Data base sets of this implementation are stored as chains of records, as discussed in definition of implementation 9. However, in this case the owner record in which a_i , $i = 1, \dots, \alpha'$ is stored contains, in addition to the member pointer mp_i , another pointer lp_i that holds a null address value if a_i is not related to any B-value, otherwise it holds the address of the record in which the last B-value related to a_i in R is stored. Also the member record in which b_k , $k = 1, \dots, \beta'$ is stored contains, in addition to the next pointer np_k , another pointer pp_k that holds a null address value if b_k is not related to any A-value in R . If b_k is related to some A-value, a_i , in R , then pp_k contains the address of the record in which a_i is stored in case b_k is the first B-value related to a_i in R otherwise pp_k contains the address of the record in which the prior B-value related to a_i in R is stored (Figure 3.11).

Again the relation $R(A, B)$ should be such that at most one A-value is related to each B-value if implementation 13 is to be used for the relation.

Implementation 14 is the converse of implementation 13,

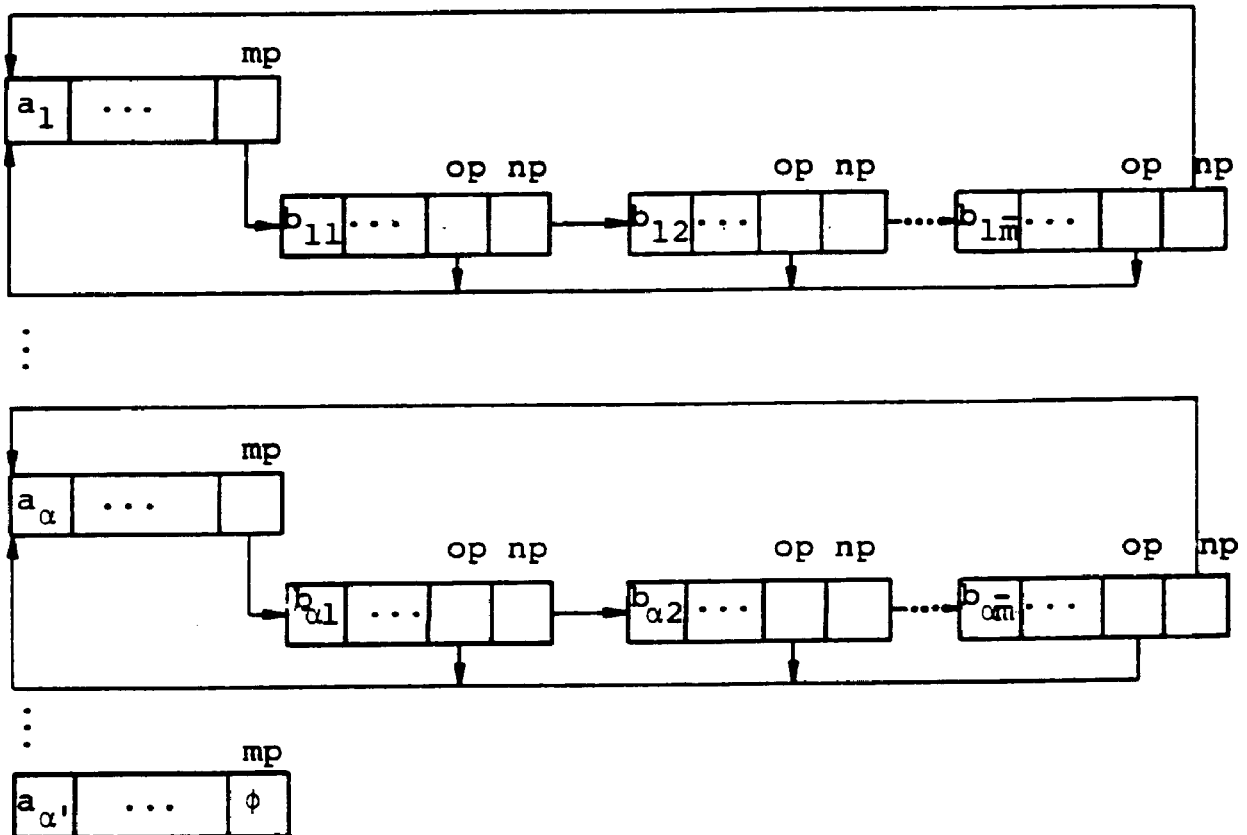


Figure 3.10 Chain with Next and Owner Pointers Association of B under A

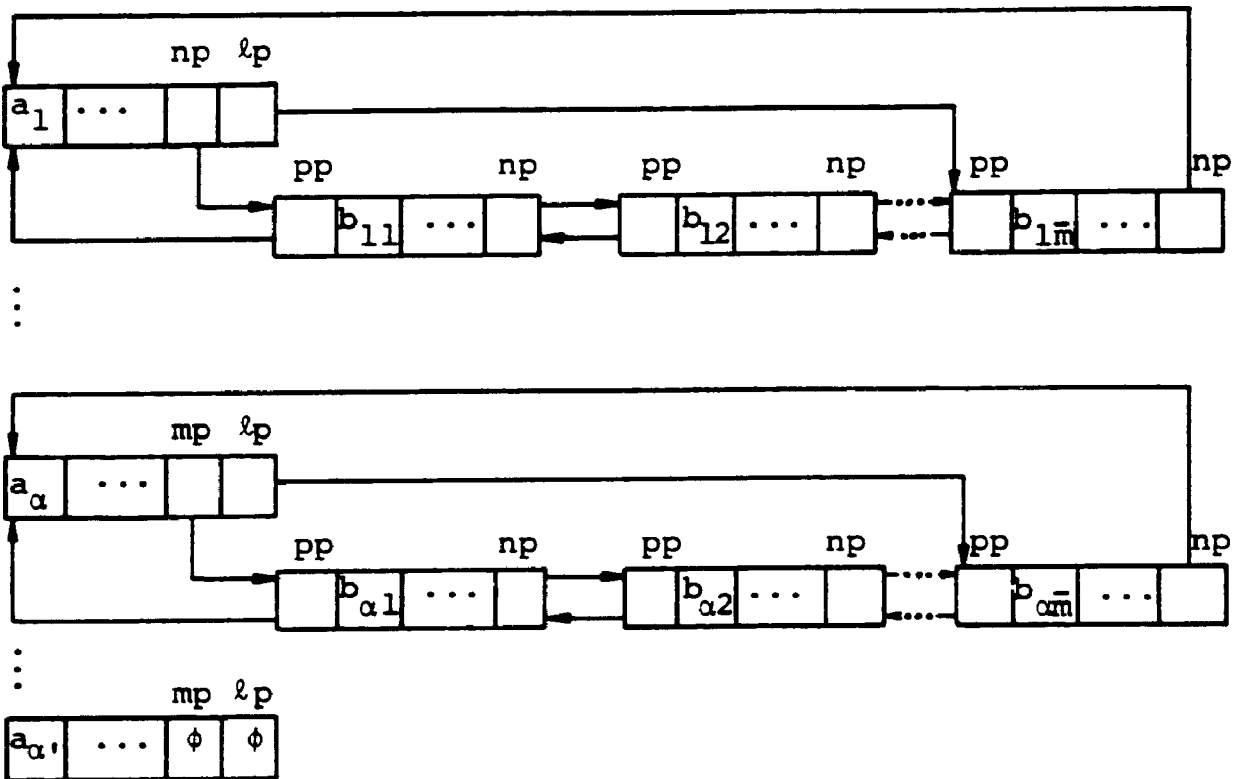


Figure 3.11 Chain with Next and Prior Pointers Association of B under A

it is called the chain with next and prior pointers association of A under B and it consists of the following two sets of 3-tuples.

$$\text{OWNR} = \{ \langle b_1, mp_1, lp_1 \rangle, \langle b_2, mp_2, lp_2 \rangle, \dots, \langle b_\beta, mp_\beta, lp_\beta \rangle \},$$

$$\text{MMBR} = \{ \langle a_1, np_1, pp_1 \rangle, \langle a_2, np_2, pp_2 \rangle, \dots, \langle a_\alpha, np_\alpha, pp_\alpha \rangle \},$$

where:

- i) $a_1, \dots, a_\alpha$ and $b_1, \dots, b_\beta$ are unique A- and B-values, respectively,
- ii) mp_i is an address occurrence that contains:
 - 1) a null address value, if b_i is not related to any A-value in R,
 - 2) the address of the record in which the first A-value related to b_i in R is stored, otherwise,
- iii) lp_i is an address occurrence that contains:
 - 1) a null address value, if b_i is related to no A-value in R,
 - 2) the address of the record in which the last A-value related to b_i in R is stored, otherwise,
- iv) np_k is an address occurrence that contains:
 - 1) a null address value, if a_k is related to no B-value in R,
 - 2) the address of the record in which the B-value, b_i , related to a_j in R is stored,

- if a_k is the last A-value related to b_i in R ,
- 3) the address of the record in which the next B-value related to a_i in R is stored, otherwise,
- v) pp_k is an address occurrence that contains:
- 1) a null address value if a_k is not related to any B-value in R ,
 - 2) the address of the record in which the B-value, b_i , related to a_k in R is stored, if a_k is the first A-value related to b_i in R ,
 - 3) the address of the record in which the prior A-value related to b_i in R is stored, otherwise,
- vi) $\langle mp_i, lp_i \rangle$ and $\langle np_k, pp_k \rangle$ are stored in the same records in which b_i and a_k are stored, respectively,
- $i = 1, \dots, \beta'$ and $k = 1, \dots, \alpha'$.

This implementation is the converse of implementation 13 in the sense that in this case unique B-values are stored in the owner records of the data base sets of implementation 14 and unique A-values are stored in the member records. Implementation 14 for $R(A, B)$ can only be used if at most one B-value is related to each A-value in R .

3.1.8 Chain with Next, Prior and Owner Pointers Associations

Implementation 15 for $R(A, B)$ is called the chain

with next, prior and owner pointers association of B under A and it consists of the two following sets.

$$\text{OWNR} = \{ \langle a_1, mp_1, lp_1 \rangle, \langle a_2, mp_2, lp_2 \rangle, \dots, \langle a_\alpha, mp_\alpha, lp_\alpha \rangle \},$$

$$\text{MMBR} = \{ \langle b_1, np_1, pp_1, op_1 \rangle, \langle b_2, np_2, pp_2, op_2 \rangle, \dots, \langle b_\beta, np_\beta, pp_\beta, op_\beta \rangle \},$$

where

- i) $a_1, \dots, a_\alpha$, and $b_1, \dots, b_\beta$, are unique A- and B-values, respectively,
- ii) mp_i is an address occurrence that contains:
 - 1) a null address value, if a_i is not related to any B-value in R,
 - 2) the address of the record in which the first B-value related to a_i in R is stored, otherwise,
- iii) lp_i is an address occurrence that contains:
 - 1) a null address value, if a_i is related to no B-value in R,
 - 2) the address of the record in which the last B-value related to a_i in R is stored, otherwise,
- iv) np_k is an address occurrence that contains:
 - 1) a null address value, if b_k is related to no A-value in R,
 - 2) the address of the record in which the A-value, a_i , related to b_k in R is stored, if b_k is the last B-value related to a_i in R,

- 3) the address of the record in which the next B-value related to a_i in R is stored, otherwise,
- v) pp_k is an address occurrence that contains:
- 1) a null address value, if b_k is related to no A-value in R,
 - 2) the address of the record in which the A-value, a_i , related to b_k in R is stored if b_k is the first B-value related to a_i in R,
 - 3) the address of the record in which the prior B-value related to a_i in R is stored, otherwise,
- vi) op_k is an address occurrence that contains:
- 1) a null address value, if b_k is related to no A-value in R,
 - 2) the address of the record in which the A-value, a_i , related to b_k in R is stored, otherwise,
- vii) $\langle mp_i, lp_i \rangle$ and $\langle np_k, pp_k, op_k \rangle$ are stored in the same records in which a_i and b_k are stored, respectively,
- $i = 1, 2, \dots, \alpha'$ and $k = 1, 2, \dots, \beta'$.

This implementation is, in some sense, a combination of implementation 11 and 13. As in the case of implementation 11 and 13, implementation 15 for $R(A, B)$ establishes the relation in the data base by data base sets. The owner record of each such set contains a unique A-value, a_i , and

(zero or more) member record(s) of that set contain unique B-values related to a_i in R. Data base sets of this implementation are stored as chains of records as discussed in the definition of implementation 9. In data base sets of implementation 15, however, member records contain both owner and prior pointers in addition to the next pointers. This implementation is illustrated in Figure 3.12, and it can be used for relation $R(A, B)$ only if at most one A-value is related to each B-value in R.

Implementation 16 is the converse of implementation 15, it is called the chain with next, prior and owner pointers association of A under B and it consists of the following two sets.

$$\begin{aligned} \text{OWNR} &= \{ \langle b_1, mp_1, lp_1 \rangle, \langle b_2, mp_2, lp_2 \rangle, \dots, \\ &\quad \langle b_\beta, mp_\beta, lp_\beta \rangle \}, \\ \text{MMBR} &= \{ \langle a_1, np_1, pp_1, op_1 \rangle, \langle a_2, np_2, pp_2, op_2 \rangle, \dots, \\ &\quad \langle a_\alpha, np_\alpha, pp_\alpha, op_\alpha \rangle \}, \end{aligned}$$

where

- i) $a_1, \dots, a_\alpha$, and $b_1, \dots, b_\beta$, are unique A- and B-values, respectively,
- ii) mp_i is an address occurrence that contains:
 - 1) a null address value if b_i is related to no A-value in R,
 - 2) the address of the record in which the first A-value related to b_i in R is stored, otherwise,

- iii) ℓp_i is an address occurrence that contains:
 - 1) a null address value, if b_i is related to no A-value in R,
 - 2) the address of the record in which the last A-value related to b_i in R is stored, otherwise,
- iv) np_k is an address occurrence that contains:
 - 1) a null address value, if a_k is related to no B-value in R,
 - 2) the address of the record in which the B-value, b_i , related to a_k in R is stored, if a_k is the last A-value related to b_i in R,
 - 3) the address of the record in which the next A-value related to b_i in R is stored, otherwise,
- v) pp_k is an address occurrence that contains:
 - 1) a null address value, if a_k is related to no B-value in R,
 - 2) the address of the record in which the B-value, b_i , related to a_k in R is stored if a_k is the first A-value related to b_i in R,
 - 3) the address of the record in which the prior A-value related to b_i in R is stored, otherwise,
- vi) op_k is an address occurrence that contains:
 - 1) a null address value if a_k is related to no B-value in R,

- 2) the address of the record in which the B-values, b_i , related to a_k in R is stored, otherwise,
- vii) $\langle mp_i, lp_i \rangle$ and $\langle np_k, pp_k, op_k \rangle$ are stored in the same record in which b_i and a_k are stored, respectively,
- $i = 1, 2, \dots, \beta'$ and $k = 1, 2, \dots, \alpha'$.

This implementation is the converse of implementation 15 in the sense that in this case unique B-values are stored in the owner records of the data base sets of implementation 16 and unique A-values are stored in the member records. Implementation 16 for $R(A, B)$ can only be used if at most one B-value is related to each A-value in R .

3.1.9 Pointer Array Associations

Implementation 17 for $R(A, B)$ is called the pointer array association of B under A and it consists of the two following sets.

$$\text{OWNR} = \{ \langle a_1, vp_1, pa_1 \rangle, \langle a_2, vp_2, pa_2 \rangle, \dots, \langle a_{\alpha'}, vp_{\alpha'}, pa_{\alpha'} \rangle \},$$

$$\text{MMBR} = \{ b_1, b_2, \dots, b_{\beta'} \},$$

where

- i) $a_1, \dots, a_{\alpha'}$, and $b_1, \dots, b_{\beta'}$, are unique A- and B-values, respectively,
- ii) pa_i is a contiguous sequence (array) of M address occurrences where M is the maximum number of B-values related to one A-value in R . On the

average the first r/α' elements of the array contain addresses of records in which B-values related to a_i in R are stored. The rest of the elements of the array contain null address values.

- iii) vp_i is an address occurrence that contains the address of the first element of pa_i .
 - iv) vp_i is stored in the same record in which a_i is stored,
- $i = 1, \dots, \alpha'$.

This implementation establishes the relation $R(A, B)$ in the data base through data base sets. Each data base set of implementation 17 contains exactly one owner record in which a unique A-value, a_i , is stored and zero or more member record(s) in which unique B-values related to a_i in R are stored. Data base sets, in this implementation are stored using the concept of pointer arrays. The data base set in whose owner record a_i is stored contains a vector pointer vp_i , that points to an array of pointers pa_i . The array pa_i is a sequence of M address occurrences stored in contiguous storage locations. If the number of B-values related to a_i in R is t_i where $0 \leq t_i \leq M$ then the first t_i elements of the array pa_i contain addresses of the records in which the related B-values are stored and the remaining $M-t_i$ elements contain null address values. We assume that the ordering of B-values is preserved by the ordering of the pointers in the array. This is to say that for

$k' = 1, 2, \dots, t_i$, the k' -th element of pa_i contains the address of the record in which the k' -th B-value related to a_i in R is stored.

As in the case of implementation 9 the set of all such data base sets constitute a data base set type whose owner record type is the set of all owner records of these data base sets. The set of all member records of these data base sets is a subset of the member record type of the set type. Implementation 17 is illustrated in Figure 3.13.

Implementation 18 for $R(A, B)$ is called the pointer array association of A under B . This implementation is the converse of implementation 17 and consists of the following two sets.

$$\begin{aligned} \text{OWNR} &= \{ \langle b_1, vp_1, pa_1 \rangle, \langle b_2, vp_2, pa_2 \rangle, \dots, \\ &\quad \langle b_{\beta}, vp_{\beta}, pa_{\beta} \rangle \}, \\ \text{MMBR} &= \{ a_1, a_2, \dots, a_{\alpha} \}, \end{aligned}$$

where

- i) $a_1, \dots, a_{\alpha}$, and $b_1, \dots, b_{\beta}$, are unique A- and B-values, respectively,
- ii) pa_i is a contiguous sequence (array) of N address occurrences where N is the maximum number of A-values related to one B-value in R . On the average, the first r/β elements of the array contain addresses of records in which A-values related to b_i in R are stored. The rest of the elements of pa_i contain null address values.
- iii) vp_i is an address occurrence that contains the

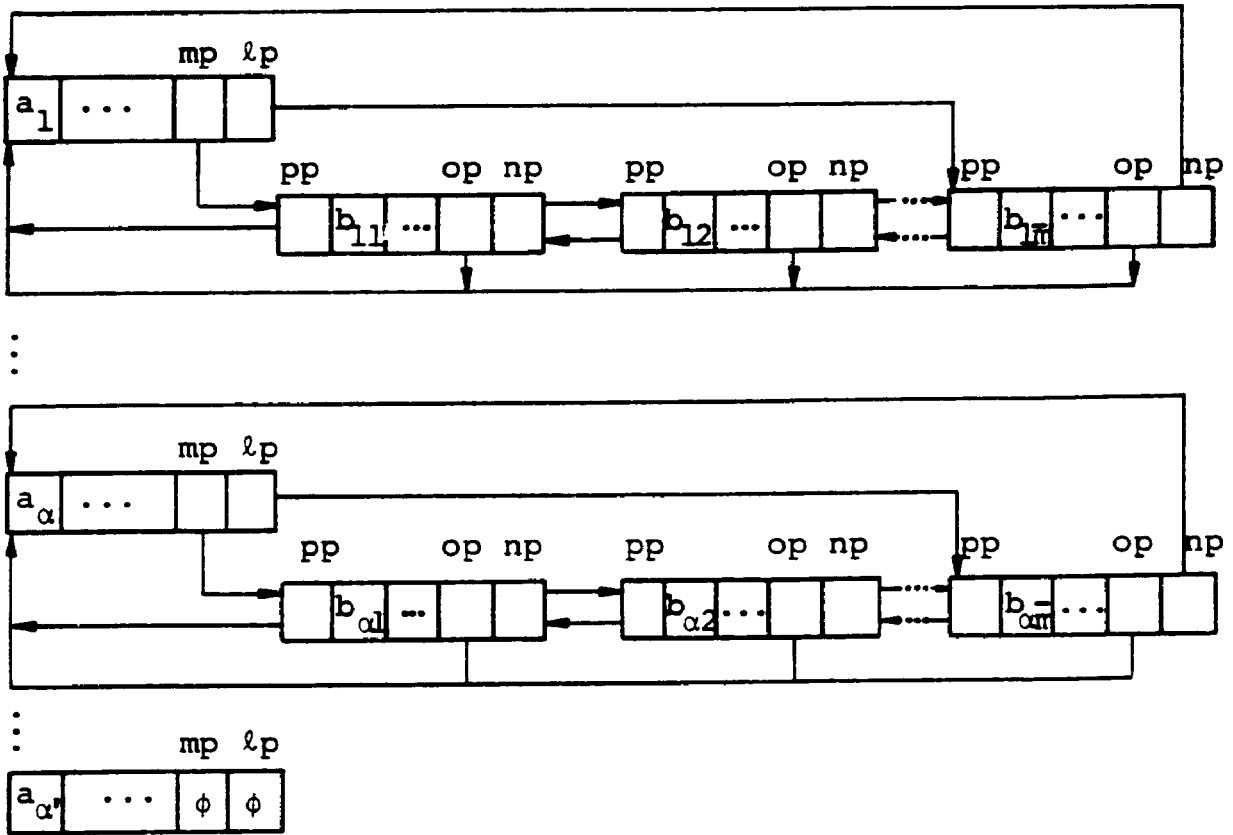


Figure 3.12 Chain with Next, Prior and Owner Pointers Association of B under A

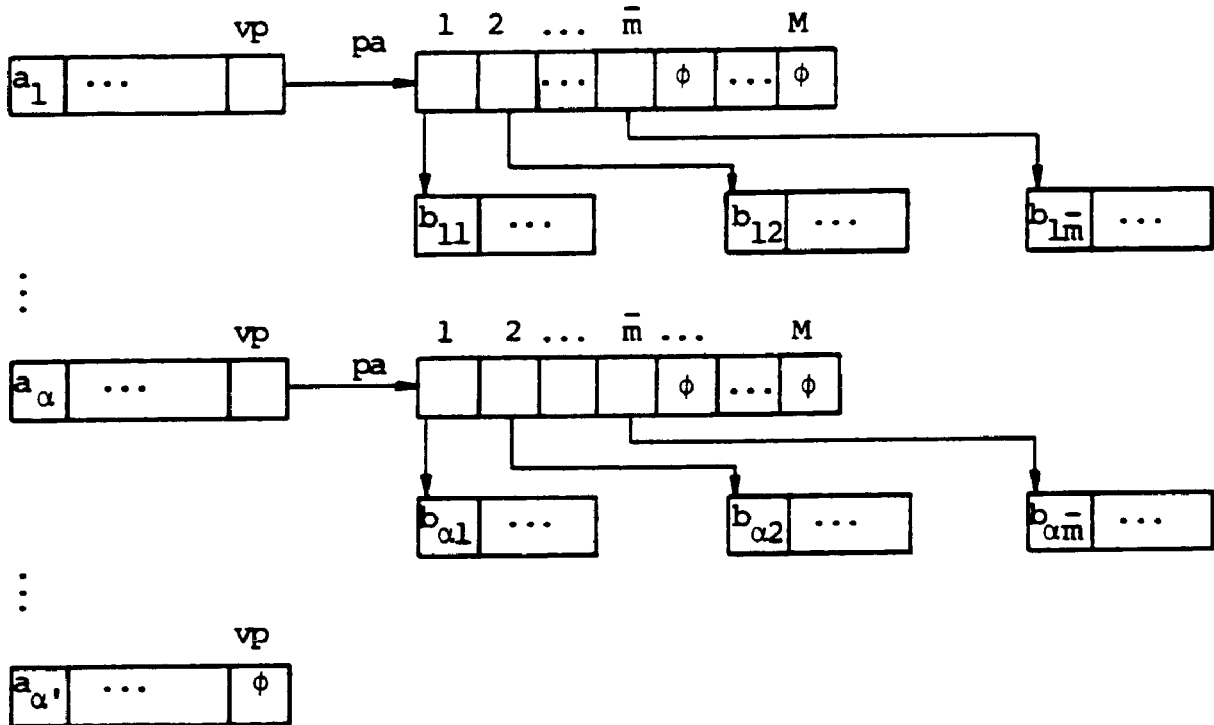


Figure 3.13 Pointer Array Association of B under A

address of the first element of pa_i .

iv) vp_i is stored in the same record in which b_i is stored,

$i = 1, 2, \dots, \beta'$.

This implementation is different from implementation 17 in that in this case owner records of the data base sets that establish the relation $R(A, B)$ in the data base contain unique B-values and the member records contain unique A-values. The pointer array pa_i has N elements where the k' -th element contains the address of the record in which the k' -th A-value related to b_i in R is stored, for $k' = 1, \dots, t'_i$ where t'_i is the number of A-values related to b_i in R. The remaining $N - t'_i$ elements of pa_i contain null address occurrences.

3.1.10 Pointer Array with Owner Pointer Associations

Implementation 19 for $R(A, B)$ is called the pointer array with owner pointers association of B under A and it consists of the following sets.

$$OWNR = \{ \langle a_1, vp_1, pa_1 \rangle, \langle a_2, vp_2, pa_2 \rangle, \dots, \langle a_\alpha, vp_\alpha, pa_\alpha \rangle \},$$

$$MMBR = \{ \langle b_1, op_1 \rangle, \langle b_2, op_2 \rangle, \dots, \langle b_\beta, op_\beta \rangle \},$$

where

- i) $a_1, \dots, a_\alpha$, and $b_1, \dots, b_\beta$, are unique A- and B-values, respectively,
- ii) pa_i is a contiguous sequence (array) of M address occurrences where M is the maximum num-

ber of B-values related to one A-value in R.

On the average the first r/α' elements of pa_i contain addresses of records in which B-values related to a_i in R are stored. The rest of the elements of pa_i contain null address values.

- iii) vp_i is an address occurrence that contains the address of the first element of pa_i .
- iv) op_k is an address occurrence that contains:
 - 1) a null address value, if b_k is related to no A-value in R,
 - 2) the address of the record in which the A-value, a_i , related to b_k in R is stored, otherwise,
- v) vp_i and op_k are stored in the same records in which a_i and b_k are stored, respectively,
 $i = 1, 2, \dots, \alpha'$ and $k = 1, 2, \dots, \beta'$.

This implementation is similar to implementation 17, it establishes the relation $R(A, B)$ in the data base, through data base sets, where the owner records of each set contains a unique A-value, a_i , and (zero or more) member records of the set contain unique B-values related to a_i in R. Data base sets of implementation 19 are stored using the pointer array technique as discussed in the definition of implementation 17. However, in this case, the (member) record in which b_k is stored, $k = 1, \dots, \beta'$, contains a pointer, op_k , that holds a null address value if b_k is not related to any A-value in R, otherwise it

holds the address of the record in which the A-value related to b_k in R is stored. This implementation can be used only for relations in which at most one A-value is related to each B-value. Figure 3.14 is an illustration of this implementation.

Implementation 20 for $R(A, B)$ is the converse of implementation 19, it is called the pointer array with owner pointers association of A under B and it consists of the following two sets.

$$\begin{aligned} \text{OWNR} &= \{ \langle b_1, vp_1, pa_1 \rangle, \langle b_2, vp_2, pa_2 \rangle, \dots, \\ &\quad \langle b_\beta, vp_\beta, pa_\beta \rangle \}, \\ \text{MMBR} &= \{ \langle a_1, op_1 \rangle, \langle a_2, op_2 \rangle, \dots, \langle a_\alpha, op_\alpha \rangle \}, \end{aligned}$$

where

- i) $a_1, \dots, a_\alpha$, and $b_1, \dots, b_\beta$, are unique A- and B-values, respectively,
- ii) pa_i is a contiguous sequence (array) of N address occurrences where N is the maximum number of A-values related to one B-value in R. On the average the first r/β elements of pa_i contain addresses of records in which A-values related to b_i in R are stored. The rest of the elements of pa_i contain null address values.
- iii) vp_i is an address occurrence that contains the address of the first element of pa_i .
- iv) op_k is an address occurrence that contains:
 - 1) a null address value if a_k is related to no B-value in R,

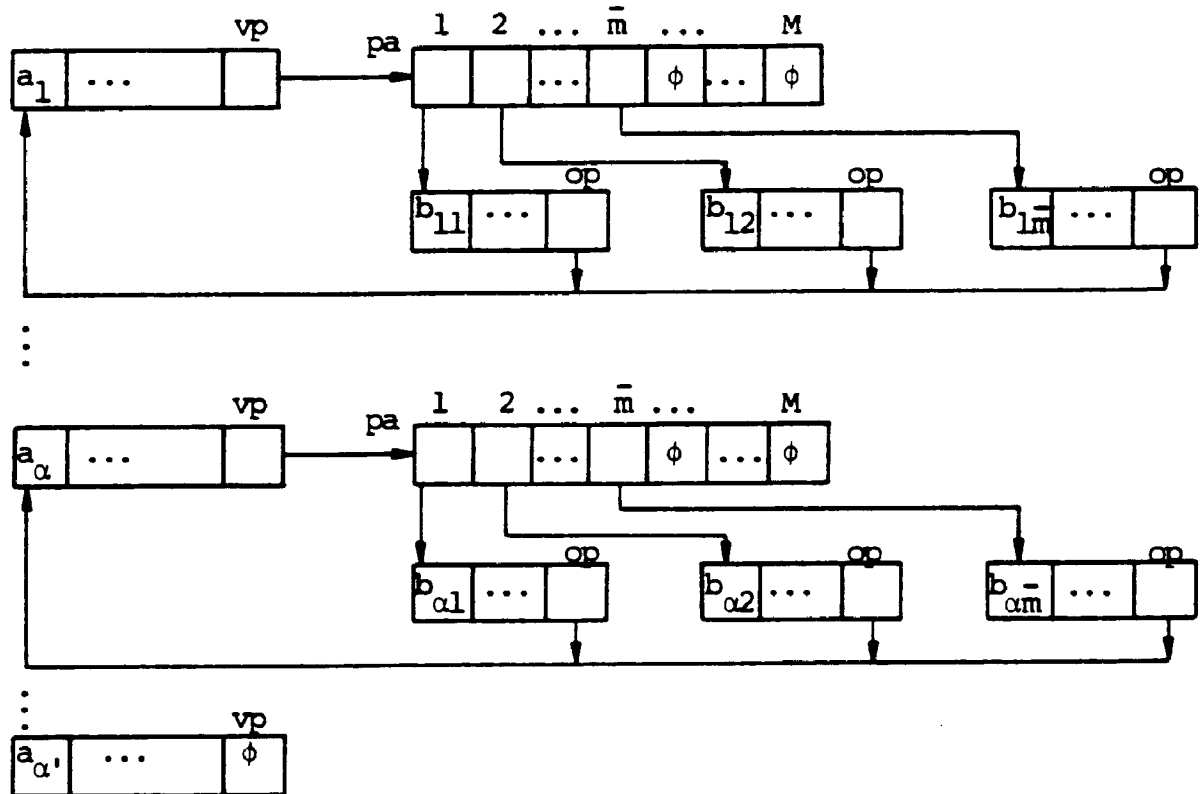


Figure 3.14 Pointer Array with Owner Pointers Association of B under A

- 2) the address of the record in which the B-value, b_i , related to a_k in R is stored, otherwise,
- v) vp_i and op_k are stored in the same records in which b_i and a_k are stored, respectively,
 $i = 1, 2, \dots, \beta'$ and $k = 1, 2, \dots, \alpha'$.

This implementation is different from implementation 20 in that in this case the owner records of the data base sets that establish the relation $R(A, B)$ in the data base contain unique B-values and the member records contain unique A-values. The pointer array pa_i has N elements with the first t_i' elements containing addresses of member records and the remaining $N - t_i'$ elements containing null address values, where t_i' is the number of A-values related to one B-value in R . For $k' = 1, 2, \dots, t_i'$ the k' -th element of pa_i contains the address of the record in which the k' -th A-value related to b_i in R is stored. Implementation 20 for relation $R(A, B)$ can only be used if at most one B-value is related to each A-value in R .

Implementation 9 through 20 all use the concept of data base sets to implement a relation such as $R(A, B)$ and only one data base set type is necessary to establish $R(A, B)$ in the data base. What makes these implementations different from one another is the set implementation technique that is used in each case and whether data item A is defined as part of the owner record type with B defined as part of the member record type or vice versa. These

implementations all have the limitation that they only implement one-to-many relationships.

3.1.11 Dummy Record Association of A and B

Implementation 21 for $R(A, B)$ is called the dummy record association of A and B and it consists of the following three sets.

$$\begin{aligned} \text{OWNER} &= \{ \langle a_1, mp_1 \rangle, \langle a_2, mp_2 \rangle, \dots, \langle a_\alpha, mp_\alpha \rangle \}, \\ \text{OWNER}' &= \{ \langle b_1, mp'_1 \rangle, \langle b_2, mp'_2 \rangle, \dots, \langle b_\beta, mp'_\beta \rangle \}, \\ \text{LINK} &= \{ \langle op'_1, op_1, np'_1, np_1 \rangle, \langle op'_2, op_2, np'_2, np_2 \rangle, \dots, \\ &\quad \langle op'_r, op_r, np'_r, np_r \rangle \}, \end{aligned}$$

where

- i) $a_1, \dots, a_\alpha$, and $b_1, \dots, b_\beta$, are unique A- and B-values, respectively,
- ii) $\langle op'_k, op_k, np'_k, np_k \rangle$, $k = 1, \dots, r$ is a contiguous sequence of 4 address occurrences where r is the cardinality of $R(A, B)$,
- iii) for each pair $\langle a, b \rangle \in R$ there exists exactly one 4-tuple $\langle op'_k, op_k, np'_k, np_k \rangle \in \text{LINK}$ (and vice versa) in which:
 - 1) op_k contains the address of the record in which a is stored;
 - 2) op'_k contains the address of the record in which b is stored;
 - 3) np_k contains the address of the first element of the 4-tuple in LINK associated with the pair $\langle a, b' \rangle \in R$ such that b' is the next

(with respect to b) B-value related to a in R . If b is the last B-value related to a in R then np_k contains the address of the record in which a is stored,

- 4) np'_k contains the address of the first element of the 4-tuple in LINK associated with the pair $\langle a', b \rangle \in R$ such that a' is the next (with respect to a) A-value related to b in R . If a is the last a -value related to b in R then np'_k contains the address of the record in which b is stored,

iv) mp_i is an address occurrence that contains:

- 1) a null address value if a_i is related to no B-value in R ,
- 2) the address of the first element of the 4-tuple in LINK associated with $\langle a_i, b \rangle$ in R such that b is the first B-value related to a_i in R ,

v) mp'_ℓ is an address occurrence that contains:

- 1) a null address value if b_ℓ is related to no A-value in R ,
- 2) the address of the first element of the 4-tuple in LINK associated with $\langle a, b_\ell \rangle$ in R such that a is the first A-value related to b_ℓ in R ,

vi) mp_i and mp'_ℓ are stored in the same records in which a_i and b_ℓ are stored, respectively,

- vii) 4-tuples $\langle op'_k, op_k, np'_k, np_k \rangle$, $k = 1, \dots, r$
are called dummy records.

Implementation 21 for $R(A, B)$ uses the concept of data base sets to establish the relation $R(A, B)$ in the data base. However, in contrast to other associations, implementation 21 uses two data base set types. Each data base set of the first type consists of exactly one owner record in which a unique A-value, a_i , is stored and zero or more member records that are dummy records associated with pairs $\langle a_i, b_\ell \rangle$ for all ℓ such that $\langle a_i, b_\ell \rangle \in R$. Similarly, each data base set of the second type consists of exactly one owner record in which a unique B-value, b_i , is stored and zero or more member records that are dummy records associated with pairs $\langle a_\ell, b_i \rangle$ for all ℓ such that $\langle a_\ell, b_i \rangle \in R$. The set of all owner records of data base sets of the first (second) type constitute the owner record type of the first (second) data base set type. The set of all member records (dummy records in this case) constitute the member record type of both set types.

Contrary to all other association implementations, implementation 21 may be used to implement many-to-many relationships.

The reason is that dummy records that constitute member records of this implementation, although of the same record type, belong to data base sets of different set types. Each data base set of implementation 21, of either type, is stored with a set implementation technique iden-

tical to that of chain with next and owner pointer associations discussed in 3.1.6. Other techniques (or combinations of other techniques) may be used for storing data base sets of this implementation thereby producing many more new relation implementation alternatives. We are not including those other alternatives because they do not bear any new ideas and they can easily be added to the model if desired. The technique adopted above has a reasonable storage/time tradeoff. It was superior to some other possible techniques which we tried.

Implementation 21 is illustrated in Figure 3.15.

3.1.12 Linkages

Implementation 22 for $R(A, B)$ is called the single linkage of B under A and it consists of the following two sets.

$$\{ \langle a_1, kp_1 \rangle, \langle a_2, kp_2 \rangle, \dots, \langle a_{\alpha}, kp_{\alpha} \rangle \},$$

$$\{ b_1, b_2, \dots, b_{\beta} \},$$

where

- i) $a_1, \dots, a_{\alpha}$ and $b_1, \dots, b_{\beta}$ are unique A- and B-values, respectively,
- ii) kp_i is a vector of M address occurrences where M is the maximum number of B-values related to an A-value in R ,
- iii) If t_i ($0 \leq t_i \leq M$) is the number of B-values related to a_i in R , then for $k = 1, \dots, t_i$, the k -th element of kp_i contains the address of

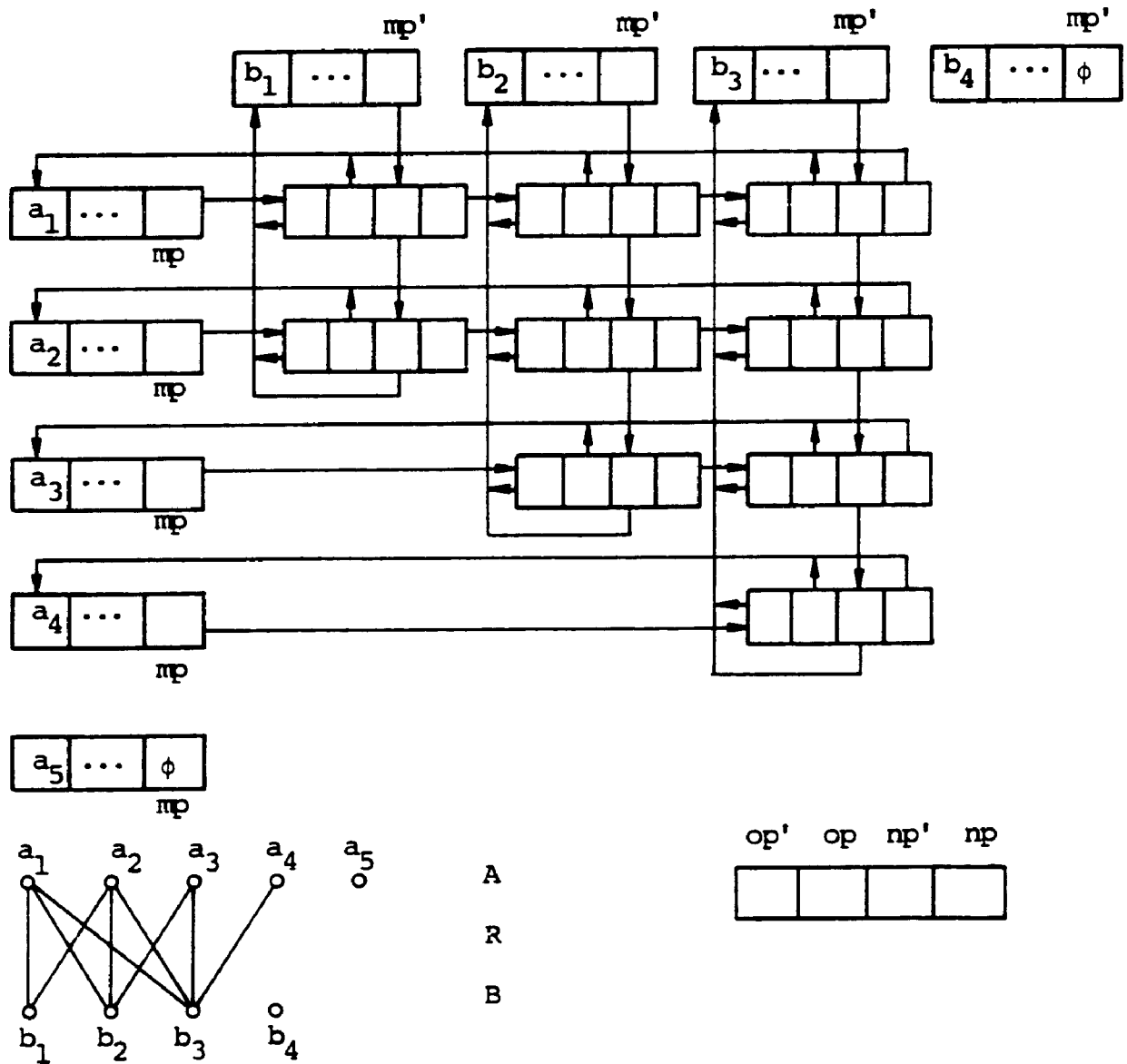


Figure 3.15 Dummy Record Association of A and B

the record in which the k -th B-value related to a_i in R is stored. The remaining $M-t_i$ elements of kp_i contain null address values. Note that if a_i is related to no B-value in R ($t_i = 0$) then all M elements of kp_i contain null address values,

- iv) kp_i is stored in the same record in which a_i is stored,
- $i = 1, \dots, \alpha'$.

Implementation 22 establishes the relation $R(A, B)$ in the data base by storing vectors of pointer (data base key) data items in records. The record in which a_i ($i = 1, \dots, \alpha'$) is stored contains a fixed-length vector kp_i of pointers (data base keys). On the average r/α' elements of kp_i contain addresses of records in which B-values related to a_i in R are stored. The rest of the elements of kp_i contain null address values.

Contrary to association implementations, implementation 22 does not use data base sets although relationships are realized through pointers. Pointers used in implementation 22 may be explicitly retrieved and/or updated, like any other data item, by programs accessing the data base, whereas pointers used to store data base sets are created and maintained by the data base management system.

The fact that no data base sets are used in implementation 22 makes it possible to implement a relation $R(A, B)$ even if it is not a one-to-many relation. Implementation 22

is illustrated in Figure 3.16.

Implementation 23 for $R(A, B)$ is called the single linkage of A under B and it consists of the following two sets.

$$\{ \langle b_1, kp_1 \rangle, \langle b_2, kp_2 \rangle, \dots, \langle b_{\beta'}, kp_{\beta'} \rangle \},$$

$$\{ a_1, \dots, a_{\alpha} \},$$

where

- i) $a_1, \dots, a_{\alpha}$ and $b_1, \dots, b_{\beta'}$ are unique A - and B -values, respectively,
 - ii) kp_i is a vector of N address occurrences where N is the maximum number of A -values related to a B -value in R ,
 - iii) If b_i is related to no A -value in R , then all N elements of kp_i contain null address values, otherwise for $k = 1, \dots, t_i'$, (where t_i' is the number of A -values related to b_i in R) the k -th element of kp_i contains the address of the record in which the k -th A -value related to b_i in R is stored,
 - iv) kp_i is stored in the same record in which b_i is stored,
- $i = 1, \dots, \beta'$.

Implementation 23 for $R(A, B)$ is the converse of implementation 22 in the sense that in this case pointer vectors kp_i are stored in records that contain B -values, and they point to records in which related A -values are stored.

Implementations 22 and 23 are very similar in the way they actually establish a data base relation to implementations 17 and 19, respectively, defined in 3.1.9. The important difference is in the fact that in pointer array associations (implementations 17 and 19) data base set types will be defined which will establish the relation. This has the implication that pointers used in the implementation will be maintained by the data base management system in the case of associations, rather than the user in the case of linkages. Clearly there is a difference between the two cases as far as user convenience is concerned which we will not be able to capture in our evaluations of implementation alternatives. However a simple modification to our implementation of the design methodology may be made to allow the designer to prevent the selection of certain implementations for certain relations.

The last implementation alternative that we include in our model is implementation 24 called the double linkage of A and B and it consists of the two following sets of ordered pairs.

$$\{ \langle a_1, kp_1 \rangle, \langle a_2, kp_2 \rangle, \dots, \langle a_\alpha, kp_\alpha \rangle \},$$

$$\{ \langle b_1, kp'_1 \rangle, \langle b_2, kp'_2 \rangle, \dots, \langle b_\beta, kp'_\beta \rangle \},$$

where

- i) $a_1, \dots, a_\alpha$ and $b_1, \dots, b_\beta$ are unique A- and B-values, respectively,
- ii) kp_i is a vector of M address occurrences where

M is the maximum number of B-values related to an A-value in R ,

- iii) if a_i is related to no B-value in R , then all M elements of kp_i contain null address values, otherwise for $k' = 1, \dots, t_i$ (where t_i is the number of B-values related to a_i in R) the k' -th element of kp_i contains the address of the record in which the k' -th B-value related to a_i in R is stored,
- iv) kp'_k is a vector of N address occurrences where N is the maximum number of A-values related to a B-value in R ,
- v) if b_k is related to no A-value in R , then all N elements of kp'_j contain null address values, otherwise for $k' = 1, \dots, t'_k$ (where t'_k is the number of A-values related to b_k in R) the k' -th element of kp'_k contains the address of the record in which the k' -th A-value related to b_k in R is stored,
- vi) kp_i and kp'_k are stored in the same records in which a_i and b_k are stored, respectively,
 $i = 1, \dots, \alpha'$ and $k = 1, \dots, \beta'$.

Implementation 24 is a combination of implementations 22 and 23. It establishes the relation $R(A, B)$ in the data base by storing vectors of pointers (data base keys) in records. The record in which a_i ($i = 1, \dots, \alpha'$) is stored contains a fixed-length vector kp_i of M pointers. On the average r/α' elements of kp_i contain addresses of

records in which B-values related to a_i in R are stored. The rest of the elements of kp_i contain null address values. Also the record in which b_k ($k = 1, \dots, \beta'$) is stored contains a fixed-length vector kp'_k of N pointers. On the average r/β' elements of kp'_k contain addresses of records in which A-values related to b_k in R are stored and the rest of the elements contain null address values (Figure 3.17).

This concludes the definitions of the set of 24 implementation alternatives in our model. In Section 3.2 we present a set of constraint conditions for each implementation and then in Section 3.3 and 3.4 we present storage and time costs for each implementation.

It is the combination of exactly one implementation for every relation that determines the structure of the data base. The constraint conditions narrow down the number of alternatives for implementing a relation to those that produce legal DBTG structures and then the optimization algorithm selects (a reasonably) optimal solution namely one with minimum time cost subject to a storage space limit.

3.2 Constraint Conditions

In this section we develop a set of conditions for every implementation alternative. These conditions must be satisfied by the variables that define a data base relation in order that it can be implemented by the implementation

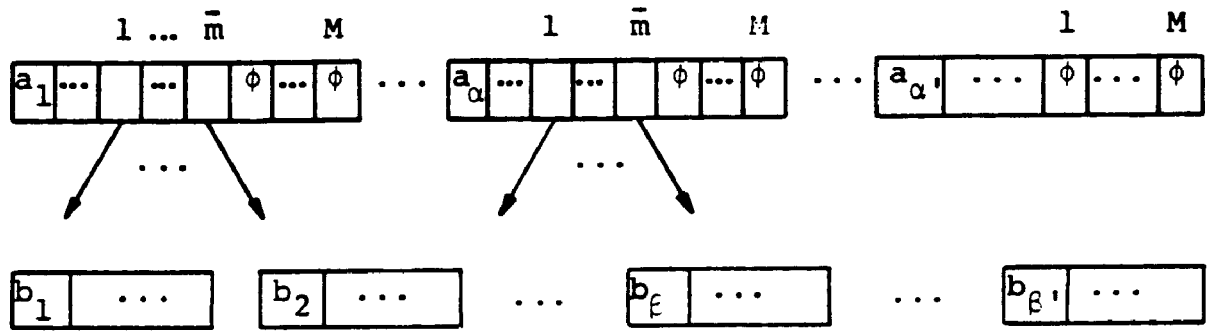


Figure 3.16 Single Linkage of B under A

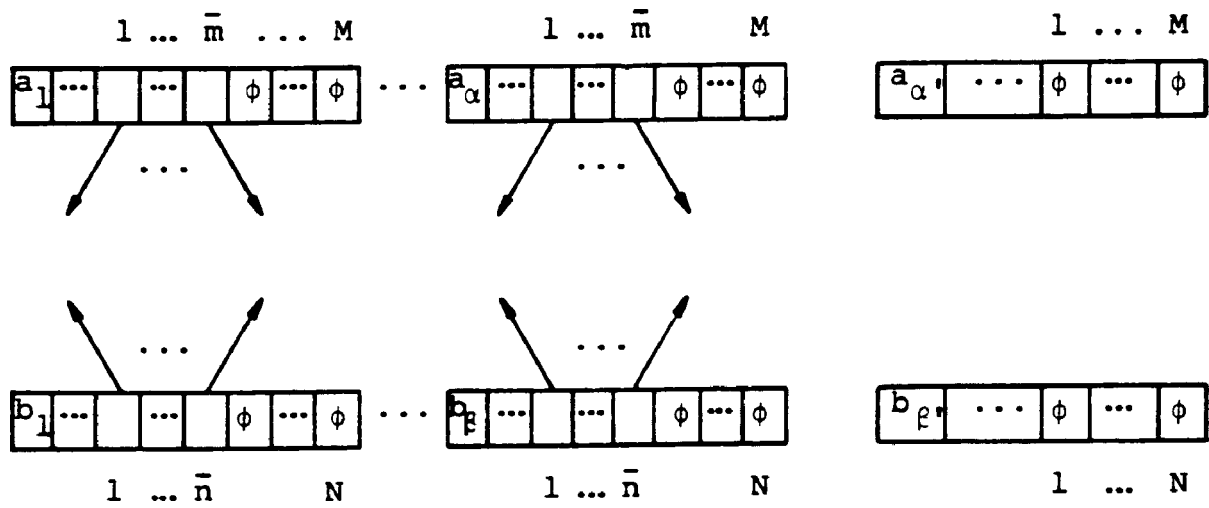


Figure 3.17 Double Linkage of A and B

to which the conditions pertain. The purpose of imposing these constraint conditions on relation implementations is to guarantee that the resulting data structure definitions stay within the limitations of DBTG specifications. In addition to that, certain security requirements of run units affect the choice of an implementation for a relation and this will also be reflected in the constraint conditions. For example, the non-redundancy convention of DBTG proposal requires that a data relationship be stored only once in the data base. This implies that exactly one implementation must be selected for each relation, and in a relation implementation each element (ordered pair) of the relation be represented (stored) only once. Another DBTG convention requires that variable dimensioned vectors or repeating groups be allowed only at level 1 of a record definition.

In order to increase security we will add constraint conditions that will prevent the implementation of a relation by duplication or aggregation when there is a run unit which must be denied access to the relation while it should be able to access individual data item types of the relation.

The constraint conditions will also reduce the size of solution space by rejecting clearly costly and/or inconsistent choices of implementations for a given relation. We now proceed with presenting, for each implementation number MPL , the constraint conditions $CC(MPL, c)$ that a relation R_j must satisfy so that it can be implemented by

implementation number MPL. (c is an integer representing the condition number.)

We first define a set of variables $AC(MPL, j)$, one for each $\langle \text{implementation}, \text{relation} \rangle$ pair. The value of this variable is 1 if implementation MPL is acceptable for relation R_j and zero otherwise. These variables will be expressed in terms of logical expressions involving constraint conditions such as $CC(MPL, c)$.

3.2.1 Implementations 1 and 2

Implementation 1 for $R_j(A, B)$ which is the fixed duplication of B under A is always acceptable except if there is a run unit RU_k such that $ORG(R_j) \in PSI_k$ and $DST(R_j) \in PSI_k$ but $R_j \notin PSR_k$. (PSI_k and PSR_k were defined in Chapter II.)

$$\begin{aligned}
 CC(1, 1) &: \forall k \in \{1, 2, \dots, NU\} (ORG(R_j) \in PSI_k \ \& \\
 &\quad DST(R_j) \in PSI_k \Rightarrow R_j \in PSR_k) \\
 AC(1, j) &= 1 \text{ if } CC(1, 1) \text{ is true} \\
 &= 0 \text{ otherwise}
 \end{aligned}$$

Similarly implementation 2 is acceptable for R_j if the above condition holds so $AC(2, j) = AC(1, j)$.

3.2.2 Implementations 3 and 4

Implementations 3 and 4 for R_j are variable duplications and hence they will be represented by variably dimensioned vectors or repeating groups. The DBTG limitation that permits variably dimensioned vectors or repeating groups only in level 1 requires the imposition of the

following constraint conditions in this case.

CC(3, 2) : $AC(5, i) = AC(7, i) = 0$ for all i

such that $DST(R_i) = ORG(R_j)$

CC(3, 3) : $AC(6, i) = AC(8, i) = 0$ for all $i \neq j$

such that $ORG(R_i) = ORG(R_j)$

These two conditions guarantee that $ORG(R_j)$ will not be aggregated under any other data item type.

Implementations 3 and 4 are among the implementation types which require that the two data item types of the relation be defined in one record type. Therefore, as in the previous case, if there is a run unit which should be given access to $ORG(R_j)$ and $DST(R_j)$ and at the same time be denied access to R_j then these two implementations (3 & 4) must be made unacceptable for R_j . In other words, the following condition must be true for R_j if it is to be implemented by implementation 3.

CC(3, 1) : $\forall k \in \{1, 2, \dots, NU\} (ORG(R_j) \in PSI_k \ \&$

$DST(R_j) \in PSI_k \Rightarrow R_j \in PSR_k)$

In summary, implementations 3 and 4 are acceptable for R_j if $AC(3, j) = 1$ and $AC(4, j) = 1$, respectively, where $AC(3, j)$ and $AC(4, j)$ are defined as follows.

CC(3, 1) = CC(1, 1) : $\forall k \in \{1, 2, \dots, NU\}$

$(ORG(R_j), DST(R_j) \in PSI_k$

$\Rightarrow R_j \in PSR_k)$

CC(3, 2) : $AC(5, i) = AC(7, i) = 0$ for all i such

that $DST(R_i) = ORG(R_j)$

$CC(3, 3) : AC(6, i) = AC(8, i) = 0$ for all $i \neq j$
 such that $ORG(R_i) = ORG(R_j)$
 $AC(3, j) = 1$ if $CC(3, c)$ is true for $c = 1, 2, 3$
 $= 0$ otherwise

Similarly

$CC(4, 1) = CC(1, 1)$
 $CC(4, 2) : AC(4, i) = AC(7, i) = 0$ for all $i \neq j$
 such that $DST(R_i) = DST(R_j)$
 $CC(4, 3) : AC(6, i) = AC(8, i) = 0$ for all i such
 that $ORG(R_i) = DST(R_j)$
 $AC(4, j) = 1$ if $CC(4, c)$ is true for $c = 1, 2, 3$
 $= 0$ otherwise

3.2.3 Constraint Conditions for Implementations 5 and 6

These two implementations are the fixed aggregations of B under A and A under B, respectively, for $R_j(A, B)$ and they were defined in 3.1.3.

Again since the two data item types of the relation, $ORG(R_j)$ and $DST(R_j)$, will be defined in one record type, we require that the following condition hold if R_j is to be implemented by, say, implementation 5.

$CC(5, 1) : \forall k \in \{1, 2, \dots, NU\} (ORG(R_j),$
 $DST(R_j) \in PSI_k \Rightarrow R_j \in PSR_k)$

In implementation 5 for $R_j(A, B)$, B-values related to a single A-value in R_j are stored in the record that the A-value is stored and these same stored occurrences of A and B values may be used to implement other relations on A

or B.

The use of same stored values of one data item type to implement multiple relations may cause a number of problems which we are trying to avoid by the imposition of constraint conditions. These problems and constraint conditions will be discussed below. But first we note that in the special case where $R_j \in \text{SINR}$, R_j is the only data base relation in which $\text{DST}(R_j)$ participates (either as origin or as destination data item type) and therefore no problems can arise. Implementation 5 for R_j is, then, acceptable if $\text{CC}(5, 1)$ and $\text{CC}(5, 2)$ both hold for R_j . $\text{CC}(5, 1)$ was given before and $\text{CC}(5, 2)$ is as follows.

$$\text{CC}(5, 2) : R_j \in \text{SINR}$$

When $\text{CC}(5, 2)$ is not true for R_j , i.e., when $\text{DST}(R_j)$ participates in other data base relations we require R_j to be of type 1, for the following reasons.

We recall that a fixed aggregation of B under A (implementation 5) uses fixed-sized vectors to store B-values under the related A-value. These vectors are of size M_j and if for some element a_1 of A the number of related B-values (in R_j) is less than M_j , say t , then $M_j - t$ null B-values must be stored in the vector. The problem arises from the fact that other implementations use the same B-values in the sense that other data items may be either aggregated or duplicated under these values which are in some instances only null values. This has the undesirable effect of producing large amounts of wasted

storage because all vectors repeated under null B-values contain (again) null values. In order to avoid this situation we require that $\bar{m}_j$ be equal to M_j . For the same reason every A-value must be related to at least one B-value that is m_j should be different from zero, which together with $\bar{m}_j = M_j$ implies:

$$M_j = \bar{m}_j = m_j.$$

On the other hand, every B-value must be related to at most one A-value. Again, since the same occurrences of B-values may be used to implement other relations, if the same B-values are repeated under more than one A-value then some pairs of those relations will be stored more than once in the data base in violation of the non-redundancy requirement. However, each B-value must be related to at least one A-value because otherwise it would be impossible to use the non-related B-values to implement other relations. Therefore the following must be true:

$$n_j = \bar{n}_j = N_j = 1.$$

This requirement together with $m_j = \bar{m}_j = M_j$ characterizes a type 1 relation and so we have our next constraint condition regarding implementation 5 for R_j .

CC(5, 3) : R_j is a type 1 relation.

The next and final constraint condition for using implementation 5 to implement R_j is intended to prevent aggregation of one data item type under more than one data item type. Implementation 5 for $R_j(A, B)$ aggregates B under A and it must be made acceptable only if A is the

only data item type under which B will be aggregated. This constraint is necessary because, by definition, the occurrences of B-values in implementation 5 are the only non-duplicate copies and any other occurrences of B-values in other parts of the data base are duplicate copies (results of implementations 1-4). If data item type B is the destination data item type of more than one relation, all of which are of type 1, then it is potentially possible to aggregate B under more than one data item type. In this case, we have to make a decision as to which one of the candidate relations must be allowed to be implemented using implementation 5. A reasonable decision is to select the relation that produces the smallest number of B-values per record. This results in records of shortest length over all other decisions.

The final constraint condition for implementation 5 is, then, given by the following.

$$\begin{aligned} \text{CC}(5, 4) : \quad & (M_j \leq M_i \text{ for all } i \neq j \text{ such that} \\ & \quad \text{DST}(R_i) = \text{DST}(R_j) \text{ and } R_i \text{ is of} \\ & \quad \text{type 1}) \\ & \text{and } (M_j \leq N_i \text{ for all } i \neq j \text{ such that} \\ & \quad \text{ORG}(R_i) = \text{DST}(R_j) \text{ and } R_i \text{ is of} \\ & \quad \text{type 2}) \end{aligned}$$

It is evident that this condition, while sufficient to prevent the aggregation of $\text{DST}(R_j)$ under more than one data item type, is stronger than necessary. This is because $\text{CC}(5, 4)$ rejects implementation 5 or 6 for some relations

R_i in favor of implementation 5 for R_j , while it is not known whether implementation 5 will actually be selected for R_j by the optimization algorithm. Although the fact that this condition is stronger than necessary has the side effect of excluding some potentially acceptable data base designs from the solution space, we need to adopt the condition as stated above because it allows the decision stages of the optimization algorithm to be independent of one another. The decision stages of the optimization algorithm must be independent in the sense that the choice of one implementation for a relation (a decision) must only affect other decisions (and consequently the solution) by the amounts of the storage and time costs of the relation implementation. It should not restrict (or be restricted by) other choices for other relations.

The complete set of constraint conditions for implementations 5 and 6 is, then, given as follows.

$$CC(5, 1) = CC(1, 1)$$

$$CC(5, 2) : R_j \in SINR$$

$$CC(5, 3) : R_j \in \{R_i \in SR \mid R_i \text{ is a type 1 relation}\}$$

$$CC(5, 4) : (M_j \leq M_i \text{ for all } i \neq j \text{ such that} \\ DST(R_i) = DST(R_j) \text{ and } R_i \text{ is of} \\ \text{type 1})$$

$$\text{and } (M_j \leq N_j \text{ for all } i \neq j \text{ such that} \\ ORG(R_i) = DST(R_j) \text{ and } R_i \text{ is of} \\ \text{type 2})$$

Implementation 5 for $R_j(A, B)$ is acceptable if $AC(5, j) = 1$,

where

$$\begin{aligned} AC(5, j) &= 1 \text{ if } CC(5, 1) \wedge (CC(5, 2) \vee (CC(5, 3) \wedge \\ &\quad \wedge CC(5, 4))) \\ &= 0 \text{ otherwise.} \end{aligned}$$

Similarly, constraint conditions for implementation 6 are:

$$CC(6, 1) = CC(1, 1)$$

$$CC(6, 2) : R_j \in \text{SONR}$$

$$CC(6, 3) : R_j \in \{R_i \in \text{SR} \mid R_i \text{ is a type 2 relation}\}$$

$$\begin{aligned} CC(6, 4) : (N_j \leq N_i \text{ for all } i \neq j \text{ such that} \\ \text{ORG}(R_i) = \text{ORG}(R_j) \text{ and } R_i \text{ is of} \\ \text{type 2}) \end{aligned}$$

$$\begin{aligned} \text{and } (N_j \leq M_i \text{ for all } i \neq j \text{ such that} \\ \text{DST}(R_j) = \text{ORG}(R_i) \text{ and } R_i \text{ is of} \\ \text{type 1}). \end{aligned}$$

Implementation 6 for $R_j(A, B)$ is acceptable if $AC(6, j) = 1$ where,

$$\begin{aligned} AC(6, j) &= 1 \text{ if } CC(6, 1) \wedge (CC(6, 2) \vee (CC(5, 3) \wedge \\ &\quad \wedge CC(5, 4))) \\ &= 0 \text{ otherwise.} \end{aligned}$$

3.2.4 Constraint Conditions for Implementations 7 and 8

Implementations 7 and 8 are similar to 5 and 6, respectively, with the difference that in 7 and 8 we are dealing with variably dimensioned vectors or repeating groups. For example, implementation 7 for $R_j(A, B)$ stores vectors of variable lengths (average $\bar{m}_j$) of B-values under the related A-values in the record in which the later is stored. In that respect constraint conditions for 7 and 8

are rather similar to those for 5 and 6 respectively, except that we have to make sure that the variably dimensioned vectors of the implementation do not occur at levels higher than 1 in a record definition. In other words, we have to make implementation 7 unacceptable for R_j if $ORG(R_j)$ could be aggregated (fixed or variable) under any other data item type to implement another relation. We now proceed with the definition of the constraint conditions regarding implementation 7 for $R_j(A, B)$. Similar conditions will be presented for implementation 8 but not discussed in detail.

Like all other duplications and aggregations, the first constraint condition deals with the security requirements. This constraint condition prevents the use of implementation 7 for $R_j(A, B)$ if there exists a run unit which must be denied access to R_j while it should be able to access A and B (through other relations). The first constraint condition, then, for implementation 7 is the following.

$$CC(7, 1) : \forall k \in \{1, 2, \dots, NU\} (ORG(R_j), \\ DST(R_j) \in PSI_k \Rightarrow R_j \in PSR_k)$$

The next two conditions are intended to guarantee that the variably dimensioned vectors of this implementation occur only at level 1 of a record type. There are two ways that the variably dimensioned vector of A-values ($A = ORG(R_j)$) can end up at levels higher than one, 1) if A is also the destination data item type of another relation

$R_j \in SR$, for which implementation 5 or 7 are acceptable and 2) if A is also the origin data item type of another relation $R_j \in SR$, for which implementation 6 or 7 are acceptable. The first situation is avoided by requiring the following condition.

$$CC(7, 2) : R_j \text{ is of type 1 or 3 for no } i (i \neq j) \\ \text{such that } DST(R_i) = ORG(R_j)$$

As will be seen later in this section one of the constraint conditions for implementations 5 and 7 for a relation is that the relation must be of types 1 and 3 respectively. The requirement that A should not participate as destination in any relation of SR which is of type 1 excludes the possibility of aggregation of A under any other data item type. This becomes clear by looking at CC(5, 3) in the last section. Note, however, that $CC(5, 3) = \text{FALSE}$ is sufficient to make $AC(5, j) = 0$ because $CC(5, 2)$ is true only if $A \in SINR$ which is impossible since $A = ORG(R_j)$. Similarly if R_i is of type 3 for any relation R_i such that $DST(R_i) = ORG(R_j)$ then implementation 7 is unacceptable for all relations in which A participates as destination.

Similarly we can show that $CC(7, 3)$ defined below is sufficient to insure that implementations 6 and 8 are unacceptable for all relations in which A participates as origin. The two conditions $CC(7, 2)$ and $CC(7, 3)$ will then guarantee that $A = ORG(R_j)$ will not be aggregated under any other data item type.

CC(7, 3) : R_i is of type 2 or 4 for no $i \neq j$ such
that $ORG(R_i) = ORG(R_j)$.

Again, these two conditions, CC(7, 2) and CC(7, 3) are only sufficient conditions. They make implementation 7 unacceptable for a relation R_j if $ORG(R_j)$ could possibly be aggregated under any data item type irrespective of whether it actually will be. We have to adopt these conditions, however, for the same reason as discussed in Section 3.2.3 in the case of constraint condition CC(5, 4).

The next constraint condition for implementation 7 is similar to CC(5, 2) discussed in 4.3. Again if no other data base relation is defined over $B = DST(R_j)$, i.e., if $R_j \in SINR$, then no difficulty can arise by aggregating B under A for R_j and hence if the following is true we make implementation 7 acceptable for R_j regardless of the type of the relation (of course CC(7, 3) and CC(7, 2) must be true).

CC(7, 4) : $R_j \in SINR$

However, if the above constraint condition is not satisfied the following conditions should hold so that no inconsistencies would result by aggregating B under A . For the same arguments as described in 3.2.3 which led to CC(5, 3) we can show that the following should hold for R_j .

$$N_j = \bar{n}_j = n_j = 1$$

However, since implementation 7 is a variable aggregation (of B under A) and it therefore does not have data item occurrences that contain null B -values, we no longer need

$M_j = \bar{m}_j = m_j$. This means that R_j can be either of type 1 or of type 3, and so we have the following.

CC(7, 5) : $R_j \in \{R_i \in SR \mid R_i \text{ is of type 1 or type 3}\}$

The remaining conditions are intended to guarantee that $B = \text{DST}(R_j)$ will be aggregated (fixed or variable) under at most one other data item type. The reason behind this requirement was explained in 3.2.3 where we introduced the constraint condition CC(5, 4). In order to accomplish this we make implementation 7 unacceptable for $R_j(A, B)$ if implementation 5 is acceptable for any $R_i \in SR$ for which $\text{DST}(R_i) = \text{DST}(R_j)$ or if implementation 6 is acceptable for any $R_i \in SR$ for which $\text{ORG}(R_i) = \text{DST}(R_j)$. This makes implementation 7 unacceptable for R_j if $\text{DST}(R_j)$ is (fixed) aggregated under any data item type in SI. And again for the same reasons as described in 3.2.3 (for implementation 5), if $\text{DST}(R_j)$ can be variably aggregated under more than one other data item types then we choose the one with minimum average repetition factor $\bar{m}_j$ (or $\bar{n}_j$ whichever applies).

In other words, we have the following two additional constraint conditions for implementation 7.

CC(7, 6) : R_i is not of type 1 for any $R_i \in SR$

($i \neq j$) such that $\text{DST}(R_i) = \text{DST}(R_j)$ and
 R_i is not of type 2 for any $R_i \in SR$
 such that $\text{ORG}(R_i) = \text{DST}(R_j)$

$CC(7, 7) : \bar{m}_j < \bar{m}_i$ for all $i \neq j$ such that
 $DST(R_i) = DST(R_j)$ and R_i is of
 type 3 and
 $\bar{m}_j < \bar{n}_i$ for all i such that
 $ORG(R_i) = DST(R_j)$ and R_i is of
 type 4.

Constraint conditions $CC(7, 5)$ to $CC(7, 7)$ are also only sufficient conditions.

The complete set of constraint conditions for implementations 7 and 8 for R_j is, then, as follows.

$CC(7, 1) = CC(1, 1)$

$CC(7, 2) : R_i$ is of type 1 or 3 for no i such
 that $DST(R_i) = ORG(R_j)$

$CC(7, 3) : R_i$ is of type 2 or 4 for no $i \neq j$ such
 that $ORG(R_i) = ORG(R_j)$

$CC(7, 4) : R_j \in SINR$

$CC(7, 5) : R_j \in \{R_i \in SR \mid R_i \text{ is type 1 or type 3}\}$

$CC(7, 6) : \text{as above}$

$CC(7, 7) : \text{as above}$

Implementation 7 for $R_j(A, B)$ is, then, acceptable if

$AC(7, j) = 1$ where:

$AC(7, j) = 1$ if $CC(7, 1) \wedge CC(7, 2) \wedge CC(7, 3) \wedge$
 $\wedge (CC(7, 4) \vee (CC(7, 5) \wedge CC(7, 6) \wedge$
 $\wedge CC(7, 7)))$
 $= 0$ otherwise.

Similarly, constraint conditions for implementation 9 are as follows.

$$CC(8, 1) = CC(1, 1)$$

$$CC(8, 2) : R_i \text{ is of type 1 or 3 for no } i \neq j \text{ such} \\ \text{that } DST(R_i) = DST(R_j)$$

$$CC(8, 3) : R_i \text{ is of type 2 or 4 for no } i \text{ such that} \\ ORG(R_i) = DST(R_j)$$

$$CC(8, 4) : R_j \in SONR$$

$$CC(8, 5) : R_j \in \{R_i \in SR \mid R_i \text{ is of type 2 or type 4}\}$$

$$CC(8, 6) : R_i \text{ is not of type 1 for any } R_i \in SR \text{ such} \\ \text{that } DST(R_i) = ORG(R_j) \text{ and} \\ R_i \text{ is not of type 2 for any } R_i \in SR (i \neq j)$$

$$\text{such that } ORG(R_i) = ORG(R_j)$$

$$CC(8, 7) \quad \bar{n}_j \leq \bar{n}_i \text{ for all } i \neq j \text{ such that} \\ ORG(R_i) = ORG(R_j) \text{ and} \\ \bar{n}_j \leq \bar{m}_i \text{ for all } i \text{ such that} \\ DST(R_i) = ORG(R_j).$$

Implementation 8 for $R_j(A, B)$ is, then, acceptable if

$AC(8, j) = 1$ where:

$$AC(8, j) = 1 \text{ if } CC(8, 1) \wedge CC(8, 2) \wedge CC(8, 3) \wedge \\ \wedge (CC(8, 4) \vee (CC(8, 5) \wedge \\ \wedge CC(8, 6) \wedge CC(8, 7))), \\ = 0 \text{ otherwise.}$$

3.2.5 Constraint Conditions for Implementations 9 through 20

Implementations 9 through 20 are all associations of different types for $R_j(A, B)$. The odd numbered implementations are associations of B under A and the even numbered

implementations are associations of A under B. The constraint conditions for implementations 9, 11, 13, 15, 17 and 19 are identical, so are the constraint conditions for implementations 10, 12, 14, 16, 18 and 20. We first develop constraint conditions for implementation 9 in detail and use them for implementations 11, 13, 15, 17 and 19 and then we will derive constraint conditions for implementations 10, 12, ... , 20.

Implementation 9 is the chain with next pointers association of B under A for $R_j(A, B)$. In that sense, as defined in 3.1.5, a data base set type will be defined where owner records will contain unique A-values and member records will contain unique B-values. The uniqueness requirement of A- and B-values in owner and member records together with the fact that implementation 9 does not use duplicate copies of A- and B-values require that data item types A and B of R_j should not be aggregated under any other data item type. The following constraint condition is sufficient to insure that neither A (= ORG(R_j)) nor B(= DST(R_j)) will be aggregated (fixed or variable) under any other data item type.

$$\begin{aligned}
 \text{CC}(9, 1) : \quad & \text{AC}(5, i) = \text{AC}(7, i) = 0 \text{ for all } i \neq j \\
 & \text{such that } \text{DST}(R_i) \in \{\text{ORG}(R_j), \\
 & \quad \text{DST}(R_j)\} \\
 & \text{AC}(6, i) = \text{AC}(7, i) = 0 \text{ for all } i \neq j \\
 & \text{such that } \text{ORG}(R_i) \in \{\text{ORG}(R_j), \\
 & \quad \text{DST}(R_j)\}
 \end{aligned}$$

The next constraint condition of implementation 9 for $R_j(A, B)$ is due to a well known DBTG requirement about data base set types. In a DBTG set type a single record cannot participate in more than one occurrence of the same set type. This has the implication that a record may not be member in (or owner of) more than one occurrences of the same set type, or even a member in one occurrence and owner in another. In other words, DBTG set types can only implement one-to-many relationships. This brings us to the next constraint condition for implementation 9 as follows.

$CC(9, 2) : R_j \in \{R_i \in SR \mid R_i \text{ is of type 5, 3 or 1}\}$

Note that in a type 5 relation, a B-value is related to at most one A-value but an A-value can be related to any number of B-values (including zero).

The constraint conditions for implementations 9 through 20 for $R_j(A, B)$ are, then, as follows.

For $MPL = 9, 11, 13, \dots, 19$;

$CC(MPL, 1) : AC(5, i) = AC(7, i) = 0$ for all $i \neq j$
 such that $DST(R_i) \in \{ORG(R_j),$
 $DST(R_j)\}$

$AC(6, i) = AC(8, i) = 0$ for all $i \neq j$
 such that $ORG(R_i) \in \{ORG(R_j),$
 $DST(R_j)\}$

$CC(MPL, 2) : R_j \in \{R_i \in SR \mid R_i \text{ is of type 5, 3 or 1}\}$

For $MPL = 10, 12, 14, \dots, 20$;

$CC(MPL, 1) : AC(5, i) = AC(7, i) = 0$ for all $i \neq j$
 such that $DST(R_i) \in \{ORG(R_j),$
 $DST(R_j)\}$

$AC(5, i) = AC(8, i) = 0$ for all $i \neq j$
 such that $ORG(R_j) \in \{ORG(R_j),$
 $DST(R_j)\}$

$CC(MPL, 2) : R_j \in \{R_i \in SR \mid R_i \text{ is of type 6, 4 or 2}\}.$

Implementation $MPL \in \{9, 10, 11, \dots, 20\}$ is, then, acceptable for $R_j(A, B)$ if $AC(MPL, j) = 1$ where;

$AC(MPL, j) = 1$ if $CC(MPL, 1)$ and $CC(MPL, 2)$ are both
 true

= 0 otherwise.

3.2.6 Constraint Conditions for Implementations 21, 22, 23 and 24

Implementations 21, 22, 23 and 24 have identical constraint conditions. These implementations were defined in 3.1.11 and 3.1.12. Implementation 21 for $R_j(A, B)$ is the dummy record association of A and B and the other three implementations were defined as linkages. These implementations are similar to other associations with the difference that implementation 21 uses two data base sets to establish relation $R_j(A, B)$ and implementations 22, 23 and 24 do not use data base sets at all. This makes it possible to use any one of the above four implementations to implement a many-to-many relation. Individual set types

of implementation 21 still have to satisfy the DBTG convention that the relationship between owners and members of data base sets (of one type) must be a one-to-many relationship. But the combination of two set types each of which satisfies the DBTG convention can implement a many-to-many relationship in a manner discussed in 3.1.11.

The effect of the observation made above on the constraint conditions for implementations 21, ... , 24 for $R_j(A, B)$ is that R_j no longer has to be of type 5 or 6 therefore a constraint condition such as $CC(MPL, 2)$ of 3.2.5 is no longer necessary. However, A- and B-values in the records related by any of the implementations should still be unique values and hence the (only) constraint condition for implementations 21, 22, 23 and 24 is similar to $CC(MPL, 1)$ described in 3.2.5 and it is given below.

$$\begin{aligned}
 CC(MPL, 1) : \quad & AC(5, i) = AC(7, i) = 0 \text{ for all } i \neq j \\
 & \quad \text{such that } DST(R_i) \in \{ORG(R_i), \\
 & \quad \quad DST(R_j)\} \text{ and} \\
 & AC(6, i) = AC(8, i) = 0 \text{ for all } i \neq j \\
 & \quad \text{such that } ORG(R_i) \in \{ORG(R_j), \\
 & \quad \quad DST(R_j)\}.
 \end{aligned}$$

Accordingly implementation MPL is acceptable for R_j if $AC(MPL, j) = 1$ where for $MPL = 21, 22, 23$ and 24 :

$$\begin{aligned}
 AC(MPL, j) &= 1 \quad \text{if } CC(MPL, 1) \text{ is true} \\
 &= 0 \text{ otherwise.}
 \end{aligned}$$

This concludes the discussion of constraint conditions for all implementation alternatives. It is evident that

some of these constraint conditions depend on the acceptability of other relation implementations. Therefore the acceptability of some implementations for a relation should be determined before others. In particular, the acceptability of aggregations (implementations 5, 6, 7 and 8) must be determined before all other implementations except the fixed duplications (implementations 1 and 2). This is because the variables $AC(5, j)$, $AC(6, j)$, $AC(7, j)$ and $AC(8, j)$ that record the acceptability of implementations 5, 6, 7 and 8, respectively, for relation R_j appear in the definitions of the constraint conditions for all other implementations for R_j (except implementations 1 and 2 for R_j).

Therefore we first determine $AC(MPL, j)$ for $MPL = 1, 2$, then we continue with $MPL \in \{5, 6, 7, 8\}$ and then we can determine $AC(MPL, j)$ for all other implementations namely for $MPL \in \{3, 4, 9, 10, \dots, 24\}$. For each MPL we determine $AC(MPL, j)$ for all $j \in \{1, \dots, NR\}$. We note, however, that the order in which relations are considered does not have any effect on the results of $AC(MPL, j)$ evaluations.

3.3 Storage Costs

In this section we will define the storage cost of each implementation alternative defined in section 3.1. The storage costs are measured in the number of characters which is the same unit used to define sizes of data items,

address occurrences (pointers), page sizes, etc. The storage space requirement of a relation implementation is the amount of storage space needed to store data item occurrences, pointers, counter occurrences and privacy locks as required by the implementation to establish the relation in the data base.

It is evident that one occurrence of each data item value of every type must be stored somewhere in the data base. There are NI data item types in SI . Size and cardinality of $I_i \in SI$, for $i = 1, 2, \dots, NI$ are given by $ITSZ(i)$ and $ITCR(i)$, respectively. The total storage space required to store one occurrence of each data item value of every type is denoted by $MMSB$ and given by the following sum.

$$MMSB = \sum_{i=1}^{NI} ITSZ(i) * ITCR(i)$$

The storage cost of a relation implementation is defined to be the amount of storage space required to implement the relation beyond the amount of storage space required to store one occurrence of each data item value of the relation's component data item types. For example, the total storage requirement of implementation 1 for relation $R_j(A, B)$ is $\alpha'_j \alpha''_j + \beta'_j \beta''_j + \alpha'_j * M_j * \beta''_j$. In this expression the first and the second terms give the amount of storage required to store one occurrence of each data item value of types A and B , respectively. The third term is the storage space needed to store duplicate B -values as

required by implementation 1 (see Section 3.1.1). The storage cost of implementation 1 for $R_j(A, B)$ is $\alpha_j' * M_j * \beta_j''$, and the first two terms are accounted for in MMSB and therefore deleted from storage costs.

The amount of storage left for optimization is then given by OMSB where:

$$\text{OMSB} = \text{TMSB} - \text{MMSB}$$

The storage required for privacy locks is calculated on the assumption that for every relation one lock will be defined in the schema for each run unit that uses the relation for every appropriate access type. If the implementation of the relation R_j requires the definition of a set type and RUUR_j is the number of run units that use relation R_j then $\text{NATS} * \text{RUUR}_j$ locks must be defined where NATS is the number of access types to a DBTG set. For a lock size of PLOKS characters, then, this would require a total of $\text{TLS}_j = \text{PLOKS} * \text{NATS} * \text{RUUR}_j$ characters of storage for privacy locks. This formula can be used for implementations 9 through 20. Implementation 21 for R_j uses two data base sets and privacy locks are defined for the two data base sets separately at a total storage requirement of $2 * \text{TLS}_j$ characters. Implementations 22 and 23 for R_j (defined in 3.1.12) require the definition of a data item type that contains data base key values. The protection of the relation in this case is achieved through the privacy locks defined for this data item. The total storage required for privacy locks in case of implementations 22 and 23 for R_j

is then given by

$$TLD_j = PLOKS * NATD * RUUR_j$$

where NATD is the number of access types to a data item in a DBTG system and PLOKS and $RUUR_j$ are defined above. Similarly implementation 24 for R_j requires $2 * TLD_j$ characters for privacy locks, because in this implementation two data items should be defined in two different record types. We will also use the two variables TLS_j and TLD_j in derivations of storage costs in this section without further explanation of their expressions.

The number of access types to a data base set and a data item (NATS and NATD), respectively, are the numbers of different DML commands that a particular DBTG implementation provides for data base sets and data items for which privacy locks may be defined. Privacy locks may, for example, be specified which apply to the use of a DBTG set type for DML commands: FIND, ORDER, INSERT and REMOVE and which apply to the member record type of a set type for DML commands: FIND, INSERT and REMOVE. Privacy locks may be specified which apply to data item types for DML commands: STORE, MODIFY and ERASE. There are other privacy locks which apply to the uses of record types, schema, sub-schema, realms, etc. which we are not taking into consideration either because we are not modelling the underlying concepts (such as privacy locks for realms) or because they are irrelevant to the optimization problem (such as privacy locks for record types). Privacy locks on data item types

in SI are also not considered because their storage requirements would be the same for all implementations.

3.3.1 Storage Costs of Fixed Duplications

Storage cost of implementation 1 for $R_j(A, B)$ which is the fixed duplication of B under A (defined in 3.1.1) is the space required to store fixed sized vectors of B-values that we called $fd(B)$ in 3.1.1. The number of these vectors is α'_j , each one has M_j elements of size β''_j hence the total storage cost of implementation MPL = 1 for relation R_j is given by:

$$SC(1, j) = \alpha'_j * M_j * \beta''_j.$$

Similarly, the storage cost of implementation 2 for $R_j(A, B)$ is given by:

$$SC(2, j) = \beta'_j * N_j * \alpha''_j.$$

3.3.2 Storage Costs of Variable Duplications

Implementations 3 and 4 for $R_j(A, B)$ were defined in 3.1.2 as variable duplications (of B under A and A under B), respectively). The storage cost of implementation 3 for $R_j(A, B)$ is the space required to store the counter occurrences c_i and variable-size vectors $vd_i(B)$ of duplicate B-values, $i = 1, \dots, \alpha'_j$. The average number of elements of $vd_i(B)$ over $i \in \{1, \dots, \alpha'_j\}$ is r_j/α'_j . The size of each element is β''_j characters. Therefore the storage cost of implementation 3 for $R_j(A, B)$ is given by:

$$\begin{aligned} SC(3, j) &= \alpha'_j * (r_j/\alpha'_j) * \beta''_j + \alpha'_j * CNTRS \\ &= r_j * \beta''_j + \alpha'_j * CNTRS \end{aligned}$$

where CNTRS is the size of c_i .

Similarly the storage cost of implementation 4 for $R_j(A, B)$ is given by the following.

$$SC(4, j) = r_j * \alpha_j'' + \beta_j' * CNTRS.$$

3.3.3 Storage Costs of Fixed Aggregations

The storage cost of implementation 5 for $R_j(A, B)$, defined as the fixed aggregation of B under A, is the space required to store $fa_i(B)$, $i = 1, \dots, \alpha_j'$ vectors defined in 3.1.3. However, since the occurrences of B-values in these vectors are shared occurrences, the space required to store one occurrence of each B-value, $\beta_j' * \beta_j''$, is accounted for in MMSB. The storage cost of implementation 5 is then the difference between these two storage space requirements:

$$SC(5, j) = \alpha_j' * M_j * \beta_j'' - \beta_j' * \beta_j'' = (\alpha_j' * M_j - \beta_j') * \beta_j''$$

Similarly storage cost of implementation 6 for $R_j(A, B)$ is:

$$SC(6, j) = (\beta_j' * N_j - \alpha_j') * \alpha_j''.$$

If relation $R_j(A, B)$ is of type 1 then $\beta_j' = \alpha_j' * M_j$, in which case $SC(5, j) = 0$. Similarly $SC(6, j) = 0$ if $R_j(A, B)$ is of type 2.

3.3.4 Storage Costs of Variable Aggregations

Implementations 7 and 8 for $R_j(A, B)$ were defined, in 3.1.4, as variable aggregations (of B under A and A under B, respectively). The storage cost of implementation 7 for $R_j(A, B)$ is the space required to store the counter occurrences c_i and variable-size vectors $va_i(B)$ of B-values,

$i = 1, \dots, \alpha'_j$. Average size (number of elements) of $va_i(B)$ over $i = 1, \dots, \alpha'_j$ is r_j/α'_j . The total space requirement of $va_i(B)$ vectors (in characters) is, then,

$$\alpha'_j * (r_j/\alpha'_j) * \beta''_j = r_j * \beta''_j.$$

However, as in the case of implementation 5, since occurrences of B-values in $va_i(B)$ vectors are shared occurrences. They are accounted for in MMSB and therefore we must subtract from the above value, the storage required to store one occurrence of each B-value which is equal to $\beta'_j * \beta''_j$. The total space requirement of c_i counters is $\alpha'_j * \text{CNTRS}$. Hence the storage cost of implementation 7 for $R_j(A, B)$ is:

$$\text{SC}(7, j) = (r_j - \beta'_j) * \beta''_j + \alpha'_j * \text{CNTRS}.$$

Similarly, the storage cost of implementation 8 for $R_j(A, B)$ is as follows,

$$\text{SC}(8, j) = (r_j - \alpha'_j) * \alpha''_j + \beta'_j * \text{CNTRS}.$$

Note that if R_j is of type 3 (4) then $r_j = \beta'_j$ ($r_j = \alpha'_j$).

3.3.5 Storage Costs of Chain Associations

Implementations 9 through 16 were defined in 3.1.5 through 3.1.8 as different types of chain associations. They all use a single data base set type to implement a relation. In the derivations of storage cost implementations 9-16, which follows, PTRS denotes the size of a data-base-key (pointer size) and TLS_j is defined at beginning of this section.

The storage cost of implementation 9 for $R_j(A, B)$ called chain with next pointers association of B under A

consists of storage required for mp_i and np_k pointers (see 3.1.5) for $i = 1, \dots, \alpha'_j$ and $k = 1, \dots, \beta'_j$. Therefore with a pointer size of PTRS and privacy lock requirement of TLS_j , the storage cost of implementation 9 for $R_j(A, B)$ is given by:

$$\begin{aligned} SC(9, j) &= \alpha'_j * PTRS + \beta'_j * PTRS + TLS_j \\ &= (\alpha'_j + \beta'_j) * PTRS + TLS_j \end{aligned}$$

Similarly the storage cost of implementation 10 for $R_j(A, B)$ is given by:

$$SC(10, j) = (\alpha'_j + \beta'_j) * PTRS + TLS_j$$

For implementation 11 (for $R_j(A, B)$) which is the chain with next and owner pointers association of B under A, defined in 3.1.6, the storage cost is due to the storage space requirement of the three sets of pointers mp_i , np_k and op_k ($i = 1, \dots, \alpha'_j$ and $k = 1, \dots, \beta'_j$). The storage requirement of mp_i , np_k and op_k pointers are $\alpha'_j * PTRS$, $\beta'_j * PTRS$ and $\beta'_j * PTRS$ characters, respectively. Hence the storage cost of implementation 11 for $R_j(A, B)$ is as follows.

$$SC(11, j) = (\alpha'_j + 2 * \beta'_j) * PTRS + TLS_j$$

Similarly the storage cost of implementation 12 for $R_j(A, B)$ is given by:

$$SC(12, j) = (2 * \alpha'_j + \beta'_j) * PTRS + TLS_j.$$

In the case of implementation 13 for $R_j(A, B)$, which is the chain with next and prior pointers association of B under A, defined in 3.1.7, there are four sets of pointers to consider. These sets of pointers were defined

as mp_i , lp_i , np_k and pp_k , $i = 1, \dots, \alpha'_j$, $k = 1, \dots, \beta'_j$. The storage requirements of these sets are $\alpha'_j * PTRS$, $\alpha'_j * PTRS$, $\beta'_j * PTRS$ and $\beta'_j * PTRS$, respectively. The storage cost of implementation 13 for $R_j(A, B)$ is, then, given by:

$$SC(13, j) = 2 * (\alpha'_j + \beta'_j) * PTRS + TLS_j.$$

Similarly the storage cost of implementation 14 for $R_j(A, B)$ is as follows.

$$SC(14, j) = 2 * (\alpha'_j + \beta'_j) * PTRS + TLS_j.$$

In evaluating the storage cost of implementation 15 for $R_j(A, B)$, defined in 3.1.8 there are five sets of pointers to be considered. The first two sets are mp_i and lp_i , $i = 1, \dots, \alpha'_j$ pointers in owner records and the next three are np_k , pp_k and op_k , $k = 1, \dots, \beta'_j$ pointers in member records. The total storage cost of implementation 15 for $R_j(A, B)$ is, then, given by:

$$SC(15, j) = (3 * \beta'_j + 2 * \alpha'_j) * PTRS + TLS_j$$

Similarly, storage cost of implementation 16 for $R_j(A, B)$ is:

$$SC(16, j) = (3 * \alpha'_j + 2 * \beta'_j) * PTRS + TLS_j.$$

3.3.6 Storage Costs of Pointer Array Associations

Implementations 17, 18, 19 and 20 were defined in 3.1.9 and 3.1.10 as pointer array associations. We will now present the storage costs of these four implementations.

In implementation 17 for $R_j(A, B)$, which is the pointer array association of B under A, the storage requirement for vp_i , $i = 1, \dots, \alpha'_j$ pointers is $\alpha'_j * PTRS$

and the pointer arrays pa_i , $i = 1, \dots, \alpha'_j$ in owner records of this implementation require $M_j * PTRS$ characters for each pa_i . The total storage cost of implementation 17 for $R_j(A, B)$ is, then, as follows.

$$SC(17, j) = (\alpha'_j * M_j + \alpha'_j) * PTRS + TLS_j$$

Similarly, the storage cost of implementation 18 for $R_j(A, B)$ is given by,

$$SC(18, j) = (\beta'_j * N_j + \beta'_j) * PTRS + TLS_j.$$

Implementation 19 for $R_j(A, B)$, called the pointer array with owner pointers association of B under A, in addition to the vp_i and pa_i , $i = 1, \dots, \alpha'_j$ pointers, includes the owner pointers, op_k , $k = 1, \dots, \beta'_j$ in its member records. The total storage cost of implementation 19 for $R_j(A, B)$ is then as follows.

$$SC(19, j) = (\alpha'_j * M_j + \alpha'_j + \beta'_j) * PTRS + TLS_j$$

Similarly the storage cost of implementation 20 for $R_j(A, B)$ is given by:

$$SC(20, j) = (\beta'_j * N_j + \beta'_j + \alpha'_j) * PTRS + TLS_j.$$

3.3.7 Storage Costs of Dummy Record Association and Linkages

Implementation 21 for $R_j(A, B)$ was defined in 3.1.11 as the dummy record association of A and B. In this implementation the two pointer sets mp_i and mp'_ℓ ($i = 1, \dots, \alpha'_j$ and $\ell = 1, \dots, \beta'_j$) require $\alpha'_j * PTRS$ and $\beta'_j * PTRS$ characters of storage, respectively. Each dummy record of this implementation requires $4 * PTRS$ characters and there are r_j such records. Also, since two data base set types are

defined for this implementation, the storage space required for privacy locks is $2 * TLS_j$ as discussed at the beginning of this section. The total storage cost of implementation 21 for $R_j(A, B)$ is, then, given by:

$$SC(21, j) = (\alpha'_j + \beta'_j + 4 * r_j) * PTRS + 2 * TLS_j.$$

Implementation 22 for $R_j(A, B)$, defined in 3.1.12, is the single linkage of B under A. The storage cost of this implementation consists of the space required to store kp_i vectors of pointers and TLD_j characters for privacy locks discussed at the beginning of this section. Each kp_i , $i = 1, \dots, \alpha'_j$ has M_j elements of size PTRS characters, so the total storage cost of implementation 22 for $R_j(A, B)$ is given by:

$$SC(22, j) = \alpha'_j * M_j * PTRS + TLD_j$$

Similarly the storage cost of implementation 23 and 24 for $R_j(A, B)$ (defined in 3.1.12) are given by the following expressions.

$$SC(23, j) = \beta'_j * N_j * PTRS + TLD_j$$

$$SC(24, j) = (\alpha'_j * M_j + \beta'_j * N_j) * PTRS + 2 * TLD_j.$$

3.4 Time Costs

Our basic assumption is that the data base, residing on secondary storage is accessed in a paging environment. Only a certain number of pages (of data) can be in main storage at any given time. We assume this number to be MSPG. When a record access is required, if the record is stored in a page that is currently in main storage then no

secondary storage access is required. The record, in this case, is simply moved from system buffers to the user working area according to the rules of the data base management systems. If, however, the accessed record is stored in a page that is not in main storage at the time of access, the entire page that contains the record must first be read from secondary storage into the system buffers before it can be delivered to the user program. For every record access, then, there is a probability that a (missing) page fault occurs and a secondary storage access is required. The expected rate of these secondary storage accesses is what we are trying to minimize.

In this section we derive the expressions for computing the expected rate of page faults for each relation implementation.

The time cost of implementation numbered MPL for relation R_j is defined to be the number of page faults expected to occur as a result of execution of run units specified in the user activities if implementation MPL is used for relation R_j . This value will be denoted by $TC(MPL, j)$ and it is defined only if implementation numbered MPL is acceptable for relation R_j , i.e., $AC(MPL, j) = 1$. We recall that SR was defined in Chapter 2 as the set of data base relations of the application $SR = \{R_1, R_2, \dots, R_{NR}\}$. A configuration C for a set of data base relations SR is a set of ordered pairs of relation implementations:

$$C = \{ \langle R_1, MPL_1 \rangle, \langle R_2, MPL_2 \rangle, \dots, \langle R_j, MPL_j \rangle, \dots, \\ \langle R_{NR}, MPL_{NR} \rangle \}$$

where MPL_j , $j = 1, \dots, NR$ is an implementation number (an integer from 1 to 24) designating one of the 24 implementation alternatives defined in Section 3.1. Note that in the definition of C , $R_i = R_j$ iff $i = j$, for $i, j \in \{1, 2, \dots, NR\}$, but MPL_i may be equal to MPL_j when $i \neq j$.

An acceptable configuration C for SR is defined as follows.

$$C = \{ \langle R_j, MPL_j \rangle \mid j = 1, \dots, NR \text{ and } AC(MPL_j, j) = 1 \\ \text{for all } j \}$$

The (cumulative) storage cost of an acceptable configuration C for SR is the sum of storage costs of individual relation implementations in the configuration and it is given by:

$$CS(c) = \sum_{j=1}^{NR} SC(MPL_j, j)$$

Similarly, the (cumulative) time cost of an acceptable configuration C is defined as follows:

$$CT(c) = \sum_{j=1}^{NR} TC(MPL_j, j)$$

This is the total time cost of a data base, measured in the expected number of page faults, and it is the sum of the time costs of exactly one acceptable implementation for each relation in SR (the set of relations defined in the application).

$CT(c)$ is the quantity that is subject to minimization, in the optimization algorithm of Chapter 4, over all acceptable configurations c for which $CS(c)$ is not greater than a given limit.

In order to compute the time cost of a relation implementation we have to accumulate the expected page fault rates of all operations (of all run units) in which the relation is used. Let O_ℓ^k be the ℓ -th operation of k -th run unit and let implementation MPL be acceptable for relation $R_j \in SR$, i.e., $AC(MPL, j) = 1$. We denote the expected number of page faults resulting from the execution of O_ℓ^k , if implementation MPL is used for R_j , by $TC(MPL, j, O_\ell^k)$ and call it the time cost of implementation MPL for R_j in O_ℓ^k . The value of $TC(MPL, j, O_\ell^k)$ is undefined if $AC(MPL, j) = 0$ and it is zero if relation R_j is not used in operation O_ℓ^k .

Recalling that NO_k denotes the number of operations in the k -th run unit and NU denotes the number of run units, the time costs for relation implementations can be computed from:

$$TC(MPL, j) = \sum_{k=1}^{NU} \sum_{\ell=1}^{NO_k} TC(MPL, j, O_\ell^k)$$

In the following we will present the formulas to compute $TC(MPL, j, O_\ell^k)$ for all implementations defined in 3.1. These formulas involve certain types of page fault probabilities which we will discuss first.

3.4.1 Page Fault Probability Model

We define four types of one step page fault probabilities as follows.

- p_1 is the probability of occurrence of a page fault when a record is accessed using the value of a data item in the record (data key value)
- $p_2(R_j)$ is the probability of occurrence of a page fault when the next member record of a set in chain mode is accessed from the current member record of the set. $p_2(R_j)$ applies to chain associations of B under A for $R_j(A, B)$. $p'_2(R_j)$ is similarly defined for chain associations of A under B for $R_j(A, B)$.
- $p_3(R_j)$ is the probability of occurrence of a page fault when a related record is accessed from another record in an association or linkage implementation for R_j . In association implementations $p_3(R_j)$ is the page fault probability for an owner to member (or member to owner) record access.
- p_4 is the probability of occurrence of a page fault when a page that contains security information (values of privacy locks) is accessed.

Our page fault probability model makes certain assumptions which we wish to discuss at this point. It is assumed that the data base will be divided into subdivisions called files. Each file consists of all records of only one type (homogeneous files). No physical ordering is assumed for records of a file. Files are subdivided into pages of PGSZ characters. Only a certain number MSPG of pages of data can be in main storage at a given time. With pages of size PGSZ characters this accounts for a total of PGSZ * MSPG characters of main storage which we call the page buffer size PBSZ.

When a record is accessed through its data key value, the value of a certain data item type is used (by the system) to directly locate the record. For example, suppose relation $R_j(A, B)$ is implemented by a chain with next pointers association of B under A (implementation 9 defined in 3.1.5), then each A-value is stored in a record that is the owner of a set that contains the records in which B-values related to the A-value in R_j are stored. Now for each execution of an operation such as $O = \{\text{FIND ALL}, B, A, R_j, f\}$, all B-values related in R_j to a specific A-value, say a , must be retrieved. The A-value will be provided at execution time and it is based on this value that the system will directly access the record that contains it.

For this type of access no assumptions can be made regarding the contents of the system buffer, because we do

not require that the f executions of the operation O be performed consecutively. For a direct record access, then, any part of the data base (of total size TMSB characters) can be in the buffer (whose size is PBSZ characters) at the time of access. The probability of the record that contains a specific A-value being in storage is $(\text{PBSZ}/\text{TMSB})$ and the page fault probability p_1 is as follows.

$$p_1 = 1 - (\text{PBSZ}/\text{TMSB})$$

Note that $0 < p_1 < 1$ because we assume $0 < \text{PBSZ} < \text{TMSB}$.

The assumption that PBSZ is less than TMSB is a trivial one because otherwise the whole data base can be accommodated in the system buffer and any structure chosen for the data base would cost the same in terms of generating page faults. Another assumption, implicit in the above derivation, is that records of the data base are accessible without the requirement of a sequential scan of the data base or a sequential search of collections of records. In DBTG terms, the data item being used as data key value in a record must be declared as a CALC KEY in the record type.

We now proceed with the derivation of $p_2(R_j)$ and $p'_2(R_j)$ defined above. Referring back to the example operation used above for derivation of p_1 , we note that when the record that contains the proper A-value, a , is retrieved, then a chain of member records must be retrieved. The first member record is accessed using a pointer in the owner record. This is an owner to member record access that we discuss later. But the rest of the record accesses

needed to retrieve all other member records that contain the desired B-values, are performed using the next member pointers in the member records. This is the type of access referred to in the definition of $p_2(R_j)$. Since we assumed that records of the same type are stored in one file, we have a smaller range of records stored in pages of a certain file with these accesses being consecutively performed. In this case, then, the accessed record may be in any of τ_j pages, where τ_j is the number of pages of the file that contains records of the desired type. It is assumed that all MSPG pages in main storage contain records of the member record type. This assumption may result in approximate computation of page fault probabilities, especially in the first few member record accesses, for some paging algorithms used by the system. The exact computation of these probabilities requires detailed modelling of paging policies which is beyond the scope of this research and some information about the final record and set structures. We will assume that the system will, in some way, fill the MSPG main storage pages with the records of the desired type when a chain of member records is being traversed. A similar assumption will be made when computing $p_3(R_j)$, namely that when a related record is accessed the MSPG pages of main storage are filled with records of the owner and member record types.

In case of relation $R_j(A, B)$ where $B = \text{DST}(R_j)$ and $\beta'_j = |B|$, there are β'_j unique B-values stored in β'_j records,

and therefore $\tau_j = (\beta'_j * \text{ARL}) / \text{PGSZ}$ where ARL is the Average Record Length and PGSZ is the size of a page. $p_2(R_j)$ is then given by the following.

$$p_2(R_j) = 1 - \frac{\text{MSPG}}{\tau_j} = 1 - \frac{\text{MSPG} * \text{PGSZ}}{\beta'_j * \text{ARL}} = 1 - \frac{\text{PBSZ}}{\beta'_j * \text{ARL}}$$

The above formula holds as long as $\text{PBSZ} \leq \beta'_j * \text{ARL}$. For $\text{PBSZ} > \beta'_j * \text{ARL}$ we note that the entire file that contains the records of the desired type can be accommodated in the page buffer and therefore in this case we set $p_2(R_j)$ equal to zero.

$$\begin{aligned} p_2(R_j) &= 1 - \frac{\text{PBSZ}}{\beta'_j * \text{ARL}} && \text{if } \text{PBSZ} < \beta'_j * \text{ARL} \\ &= 0 && \text{otherwise} \end{aligned}$$

Similarly $p'_2(R_j)$, which is the page fault probability when the next record in a chain association of A under B for $R_j(A, B)$ is accessed, is given by the following.

$$\begin{aligned} p'_2(R_j) &= 1 - \frac{\text{PBSZ}}{\alpha'_j * \text{ARL}} && \text{if } \text{PBSZ} < \alpha'_j * \text{ARL} \\ &= 0 && \text{otherwise,} \end{aligned}$$

where $\alpha'_j = |\text{ORG}(R_j)|$.

The above formulas for $p_2(R_j)$ and $p'_2(R_j)$ hold for all association implementations except implementation 21 for $R_j(A, B)$ where ARL and α'_j or β'_j parameters are different. This case will be discussed later.

The next category of page fault probabilities to be discussed is the one defined as $p_3(R_j)$ or the probability of occurrence of a page fault when a related record is accessed. In a chain or a pointer array association imple-

mentation, when an owner record is accessed using an owner pointer in a member record or a member record is accessed using a member pointer in an owner record we say that a related record has been accessed. Accesses made to records related by linkage implementations using pointer data items of the implementation is also called a related record access. As in the case of $p_2(R_j)$ discussed above, $p_3(R_j)$ applies only to association and linkage implementations of $R_j(A, B)$. These implementations for $R_j(A, B)$ all require unique A and B-values to be stored in different records. The number of record occurrences involved is therefore known and using ARL as before the sizes of the associated files can be estimated. The record types that contain data item types A and B have α'_j and β'_j records, respectively, where $\alpha'_j = |\text{ORG}(R_j)|$ and $\beta'_j = |\text{DST}(R_j)|$. Sizes of the associated files is then $\alpha'_j * \text{ARL}$ and $\beta'_j * \text{ARL}$ characters respectively, where ARL is the Average Record Length. Assuming that any parts of these two files may be in main storage at the time of access, the formula for $p_3(R_j)$ is given by the following.

$$p_3(R_j) = 1 - \frac{\text{PBSZ}}{(\alpha'_j + \beta'_j) * \text{ARL}} \quad \text{if } \text{PBSZ} \leq (\alpha'_j + \beta'_j) * \text{ARL}$$

$$= 0 \quad \text{otherwise,}$$

where again $\text{PBSZ} = \text{MSPG} * \text{PGSZ}$ is the size of the page buffer in characters.

Finally for p_4 , the probability of occurrence of a page fault when a page that contains security information

is accessed we use an average value based on the assumption that one page of main storage (PGSZ characters) is devoted to security information. Let LKSZ be the maximum amount of storage needed for security locks over all possible configurations. If a configuration needs S bytes then this probability is,

$$p = 1 - \frac{\text{PGSZ}}{S} \quad \text{if } \text{PGSZ} \leq S \text{ and } p = 0 \quad \text{if } \text{PGSZ} > S,$$

where $0 \leq S \leq \text{LKSZ}$. The average value of p denoted by p_4 , assuming S varies continuously from zero to LKSZ, is

$$1 - \frac{\text{PGSZ}}{\text{LKSZ}} * [1 + \ln \frac{\text{LKSZ}}{\text{PGSZ}}].$$

For example, for $\text{LKSZ} = 2400$ characters and $\text{PGSZ} = 2000$ we have $p_4 \approx 0.01$, and for the same LKSZ and $\text{PGSZ} = 4000$; $p_4 = 0$.

Page fault probabilities $p_2(R_j)$, $p'_2(R_j)$ and $p_3(R_j)$ do not apply to implementation 21. In the case of this implementation for $R_j(A, B)$ we note that for a related record access ($p_3(R_j)$) there are three files of records whose pages will be competing for the MSPG available pages in main storage. The first two files are associated with the record types in which A and B are defined, respectively. These two files, respectively, have α'_j and β'_j records of average length ARL. The third file contains the r_j dummy records of size DRSZ of implementation 21. In our definition of implementation 21, the value of DRSZ is given by $\text{DRSZ} = 4 * \text{PTRS}$, where PTRS is the size of an address occurrence (pointer size). This is because each dummy

record, as defined in 3.1.11, consist of four address occurrences. The total size of these three files is then $(\alpha'_j + \beta'_j) * ARL + r_j * DRSZ$ and $p_3(R_j)$ is given by:

$$p_3(R_j) = \max \left[1 - \frac{PBSZ}{(\alpha'_j + \beta'_j) * ARL + r_j * DRSZ}, 0 \right]$$

Similarly, for $p_2(R_j)$ and $p'_2(R_j)$ in the case of implementation 21 for $R_j(A, B)$ only the third file discussed above, namely the file of r_j dummy records, must be considered. $p_2(R_j)$ and $p'_2(R_j)$ are equal in this case because there is only one file of dummy records of total size $r_j * DRSZ$ and they are given by:

$$p_2(R_j) = p'_2(R_j) = \max \left[1 - \frac{PBSZ}{r_j * DRSZ}, 0 \right].$$

In the above formula, $r_j * DRSZ$ is the size of the file of dummy records of implementation 21 for $R_j(A, B)$ and $PBSZ$ is the page buffer size as before. The above probabilities are zero if the entire file can be stored in the buffer, i.e., if $PBSZ > r_j * DRSZ$.

The values of $PBSZ$, $DRSZ$, α'_j , β'_j and r_j used in computing the page fault probabilities are fixed known values. However, values of ARL and $LKSZ$ are estimated values. We will discuss later how these values should be estimated and how sensitive the results are with respect to errors in the estimation. We now proceed with the derivations of time cost formulas for each of the 24 defined implementations for the typical relation $R_j(A, B)$, where again

$$A = \text{ORG}(R_j) , \quad B = \text{DST}(R_j) ,$$

$$\alpha'_j = |A| , \quad \beta'_j = |B| ,$$

$$\bar{m}_j = \text{average number of B-values related to one A-value} \\ \text{in } R_j ,$$

$$\bar{n}_j = \text{average number of A-values related to one B-value} \\ \text{in } R_j .$$

We also recall that SOP denotes the set of operation codes defined as follows: $\text{SOP} = \{\text{FIND FIRST, FIND LAST, FIND ITH, FIND NEXT, FIND PREV., FIND ALL, FIND KEY, STORE FIRST, STORE LAST, STORE KEY, MODIFY, ERASE}\}.$

3.4.2 Time Cost of Implementations 1, 3, 5 and 7

Implementations 1, 3, 5 and 7 for $R_j(A, B)$ as mentioned before store all B-values related to a specific A-value, say a , in the record in which a is stored. This implies that once the record that contains a is accessed all B-values related to it in $R_j(A, B)$ are available without any more record accesses required. For example, an operation such as $\langle \text{FIND ALL, B, A, } R_j(A, B), f \rangle$ in a run unit will require f record accesses. Each record access results in a page fault with probability p_1 giving an expected page fault rate of $f * p_1$.

The same formula holds for all operation codes $op \in \text{SOP}$. On the other hand, for an operation such as $\langle \text{FIND ALL, A, B, } R_j(A, B), f \rangle$, all $\bar{n}_j$ A-values related to a specific B-value are to be accessed. The total record access rate is $f * \bar{n}_j$ and the expected page fault rate is

$f * \bar{n}_j * p_1$. For other operations, namely operations such as $\langle op, A, B, R_j(A, B), f \rangle$ with $op \in SOP - \{FIND\ ALL\}$ the expected page fault rate is $f * p_1$. As far as the privacy decisions are concerned, two locks (one for each of the data item types A and B) must be accessed and checked at every operation execution with each access having a probability p_4 of resulting in a page fault.

The overall time cost of the above implementations (i.e. $MPL = 1, 3, 5, 7$) for $R_j(A, B)$ in O_ℓ^k (ℓ -th operation of k -th run unit) is given by the following.

$$\begin{aligned}
 TC(MPL, j, O_\ell^k) &= f_\ell^k(p_1 + 2p_4) \\
 &\quad \text{if } O_\ell^k = \langle op, B, A, R_j, f_\ell^k \rangle \\
 &= f_\ell^k(\bar{n}_j p_1 + 2p_4) \\
 &\quad \text{if } O_\ell^k = \langle op, A, B, R_j, f_\ell^k \rangle \\
 &\quad \text{and } op = FIND\ ALL \\
 &= f_\ell^k(p_1 + 2p_4) \\
 &\quad \text{if } O_\ell^k = \langle op, A, B, R_j, f_\ell^k \rangle \\
 &\quad \text{and } op \in SOP - \{FIND\ ALL\} \\
 &= 0 \quad \text{otherwise.}
 \end{aligned}$$

3.4.3 Time Cost of Implementations 2, 4, 6 and 8

Implementations 2, 4, 6 and 8 for $R_j(A, B)$ as described before, require that all A-values related to one B-value, say b , be stored in the record in which b is stored. The time costs of these implementations, in terms of the expected page fault rates, can be measured in a manner analogous to those in Section 3.4.2. So for the

above implementations (i.e., for $MPL = 2, 4, 6, 8$) for $R_j(A, B)$ the time cost in operation O_ℓ^k is given by the following.

$$\begin{aligned}
 TC(MPL, j, O_\ell^k) &= f_\ell^k(p_1 + 2p_4) \\
 &\quad \text{if } O_\ell^k = \langle op, A, B, R_j, f_\ell^k \rangle \\
 &= f_\ell^k(\bar{m}_j p_1 + 2p_4) \\
 &\quad \text{if } O_\ell^k = \langle op, B, A, R_j, f_\ell^k \rangle \\
 &\quad \text{and } OP = \text{FIND ALL} \\
 &= f_\ell^k(p_1 + 2p_4) \\
 &\quad \text{if } O_\ell^k = \langle op, B, A, R_j, f_\ell^k \rangle \\
 &\quad \text{and } op \in SOP - \{\text{FIND ALL}\} \\
 &= 0 \quad \text{otherwise.}
 \end{aligned}$$

3.4.4 Time Costs of Implementations 9 and 10

As defined in Section 3.1.5 implementation 9 for $R_j(A, B)$ is called the chain with next pointers association of B under A in which a set type in chain mode establishes the relationships in R_j . We now present time costs of implementation 9 for R_j for different operations.

First consider operations such as

$O_\ell^k = \langle op, B, A, R_j(A, B), f_\ell^k \rangle$. If $op \in \{\text{FIND FIRST}, \text{STORE FIRST}\}$ then for every operation execution the owner record of a set should be accessed based on its data key value (the A -value presented at execution time) at a cost of p_1 and then the first member record of the set must be accessed at a cost of $p_3(R_j)$.

There will be three privacy locks to be accessed (locks

on A, B and R_j) at a cost of $3p_4$.

The total cost of implementation 9 for R_j in $O_\ell^k = \langle \text{op}, B, A, R_j, f_\ell^k \rangle$ is then
 $TC(9, j, O_\ell^k) = f_\ell^k(p_1 + p_3(R_j) + 3p_4)$ for
 $\text{op} \in \{\text{FIND FIRST}, \text{STORE FIRST}\}$.

Similarly for $O_\ell^k = \langle \text{op}, B, A, R_j(A, B), f_\ell^k \rangle$ and
 $\text{op} \in \{\text{FIND LAST}, \text{STORE LAST}, \text{FIND ALL}\}$, accessing the
owner record costs p_1 , accessing the first member record
costs $p_3(R_j)$ and then $(\bar{m}_j - 1)$ member records must be tra-
versed to hit the last member record which costs
 $(\bar{m}_j - 1)p_2(R_j)$. Again there are three locks to be
accessed at a cost of $3p_4$. The total time cost is then
given by the following formula.

$$TC(9, j, O_\ell^k) = f_\ell^k(p_1 + p_3(R_j) + (\bar{m}_j - 1)p_2(R_j) + 3p_4)$$

for $O_\ell^k = \langle \text{op}, B, A, R_j, f_\ell^k \rangle$ and
 $\text{op} \in \{\text{FIND LAST}, \text{STORE LAST},$
FIND ALL}\}

For an operation of the form
 $O_\ell^k = \langle \text{op}, B, A, R_j(A, B), f_\ell^k \rangle$ where $\text{op} = \text{FIND NEXT}$, at
each execution of the operation the next B-value related
in R_j to a specific value a of type A is to be retrieved.
The assumption here is that the owner record has been
accessed and a current member record has been established
in a previous operation, and now the next member record
with respect to this current one is to be retrieved. This
record access would cost $p_2(R_j)$ for every execution of the
operation.

The locks needed to be accessed for privacy decisions are those for the relation R_j and data item type B at a total cost of $2p_4$.

The overall cost of implementing R_j by implementation 9 for operation $O_\ell^k = \langle \text{FIND NEXT}, B, A, R_j(A, B), f_\ell^k \rangle$ is as follows.

$$TC(9, j, O_\ell^k) = f_\ell^k(p_2(R_j) + 2p_4)$$

We now consider an operation such as $O_\ell^k = \langle \text{FIND PREV.}, B, A, R_j(A, B), f_\ell^k \rangle$. This operation requires the retrieval of the "prior" B -value related to a specific A -value (like a). Here again we assume that a certain member record has been retrieved and established as the current member record of the set in a previous operation, and now the prior member record with respect to the current one is to be retrieved. Recall that in implementation 9 only next pointers are provided. Retrieval of a prior member record, then, requires i) the traversal of the chain of pointers in the records that follow the current one until the last one is hit, ii) retrieving the owner record, iii) retrieving the first record in the chain and iv) the traversal of the chain of pointers from the first member record to the one prior to the current record. The traversals in i) and iv) require a total of $(\bar{m}_j - 2)$ record retrievals at a cost of $p_2(R_j)$ per retrieval, record retrievals of ii) and iii) cost $p_3(R_j)$ each. The cost of retrieving the privacy locks will be $3p_4$.

So for $O_\ell^k = \langle \text{FIND PREV.}, B, A, R_j(A, B), f_\ell^k \rangle$ we have

the following.

$$TC(9, j, O_\ell^k) = f_\ell^k(p_3(R_j) + p_3(R_j) + (\bar{m}_j - 2)p_2(R_j) + 3p_4)$$

The time cost of implementation 9 for $R_j(A, B)$ in $O_\ell^k = \langle op, B, A, R_j(A, B) \rangle$ will now be given for $op \in \{\text{FIND ITH, FIND KEY, STORE KEY, MODIFY, ERASE}\}$.

For $op = \text{FIND ITH}$ the i -th B-value related (in R_j) to a specific A-value is to be retrieved. Since implementation 9 is a chain with next pointer association, this means that in a certain set in the data base the i -th member record is to be retrieved. The value of i will be provided at execution time and it can be any value from 1 to $\bar{m}_j$ with equal probability, and so on the average it is

$(\bar{m}_j(\bar{m}_j + 1)/(2\bar{m}_j)) = 1 + (\bar{m}_j - 1/2)$. In order to retrieve the i -th member record, then, it is necessary to access the owner of the set based on its data key value at a cost of p_1 , then access the first member record at a cost of $p_3(R_j)$ and finally traverse a chain of $(\bar{m}_j - 1)/2$ pointers at a cost of $((\bar{m}_j - 1)/2)p_2(R_j)$. Again accessing the three locks required for privacy decisions would cost $3p_4$. Similar arguments can be given for

$op \in \{\text{FIND KEY, STORE KEY, MODIFY, ERASE}\}$ and hence we have the following.

$$TC(9, j, O_\ell^k) = f_\ell^k(p_1 + p_3(R_j) + \frac{\bar{m}_j - 1}{2} p_2(R_j) + 3p_4)$$

$$\text{for } O_\ell^k = \langle op, B, A, R_j(A, B), f_\ell^k \rangle$$

and $op \in \{\text{FIND ITH, FIND KEY, STORE KEY, MODIFY, ERASE}\}$.

We have presented so far the time costs of implementations 9 for $R_j(A, B)$ in all possible operations of type $O_\ell^k = \langle \text{op}, B, A, R_j, f_\ell^k \rangle$. We will now consider operation of type $O_\ell^k = \langle \text{op}, A, B, R_j, f_\ell^k \rangle$.

We recall that implementation 9 for $R_j(A, B)$ is the chain with next pointers association of B under A (as defined in 3.1.5) in which A -values are stored in owner records of some set type. The operations of the above form (i.e., $\langle \text{op}, A, B, R_j, f_\ell^k \rangle$) are then basically owner record accesses from some member record of a set. Since all member records of a set are related to only one owner record, time costs of implementation 9 for R_j is the same for all $\text{op} \in \text{SOP}$. This time cost consists of i) a member record access through its data key value, ii) traversal of a chain of pointers in the member records and iii) accessing the owner record from the last member record. For every operation execution the record accesses in i) and iii) cost p_1 and $p_3(R_j)$ (in expected page fault rate), respectively. The member record accessed in i) is, on the average, located half way down the chain of $\bar{m}_j$ member records and hence the traversals in ii) would cost $\frac{1}{2}(\bar{m}_j - 1)p_2(R_j)$ in expected page fault rate. And as before privacy lock accesses, for locks on A , B and R_j will cost $3p_4$.

The total cost of implementation 9 for R_j in operations of type $O_\ell^k = \langle \text{op}, A, B, R_j(A, B), f_\ell^k \rangle$, $\text{op} \in \text{SOP}$ is given by the following.

$$\text{TC}(9, j, O_\ell^k) = f_\ell^k(p_1 + \frac{\bar{m}_j - 1}{2} p_2(R_j) + p_3(R_j) + 3p_4).$$

Following is a summary of formulas derived in this subsection for time costs of implementation 9 for $R_j(A, B)$ in O_ℓ^k .

$$\begin{aligned} TC(9, j, O_\ell^k) &= f_\ell^k(p_1 + p_3(R_j) + 3p_4) \\ &\quad \text{if } O_\ell^k = \langle \text{op}, B, A, R_j, f_\ell^k \rangle \\ &\quad \text{and op} \in \{\text{FIND FIRST, STORE FIRST}\} \end{aligned}$$

$$\begin{aligned} &= f_\ell^k(p_1 + p_3(R_j) + (\bar{m}_j - 1)p_2(R_j) + 3p_4) \\ &\quad \text{if } O_\ell^k = \langle \text{op}, B, A, R_j, f_\ell^k \rangle \\ &\quad \text{and op} \in \{\text{FIND LAST, STORE LAST,} \\ &\quad \quad \text{FIND ALL}\} \end{aligned}$$

$$\begin{aligned} &= f_\ell^k(p_2(R_j) + 2p_4) \\ &\quad \text{if } O_\ell^k = \langle \text{FIND NEXT, B, A, R}_j, f_\ell^k \rangle \end{aligned}$$

$$\begin{aligned} &= f_\ell^k(2p_3(R_j) + (\bar{m}_j - 2)p_2(R_j) + 3p_4) \\ &\quad \text{if } O_\ell^k = \langle \text{FIND PREV., B, A, R}_j, f_\ell^k \rangle \end{aligned}$$

$$\begin{aligned} &= f_\ell^k(p_1 + p_3(R_j) + \frac{\bar{m}_j - 1}{2} p_2(R_j) + 3p_4) \\ &\quad \text{if } O_\ell^k = \langle \text{op}, B, A, R_j, f_\ell^k \rangle \\ &\quad \text{and op} \in \{\text{FIND ITH, FIND KEY,} \end{aligned}$$

STORE KEY, MODIFY, ERASE}

$$\begin{aligned} &= f_\ell^k(p_1 + \frac{\bar{m}_j - 1}{2} p_2(R_j) + p_3(R_j) + 3p_4) \\ &\quad \text{if } O_\ell^k = \langle \text{op}, A, B, R_j, f_\ell^k \rangle \\ &\quad \text{and op} \in \text{SOP} \end{aligned}$$

$$= 0 \quad \text{otherwise.}$$

As described in 3.1.5, implementation 10 for $R_j(A, B)$ is the chain with next pointer association of A under B. Since this implementation is the converse of 9, derivations of time cost equations for it are analogous to those

presented above for implementation 9 and so the details will not be repeated. The time cost of implementation 10 for $R_j(A, B)$ in operation O_ℓ^k is as follows.

$$\begin{aligned} TC(10, j, O_\ell^k) &= f_\ell^k(p_1 + p_3(R_j) + 3p_6) \\ &\quad \text{if } O_\ell^k = \langle \text{op}, A, B, R_j, f_\ell^k \rangle \\ &\quad \text{and op} \in \{\text{FIND FIRST, STORE FIRST}\} \end{aligned}$$

$$\begin{aligned} &= f_\ell^k(p_1 + p_3(R_j) + (\bar{n}_j - 1)p_2'(R_j) + 3p_4) \\ &\quad \text{if } O_\ell^k = \langle \text{op}, A, B, R_j, f_\ell^k \rangle \\ &\quad \text{and op} \in \{\text{FIND LAST, STORE LAST,} \\ &\quad \quad \text{FIND ALL}\} \end{aligned}$$

$$\begin{aligned} &= f_\ell^k(p_2'(R_j) + 2p_4) \\ &\quad \text{if } O_\ell^k = \langle \text{FIND NEXT, A, B, R}_j, f_\ell^k \rangle \\ &= f_\ell^k(2p_3(R_j) + (\bar{n}_j - 2)p_2'(R_j) + 3p_4) \\ &\quad \text{if } O_\ell^k = \langle \text{FIND PREV., A, B, R}_j, f_\ell^k \rangle \end{aligned}$$

$$\begin{aligned} &= f_\ell^k(p_1 + p_3(R_j) + \frac{\bar{n}_j - 1}{2} p_2'(R_j) + 3p_4) \\ &\quad \text{if } O_\ell^k = \langle \text{op}, A, B, R_j, f_\ell^k \rangle \\ &\quad \text{and op} \in \{\text{FIND ITH, FIND KEY,} \\ &\quad \quad \text{STORE KEY, MODIFY, ERASE}\} \end{aligned}$$

$$\begin{aligned} &= f_\ell^k(p_1 + \frac{\bar{n}_j - 1}{2} p_2'(R_j) + p_3(R_j) + 3p_4) \\ &\quad \text{if } O_\ell^k = \langle \text{op}, B, A, R_j, f_\ell^k \rangle \\ &\quad \text{and op} \in \text{SOP} \end{aligned}$$

$$= 0 \quad \text{otherwise.}$$

3.4.5 Time Cost of Implementations 11 and 12

Implementation 11 for $R_j(A, B)$, defined in 3.1.6 is called the chain with next and owner pointers association

of B under A. Time costs of this implementation differ from those of implementation 9 only where owner pointers (in member records) are used in a series of record accesses. The owner pointers are used in two cases. The first case is for an operation such as $O_{\ell}^k = \langle \text{FIND PREV.}, B, A, R_j(A, B), f_{\ell}^k \rangle$ where the prior (with respect to the current) member record of the set (that implements R_j) is accessed. We assume that the current member record is, on the average, half way down the chain of member records. The above operation, then, requires one record access to the owner record of the set with a page fault probability of $p_3(R_j)$, one record access from the owner to the first member at a cost of $p_3(R_j)$ and $(\bar{m}_j - 1)/2$ record traversals in the chain of member records at a cost of $((\bar{m}_j - 1)/2)p_2(R_j)$.

The cost of accessing privacy locks is $3p_4$ as for implementation 9.

The second case in which time cost of implementation 11 is different from that of implementation 9 is for operations such as $\langle \text{op}, A, B, R_j(A, B), f_{\ell}^k \rangle$, for $\text{op} \in \text{SOP}$. In these types of operations, each execution of the operation requires one record access, through data key values, to the member record that contains the specific B-value and then an owner access using the owner pointer in the member record. These two record accesses cost p_1 and $p_3(R_j)$, respectively, in terms of expected page fault rate.

Again, accessing the privacy locks would cost $3p_4$.

The overall time cost of implementation 11 for $R_j(A, B)$ in operation O_ℓ^k is as follows.

$$\begin{aligned}
 TC(11, j, O_\ell^k) &= f_\ell^k(p_1 + p_3(R_j) + 3p_4) \\
 &\quad \text{if } O_\ell^k = \langle \text{op}, B, A, R_j, f_\ell^k \rangle \\
 &\quad \text{and } \text{op} \in \{\text{FIND FIRST}, \text{STORE FIRST}\} \\
 &= f_\ell^k(p_1 + p_3(R_j) + (\bar{m}_j - 1)p_2(R_j) + 3p_4) \\
 &\quad \text{if } O_\ell^k = \langle \text{op}, B, A, R_j, f_\ell^k \rangle \\
 &\quad \text{and } \text{op} \in \{\text{FIND LAST}, \text{STORE LAST}, \\
 &\quad \quad \quad \text{FIND ALL}\} \\
 &= f_\ell^k(p_2(R_j) + 2p_4) \\
 &\quad \text{if } O_\ell^k = \langle \text{FIND NEXT}, B, A, R_j, f_\ell^k \rangle \\
 &= f_\ell^k(2p_3(R_j) + \frac{\bar{m}_j - 1}{2} p_2(R_j) + 3p_4) \\
 &\quad \text{if } O_\ell^k = \langle \text{FIND PREV.}, B, A, R_j, f_\ell^k \rangle \\
 &= f_\ell^k(p_1 + p_3(R_j) + \frac{\bar{m}_j - 1}{2} p_2(R_j) + 3p_4) \\
 &\quad \text{if } O_\ell^k = \langle \text{op}, B, A, R_j, f_\ell^k \rangle \\
 &\quad \text{and } \text{op} \in \{\text{FIND ITH}, \text{FIND KEY}, \\
 &\quad \quad \quad \text{STORE KEY}, \text{MODIFY}, \text{ERASE}\} \\
 &= f_\ell^k(p_1 + p_3(R_j) + 3p_4) \\
 &\quad \text{if } O_\ell^k = \langle \text{op}, A, B, R_j, f_\ell^k \rangle \\
 &\quad \text{and } \text{op} \in \text{SOP} \\
 &= 0 \quad \quad \quad \text{otherwise.}
 \end{aligned}$$

Implementation 12 for $R_j(A, B)$ was defined in 3.1.6 and it is the chain with owner pointer association of A under B. Time costs of this implementation are analogous to those for implementation 11, presented in 3.4.5.

Following is the time cost of implementation 12 for

$R_j(A, B)$ in operation O_ℓ^k .

$$TC(12, j, O_\ell^k) = f_\ell^k(p_1 + p_3(R_j) + 3p_4)$$

$$\text{if } O_\ell^k = \langle \text{op}, A, B, R_j, f_\ell^k \rangle$$

$$\text{and op} \in \{\text{FIND FIRST, STORE FIRST}\}$$

$$= f_\ell^k(p_1 + p_3(R_j) + (\bar{n}_j - 1) p_2'(R_j) + 3p_4)$$

$$\text{if } O_\ell^k = \langle \text{op}, A, B, R_j, f_\ell^k \rangle$$

$$\text{and op} \in \{\text{FIND LAST, STORE LAST,}$$

$$\text{FIND ALL}\}$$

$$= f_\ell^k(p_2'(R_j) + 2p_4)$$

$$\text{if } O_\ell^k = \langle \text{FIND NEXT, A, B, R}_j, f_\ell^k \rangle$$

$$= f_\ell^k(2p_3(R_j) + \frac{1}{2} (\bar{n}_j - 1) p_2'(R_j) + 3p_4)$$

$$\text{if } O_\ell^k = \langle \text{FIND PREV., A, B, R}_j, f_\ell^k \rangle$$

$$= f_\ell^k(p_1 + p_3(R_j) + \frac{1}{2} (\bar{n}_j - 1) p_2'(R_j) + 3p_4)$$

$$\text{if } O_\ell^k = \langle \text{op}, A, B, R_j, f_\ell^k \rangle$$

$$\text{and op} \in \{\text{FIND ITH, FIND KEY,}$$

$$\text{STORE KEY, MODIFY, ERASE}\}$$

$$= f_\ell^k(p_1 + p_3(R_j) + 3p_4)$$

$$\text{if } O_\ell^k = \langle \text{op}, B, A, R_j, f_\ell^k \rangle$$

$$\text{and op} \in \text{SOP}$$

$$= 0$$

$$\text{otherwise.}$$

3.4.6 Time Cost of Implementations 13 and 14

Implementation 13 for $R_j(A, B)$, defined in 3.1.7 is called the chain with prior pointer association of B under A. Time costs of this implementation differ from those of implementation 9 in two cases where the prior

pointers are used in a series of record accesses.

The first case in which a prior pointer is used is for an operation like $O_\ell^k = \langle \text{op}, B, A, R_j(A, B), f_\ell^k \rangle$ where $\text{op} \in \{\text{FIND LAST}, \text{STORE LAST}\}$. This is when the last record of a set is to be accessed. The prior pointer in the owner record in this case points to the last member in the chain and it is used to access the last record at a cost of $p_3(R_j)$ in expected page fault rate. Of course, the owner record should, like before, be first accessed through its data key value at a cost of p_1 . The total cost of implementation 13 for R_j in O_ℓ^k will then be:

$$\begin{aligned} \text{TC}(13, j, O_\ell^k) &= f_\ell^k(p_1 + p_3(R_j) + 3p_4) \\ &\quad \text{if } O_\ell^k = \langle \text{op}, B, A, R_j, f_\ell^k \rangle \\ &\quad \text{and } \text{op} \in \{\text{FIND LAST}, \text{STORE LAST}\}. \end{aligned}$$

The second case where the cost of implementation 13 is different from that of implementation 9 for $R_j(A, B)$ is for an operation such as $O_\ell^k = \langle \text{FIND PREV.}, B, A, R_j, f_\ell^k \rangle$. This operation requires the retrieval of the "prior" member record (with respect to a previously established "current" member record) of the set that implements R_j . At each execution of O_ℓ^k , then, a single record access, using the prior pointer, is sufficient to locate the previous member record. This record access will cost $p_2(R_j)$ in expected page fault rate. The time cost of implementation 13 for $R_j(A, B)$ in $O_\ell^k = \langle \text{FIND PREV.}, B, A, R_j, f_\ell^k \rangle$ is as follows.

$$\text{TC}(13, j, O_\ell^k) = f_\ell^k(p_2(R_j) + 2p_4).$$

The complete time costs of implementation 13 for

$R_j(A, B)$ in O_ℓ^k is given below.

$$TC(13, j, O_\ell^k) = f_\ell^k(p_1 + p_3(R_j) + (\bar{m}_j - 1)p_2(R_j) + 3p_4)$$

$$\text{if } O_\ell^k = \langle \text{FIND ALL}, B, A, R_j, f_\ell^k \rangle$$

$$= f_\ell^k(p_1 + p_3(R_j) + 3p_4)$$

$$\text{if } O_\ell^k = \langle \text{op}, B, A, R_j, f_\ell^k \rangle$$

$$\text{and op} \in \{\text{FIND FIRST}, \text{STORE FIRST},$$

$$\text{FIND LAST}, \text{STORE LAST}\}$$

$$= f_\ell^k(p_2(R_j) + 2p_4)$$

$$\text{if } O_\ell^k = \langle \text{op}, B, A, R_j, f_\ell^k \rangle$$

$$\text{and op} \in \{\text{FIND NEXT}, \text{FIND PREV.}\}$$

$$= f_\ell^k(p_1 + p_3(R_j) + \frac{1}{2}(\bar{m}_j - 1)p_2(R_j) + 3p_4)$$

$$\text{if } O_\ell^k = \langle \text{op}, B, A, R_j, f_\ell^k \rangle$$

$$\text{and op} \in \{\text{FIND ITH}, \text{FIND KEY},$$

$$\text{STORE KEY}, \text{MODIFY}, \text{ERASE}\}$$

$$= f_\ell^k(p_1 + p_3(R_j) + 3p_4 + \frac{\bar{m}_j - 1}{2} p_2(R_j))$$

$$\text{if } O_\ell^k = \langle \text{op}, A, B, R_j, f_\ell^k \rangle$$

$$\text{and op} \in \text{SOP}$$

$$= 0$$

$$\text{otherwise.}$$

Implementation 14 for $R_j(A, B)$, as defined in 3.1.7 is called the chain with prior pointers association of A under B. Time costs of this implementation are analogous to those for implementation 13 presented above.

Following is the time cost of implementation 14 for $R_j(A, B)$ in operation O_ℓ^k .

$$\begin{aligned}
TC(14, j, O_\ell^k) &= f_\ell^k(p_1 + p_3(R_j) + (\bar{n}_j - 1)p_2'(R_j) + 3p_4) \\
&\quad \text{if } O_\ell^k = \langle \text{FIND ALL}, A, B, R_j, f_\ell^k \rangle \\
&= f_\ell^k(p_1 + p_3(R_j) + 3p_4) \\
&\quad \text{if } O_\ell^k = \langle \text{op}, A, B, R_j, f_\ell^k \rangle \\
&\quad \text{and op} \in \{\text{FIND FIRST, STORE FIRST,} \\
&\quad \quad \text{FIND LAST, STORE LAST}\} \\
&= f_\ell^k(p_2'(R_j) + 2p_4) \\
&\quad \text{if } O_\ell^k = \langle \text{op}, A, B, R_j, f_\ell^k \rangle \\
&\quad \text{and op} \in \{\text{FIND NEXT, FIND PREV.}\} \\
&= f_\ell^k(p_1 + p_3(R_j) + \frac{1}{2}(\bar{n}_j - 1)p_2'(R_j) + 3p_4) \\
&\quad \text{if } O_\ell^k = \langle \text{op}, A, B, R_j, f_\ell^k \rangle \\
&\quad \text{and op} \in \{\text{FIND ITH, FIND KEY,} \\
&\quad \quad \text{STORE KEY, MODIFY, ERASE}\} \\
&= f_\ell^k(p_1 + p_3(R_j) + 3p_4 + \frac{\bar{n}_j - 1}{2} p_2'(R_j)) \\
&\quad \text{if } O_\ell^k = \langle \text{op}, B, A, R_j, f_\ell^k \rangle \\
&\quad \text{and op} \in \text{SOP} \\
&= 0 \quad \quad \text{otherwise.}
\end{aligned}$$

3.4.7 Time Cost of Implementations 15 and 16

Implementation 15 for $R_j(A, B)$ as defined in 3.1.8 is called the chain with next, prior and owner pointers association of B under A. Time costs of this implementation are different from those for implementation 9 only when owner or prior (or both) pointers are used in a series of record accesses. Owner pointers are used in operations of the form $O_\ell^k = \langle \text{op}, A, B, R_j(A, B), f_\ell^k \rangle$ with $\text{op} \in \text{SOP}$ and prior pointers are used in operations such as

$O_\ell^k = \langle op, B, A, R_j(A, B), f_\ell^k \rangle$ with

$op \in \{\text{FIND LAST, STORE LAST, FIND PREV.}\}$. The time cost of implementation 15 for $R_j(A, B)$ in O_ℓ^k is then as follows.

$$\begin{aligned}
 TC(15, j, O_\ell^k) &= f_\ell^k(p_1 + p_3(R_j) + (\bar{m}_j - 1)p_2(R_j) + 3p_4) \\
 &\quad \text{if } O_\ell^k = \langle \text{FIND ALL, B, A, R}_j, f_\ell^k \rangle \\
 &= f_\ell^k(p_1 + p_3(R_j) + 3p_4) \\
 &\quad \text{if } O_\ell^k = \langle op, B, A, R_j, f_\ell^k \rangle \\
 &\quad \text{and } op \in \{\text{FIND FIRST, STORE FIRST,} \\
 &\quad \quad \quad \text{FIND LAST, STORE LAST}\} \\
 &= f_\ell^k(p_2(R_j) + 2p_4) \\
 &\quad \text{if } O_\ell^k = \langle op, B, A, R_j, f_\ell^k \rangle \\
 &\quad \text{and } op \in \{\text{FIND NEXT, FIND PREV.}\} \\
 &= f_\ell^k(p_1 + p_3(R_j) + \frac{1}{2}(\bar{m}_j - 1)p_2(R_j) + 3p_4) \\
 &\quad \text{if } O_\ell^k = \langle op, B, A, R_j, f_\ell^k \rangle \\
 &\quad \text{and } op \in \{\text{FIND ITH, FIND KEY,} \\
 &\quad \quad \quad \text{STORE KEY, MODIFY, ERASE}\} \\
 &= f_\ell^k(p_1 + p_3(R_j) + 3p_4) \\
 &\quad \text{if } O_\ell^k = \langle op, A, B, R_j, f_\ell^k \rangle \\
 &\quad \text{and } op \in \text{SOP} \\
 &= 0 \quad \quad \quad \text{otherwise.}
 \end{aligned}$$

Implementation 16 for $R_j(A, B)$, as defined in 3.1.8, is the chain with next, prior and owner pointers association of A under B. This implementation is the converse of implementation 15 and therefore formulas for computing time costs for the implementation are analogous to those for implementation 15 presented above.

Time costs of implementation 16 for $R_j(A, B)$ in O_ℓ^k is

given by the following.

$$\begin{aligned}
 TC(16, j, O_\ell^k) &= f_\ell^k(p_1 + p_3(R_j) + (\bar{n}_j - 1)p_2'(R_j) + 3p_4) \\
 &\quad \text{if } O_\ell^k = \langle \text{FIND ALL}, A, B, R_j, f_\ell^k \rangle \\
 &= f_\ell^k(p_1 + p_3(R_j) + 3p_4) \\
 &\quad \text{if } O_\ell^k = \langle \text{op}, A, B, R_j, f_\ell^k \rangle \\
 &\quad \text{and op} \in \{\text{FIND FIRST, STORE FIRST,} \\
 &\quad \quad \text{FIND LAST, STORE LAST}\} \\
 &= f_\ell^k(p_2'(R_j) + 2p_4) \\
 &\quad \text{if } O_\ell^k = \langle \text{op}, A, B, R_j, f_\ell^k \rangle \\
 &\quad \text{and op} \in \{\text{FIND NEXT, FIND PREV.}\} \\
 &= f_\ell^k(p_1 + p_3(R_j) + \frac{1}{2}(\bar{n}_j - 1)p_2'(R_j) + 3p_4) \\
 &\quad \text{if } O_\ell^k = \langle \text{op}, A, B, R_j, f_\ell^k \rangle \\
 &\quad \text{and op} \in \{\text{FIND ITH, FIND KEY,} \\
 &\quad \quad \text{STORE KEY, MODIFY, ERASE}\} \\
 &= f_\ell^k(p_1 + p_3(R_j) + 3p_4) \\
 &\quad \text{if } O_\ell^k = \langle \text{op}, B, A, R_j, f_\ell^k \rangle \\
 &\quad \text{and op} \in \text{SOP} \\
 &= 0 \quad \text{otherwise.}
 \end{aligned}$$

3.4.8 Time Cost of Implementations 17 and 18

Implementation 17 for $R_j(A, B)$ was described in 3.1.9 and is called the pointer array association of B under A. The pointer array of each set is assumed to be stored in the page in which the owner record of the set is stored so the retrieval of the pointer array is accomplished at no extra cost once the owner record is accessed. In the following we will first present the time cost of implemen-

tation 17 for R_j in operations of the form $\langle \text{op}, A, B, R_j, f_\ell^k \rangle$ and then we will consider operations of the form $\langle \text{op}, B, A, R_j, f_\ell^k \rangle$.

Operations such as $\langle \text{op}, A, B, R_j, f_\ell^k \rangle$ require owner record accesses when implementation 17 is selected for R_j . As defined in 3.1.9, owner pointers are not provided in the member records of the pointer array associations and therefore owner record accesses are not possible in this type of implementation. We associate a time cost of infinity with implementation 17 in the above operations, i.e.,

$$\text{TC}(17, j, O_\ell^k) = \infty \quad \text{if } O_\ell^k = \langle \text{op}, A, B, R_j(A, B), f_\ell^k \rangle \\ \text{and } \text{op} \in \text{SOP}.$$

An operation such as $\langle \text{FIND ALL}, B, A, R_j, f_\ell^k \rangle$ requires i) one record access to the owner record of the set (that contain the specified A-value) and ii) $\bar{m}_j$ accesses to member records to retrieve all B-values related to that A-value in R_j . The record access in i) is an access through the data key value and costs p_1 in expected page fault rate. The $\bar{m}_j$ record accesses in ii) are related record accesses (owner to member) at a total cost of $\bar{m}_j p_3(R_j)$. For making the proper privacy decisions three locks have to be accessed and checked, namely those for A, B and R_j .

The total cost of implementation 17 for $R_j(A, B)$ in $O_\ell^k = \langle \text{FIND ALL}, B, A, R_j, f_\ell^k \rangle$ is then,

$$\text{TC}(17, j, O_\ell^k) = f_\ell^k(p_1 + \bar{m}_j p_3(R_j) + 3p_4).$$

We now consider the operation $\langle \text{FIND KEY}, B, A, R_j, f_\ell^k \rangle$ which requires the retrieval of a record containing a specific B-value related to a given A-value in $R_j(A, B)$. The record access required by this operation are i) an owner record access through the data key value (the A-value), ii) an average of $\frac{1}{2} \bar{m}_j$ member record accesses, using the pointers in the pointer array until the member record that contains the desired B-value is hit and iii) three accesses to retrieve privacy locks. Record accesses in i), ii) and iii) have time cost of p_1 , $\frac{1}{2} \bar{m}_j p_3(R_j)$ and $3p_4$, in expected page fault rate, respectively.

It can similarly be shown that the same time costs result for every $op \in \{\text{FIND KEY}, \text{STORE KEY}, \text{MODIFY}, \text{ERASE}\}$ and therefore the time cost of implementation 17 for $R_j(A, B)$ in $O_\ell^k = \langle op, B, A, R_j, f_\ell^k \rangle$ is as follows.

$$\begin{aligned} \text{TC}(17, j, O_\ell^k) &= f_\ell^k (p_1 + \frac{1}{2} \bar{m}_j p_3(R_j) + 3p_4) \\ &\quad \text{if } op \in \{\text{FIND KEY}, \text{STORE KEY}, \\ &\quad \quad \quad \text{MODIFY}, \text{ERASE}\}. \end{aligned}$$

The next set of operations to be considered is the set of operations such as $O_\ell^k = \langle op, B, A, R_j, f_\ell^k \rangle$ with $op \in \{\text{FIND FIRST}, \text{STORE FIRST}, \text{FIND LAST}, \text{STORE LAST}, \text{FIND ITH}\}$. For example, operation $\langle \text{FIND ITH}, B, A, R_j, f_\ell^k \rangle$, at each operation execution, requires the retrieval of the i-th B-value related to a specific A-value, say a, in R_j . The values of i and a will be provided at execution time. Each execution of the operation requires one owner record access, through data key value to locate the owner record

that contains a . Once this record is located the array of pointers to member records is available. The i -th element of the array contains the address of the i -th member record which in turn contains the desired B-value. So two record accesses are needed that cost p_1 and $p_3(R_j)$ in expected page fault rate, respectively. So we have the following.

$$\begin{aligned} TC(17, j, O_\ell^k) &= f_\ell^k(p_1 + p_3(R_j) + 3p_4) \\ &\quad \text{if } O_\ell^k = \langle \text{op}, B, A, R_j, f_\ell^k \rangle \\ &\quad \text{and } \text{op} \in \{\text{FIND ITH}, \text{FIND FIRST}, \\ &\quad \quad \quad \text{FIND LAST}, \text{STORE FIRST}, \\ &\quad \quad \quad \text{STORE LAST}\}. \end{aligned}$$

The last two operations to be considered are $\langle \text{FIND NEXT}, B, A, R_j, f_\ell^k \rangle$ and $\langle \text{FIND PREV.}, B, A, R_j, f_\ell^k \rangle$. In these two cases the operation requires the retrieval of the next (or prior) member record (with respect to some previously established current member record) of the set. Member records of a set in pointer array mode, as defined in Section 3.1, do not contain any pointers to either the owner record or any other member record of the set. This makes it impossible (at least extremely hard) to process the FIND NEXT and FIND PREV. operations and therefore we associate a time cost of infinity to implementation 17 for R_j in $O_\ell^k = \langle \text{op}, B, A, R_j(A, B), f_\ell^k \rangle$ where $\text{op} \in \{\text{FIND NEXT}, \text{FIND PREV.}\}$.

The complete time cost of implementation 17 for $R_j(A, B)$ in operation O_ℓ^k is as follows.

$$\begin{aligned}
TC(17, j, O_\ell^k) &= f_\ell^k(p_1 + \bar{m}_j p_3(R_j) + 3p_4) \\
&\quad \text{if } O_\ell^k = \langle \text{FIND ALL}, B, A, R_j, f_\ell^k \rangle \\
&= f_\ell^k(p_1 + \frac{1}{2} \bar{m}_j p_3(R_j) + 3p_4) \\
&\quad \text{if } O_\ell^k = \langle \text{op}, B, A, R_j, f_\ell^k \rangle \\
&\quad \text{and op} \in \{\text{FIND KEY}, \text{STORE KEY}, \\
&\quad \quad \text{MODIFY}, \text{ERASE}\} \\
&= f_\ell^k(p_1 + p_3(R_j) + 3p_4) \\
&\quad \text{if } O_\ell^k = \langle \text{op}, B, A, R_j, f_\ell^k \rangle \\
&\quad \text{and op} \in \{\text{FIND ITH}, \text{FIND FIRST}, \\
&\quad \quad \text{STORE FIRST}, \text{FIND LAST}, \\
&\quad \quad \text{STORE LAST}\} \\
&= \infty \quad \text{if } O_\ell^k = \langle \text{op}, B, A, R_j, f_\ell^k \rangle \\
&\quad \text{and op} \in \{\text{FIND NEXT}, \text{FIND PREV.}\} \text{ or} \\
&\quad O_\ell^k = \langle \text{op}, A, B, R_j, f_\ell^k \rangle \\
&\quad \text{and op} \in \text{SOP} \\
&= 0 \quad \quad \quad \text{otherwise.}
\end{aligned}$$

Implementation 18 for $R_j(A, B)$ was defined in 3.1.9 as the pointer array association of A under B. Since this implementation is the converse of implementation 17 for $R_j(A, B)$, derivations of time cost equations for it are analogous to those presented above and hence only the results are given below.

Time cost of implementation 18 for $R_j(A, B)$ in operation O_ℓ^k is as follows.

$$\begin{aligned}
TC(18, j, O_\ell^k) &= f_\ell^k(p_1 + \bar{m}_j p_3(R_j) + 3p_4) \\
&\quad \text{if } O_\ell^k = \langle \text{FIND ALL}, B, A, R_j, f_\ell^k \rangle \\
&= f_\ell^k(p_1 + \frac{1}{2} \bar{n}_j p_3(R_j) + 3p_4) \\
&\quad \text{if } O_\ell^k = \langle \text{op}, B, A, R_j, f_\ell^k \rangle \\
&\quad \text{and op} \in \{\text{FIND KEY}, \text{STORE KEY}, \\
&\quad \quad \text{MODIFY}, \text{ERASE}\} \\
&= f_\ell^k(p_1 + p_3(R_j) + 3p_4) \\
&\quad \text{if } O_\ell^k = \langle \text{op}, B, A, R_j, f_\ell^k \rangle \\
&\quad \text{and op} \in \{\text{FIND ITH}, \text{FIND FIRST}, \\
&\quad \quad \text{STORE FIRST}, \text{FIND LAST}, \\
&\quad \quad \text{STORE LAST}\} \\
&= \infty \quad \text{if } O_\ell^k = \langle \text{op}, B, A, R_j, f_\ell^k \rangle \\
&\quad \text{and op} \in \{\text{FIND NEXT}, \text{FIND PREV.}\} \text{ or} \\
&\quad O_\ell^k = \langle \text{op}, A, B, R_j, f_\ell^k \rangle \\
&\quad \text{and op} \in \text{SOP} \\
&= 0 \quad \quad \quad \text{otherwise.}
\end{aligned}$$

3.4.9 Time Cost of Implementation 19 and 20

Implementation 19 for $R_j(A, B)$ as defined in 3.1.10 is called the pointer array with owner pointer association of B under A. It is similar to implementation 17 except that in each set of the association member records have an extra pointer field that contains the address of the owner record. Time costs of implementation 19 are different from those of implementation 19 for $R_j(A, B)$ only in operations that require the use of these owner pointers in a series of record traversals.

In an operation such as $O_\ell^k = \langle \text{op}, B, A, R_j(A, B), f_\ell^k \rangle$ and where $\text{op} \in \{\text{FIND NEXT}, \text{FIND PREV.}\}$, the next (or prior) member record of a set is to be accessed from the current member record. The operation (at each execution) requires one record access to the owner of the set at a cost of $p_3(R_j)$ and a subsequent member record access (to the prior or next record) again at a cost of $p_3(R_j)$. Accesses to privacy locks will cost $3p_4$ since locks on data items A and B, and relation R_j should be retrieved and checked.

The total time cost is then as follows.

$$\begin{aligned} \text{TC}(19, j, O_\ell^k) &= f_\ell^k(2p_3(R_j) + 3p_4) \\ &\quad \text{if } O_\ell^k = \langle \text{op}, B, A, R_j(A, B), f_\ell^k \rangle \\ &\quad \text{and } \text{op} \in \{\text{FIND NEXT}, \text{FIND PREV.}\} \end{aligned}$$

Operations of the form of $O_\ell^k = \langle \text{op}, A, B, R_j(A, B), f_\ell^k \rangle$ where $\text{op} \in \text{SOP}$ are basically owner record accesses from a member record (accessed through its data key value) when implementation 19 is chosen for $R_j(A, B)$. Each execution of the operation requires one record access to the member record that contains a specific B-value and one record access (using the owner pointer in the member record) to the owner record of the set (that implements R_j). These two record accesses cost p_1 and $p_3(R_j)$, in expected page fault rate, respectively. So we have the following.

$$\begin{aligned} \text{TC}(19, j, O_\ell^k) &= f_\ell^k(p_1 + p_3(R_j) + 3p_4) \\ &\quad \text{if } O_\ell^k = \langle \text{op}, A, B, R_j(A, B), f_\ell^k \rangle \\ &\quad \text{and } \text{op} \in \text{SOP}. \end{aligned}$$

The complete time cost of implementation 19 for $R_j(A, B)$ in operation O_ℓ^k is as follows.

$$\begin{aligned}
 TC(19, j, O_\ell^k) &= f_\ell^k(p_1 + \bar{m}_j p_3(R_j) + 3p_4) \\
 &\quad \text{if } O_\ell^k = \langle \text{FIND ALL}, B, A, R_j, f_\ell^k \rangle \\
 &= f_\ell^k(p_1 + \frac{1}{2} \bar{m}_j p_3(R_j) + 3p_4) \\
 &\quad \text{if } O_\ell^k = \langle \text{op}, B, A, R_j, f_\ell^k \rangle \\
 &\quad \text{and op} \in \{\text{FIND KEY}, \text{STORE KEY}, \\
 &\quad \quad \quad \text{MODIFY}, \text{ERASE}\} \\
 &= f_\ell^k(p_1 + p_3(R_j) + 3p_4) \\
 &\quad \text{if } O_\ell^k = \langle \text{op}, B, A, R_j, f_\ell^k \rangle \\
 &\quad \text{and op} \in \{\text{FIND ITH}, \text{FIND FIRST}, \\
 &\quad \quad \quad \text{STORE FIRST}, \text{FIND LAST}, \\
 &\quad \quad \quad \text{STORE LAST}\} \\
 &= f_\ell^k(2p_3(R_j) + 3p_4) \\
 &\quad \text{if } O_\ell^k = \langle \text{op}, B, A, R_j, f_\ell^k \rangle \\
 &\quad \text{and op} \in \{\text{FIND NEXT}, \text{FIND PREV.}\} \\
 &= f_\ell^k(p_1 + p_3(R_j) + 3p_4) \\
 &\quad \text{if } O_\ell^k = \langle \text{op}, A, B, R_j, f_\ell^k \rangle \\
 &\quad \text{and op} \in \text{SOP} \\
 &= 0 \quad \quad \quad \text{otherwise.}
 \end{aligned}$$

Implementation 20 for $R_j(A, B)$ as defined in 3.1.10 is called the pointer array with owner pointer association of A under B. The derivations of time cost equations for this implementation are similar to those for implementation 19 for R_j and hence only the results are presented below.

Time cost of implementation 20 for $R_j(A, B)$ in operation O_ℓ^k is as follows.

$$\begin{aligned}
TC(20, j, O_\ell^k) &= f_\ell^k(p_1 + \bar{n}_j p_3(R_j) + 3p_4) \\
&\quad \text{if } O_\ell^k = \langle \text{FIND ALL}, A, B, R_j, f_\ell^k \rangle \\
&= f_\ell^k(p_1 + \frac{1}{2} \bar{n}_j p_3(R_j) + 3p_4) \\
&\quad \text{if } O_\ell^k = \langle \text{op}, A, B, R_j, f_\ell^k \rangle \\
&\quad \text{and op} \in \{\text{FIND KEY}, \text{STORE KEY}, \\
&\quad \quad \text{MODIFY}, \text{ERASE}\} \\
&= f_\ell^k(p_1 + p_3(R_j) + 3p_4) \\
&\quad \text{if } O_\ell^k = \langle \text{op}, A, B, R_j, f_\ell^k \rangle \\
&\quad \text{and op} \in \{\text{FIND ITH}, \text{FIND FIRST}, \\
&\quad \quad \text{STORE FIRST}, \text{FIND LAST}, \\
&\quad \quad \text{STORE LAST}\} \\
&= f_\ell^k(2p_3(R_j) + 3p_4) \\
&\quad \text{if } O_\ell^k = \langle \text{op}, A, B, R_j, f_\ell^k \rangle \\
&\quad \text{and op} \in \{\text{FIND NEXT}, \text{FIND PREV.}\} \\
&= f_\ell^k(p_1 + p_3(R_j) + 3p_4) \\
&\quad \text{if } O_\ell^k = \langle \text{op}, B, A, R_j, f_\ell^k \rangle \\
&\quad \text{and op} \in \text{SOP} \\
&= 0 \quad \quad \text{otherwise.}
\end{aligned}$$

3.4.10 Time Cost of Implementation 21

Implementation 21 for $R_j(A, B)$, as defined in 3.1.11 is the dummy record association of A and B. We first consider operations such as $\langle \text{op}, B, A, R_j(A, B), f \rangle$ and derive time costs of implementation 21 for different operation codes, op, in the set of operation codes SOP. We will then present the time cost formulas for the converse operations $\langle \text{op}, A, B, R_j(A, B), f \rangle$, $\text{op} \in \text{SOP}$. We recall

that in this implementation two set types are defined where A and B appear in the owner record type of the "first" and "second" set type, respectively.

An operation such as $\langle \text{FIND ALL}, B, A, R_j(A, B), f \rangle$ requires, at each execution, the following record accesses. One record access to locate the record that contains a specific A-value at a cost of p_1 . This record is the owner record of a set of the first type with an average of $\bar{m}_j$ dummy member records that have to be accessed. The first one of these records can be accessed at a cost of $p_3(R_j)$ and the remaining $(\bar{m}_j - 1)$ at a cost of $(\bar{m}_j - 1) * p_2(R_j)$. And finally, the owner records of the $\bar{m}_j$ data base sets (of the second type) must be accessed to obtain all B-values related to the A-value in $R_j(A, B)$. These last $\bar{m}_j$ record accesses cost $\bar{m}_j * p_3(R_j)$ in expected page fault rate. Adding to the above $4 * p_4$ the cost of four privacy decisions to check the locks on the two data items A and B and the two data base sets we get the following.

$$TC(21, j, O_\ell^k) = f_\ell^k(p_1 + (\bar{m}_j - 1)p_3(R_j) + (\bar{m}_j - 1)p_2(R_j) + 4p_4)$$

$$\text{if } O_\ell^k = \langle \text{FIND ALL}, B, A, R_j, f_\ell^k \rangle.$$

Next we examine the time cost of implementation 21 for $R_j(A, B)$ in an operation such as $O_\ell^k = \langle \text{op}, B, A, R_j(A, B), f_\ell^k \rangle$ where $\text{op} \in \{\text{FIND FIRST}, \text{STORE FIRST}\}$. In this case the first record access that locates the record containing a specific A-value costs p_1 . Then only one dummy record is accessed from that record at

a cost of $p_3(R_j)$. The later record contains the address of the record in which the first B-value related to that specific A-value is stored and it is accessed at a cost of $p_3(R_j)$. Again the four required privacy decisions cost $4p_4$ in expected page fault rate. The time cost of implementation 21 for $R_j(A, B)$ in operations of the above form is given by

$$TC(21, j, O_\ell^k) = f_\ell^k(p_1 + 2p_3(R_j) + 4p_4).$$

We can also arrive at the same formula by setting $\bar{m}_j = 1$ in the time cost formula derived for

$$O_\ell^k = \langle \text{FIND ALL}, B, A, R_j, f_\ell^k \rangle.$$

The derivation of time costs of implementation 21 for $R_j(A, B)$ in operations such as $O_\ell^k = \langle \text{op}, B, A, R_j(A, B), f_\ell^k \rangle$ where $\text{op} \in \{\text{FIND LAST}, \text{STORE LAST}\}$ can be simplified by observing that this case is similar to the case where $\text{op} = \text{FIND ALL}$ discussed above with the difference that instead of $\bar{m}_j$ owner record accesses to data base sets of the second type only one owner record access is necessary and hence we have the following.

$$\begin{aligned} TC(21, j, O_\ell^k) &= f_\ell^k(p_1 + 2p_3(R_j) + (\bar{m}_j - 1)p_2(R_j) + 4p_4) \\ &\quad \text{if } O_\ell^k = \langle \text{op}, B, A, R_j, f_\ell^k \rangle \\ &\quad \text{and } \text{op} \in \{\text{FIND LAST}, \text{STORE LAST}\}. \end{aligned}$$

The time cost of implementation 21 for $R_j(A, B)$ in an operations such as $O_\ell^k = \langle \text{FIND ITH}, B, A, R_j, f_\ell^k \rangle$, at each execution of the operation requires the following record accesses. As in the previous case, a record containing a specific A-value must be accessed at a time cost of p_1 .

This record is the owner of a data base set of the first type; accessing the first member record of this set costs $p_3(R_j)$. Since the required i -th B-value related to the A-value retrieved above (in R_j) is on the average halfway down the list of such B-values, $\frac{1}{2}(\bar{m}_j-1)$ of the dummy records must be accessed at a cost of $\frac{1}{2}(\bar{m}_j-1)p_2(R_j)$, and then the owner of the final dummy record in its role as a member of a set of the second type must be accessed at a cost of $p_3(R_j)$. Adding to the above $4p_4$ for the cost of making necessary privacy decisions gives the following formula for the time cost of implementation 21 for $R_j(A, B)$ in operations such as $O_\ell^k = \langle \text{op}, B, A, R_j, f_\ell^k \rangle$, $\text{op} = \text{FIND ITH}$. With similar arguments we can show the formula to be valid for all $\text{op} \in \{\text{FIND ITH}, \text{FIND KEY}, \text{STORE KEY}, \text{MODIFY}, \text{ERASE}\}$.

$$\begin{aligned} \text{TC}(21, j, O_\ell^k) &= f_\ell^k(p_1 + 2p_3(R_j) + \frac{1}{2}(\bar{m}_j-1)p_2(R_j) + 4p_4) \\ &\quad \text{if } O_\ell^k = \langle \text{op}, B, A, R_j, f_\ell^k \rangle \\ &\quad \text{and } \text{op} \in \{\text{FIND ITH}, \text{FIND KEY}, \\ &\quad \quad \quad \text{STORE KEY}, \text{MODIFY}, \text{ERASE}\}. \end{aligned}$$

We next consider the time cost of implementation 21 for $R_j(A, B)$ in an operation such as $O_\ell^k = \langle \text{FIND NEXT}, B, A, R_j, f_\ell^k \rangle$. At each execution of this operation the next B-value, b , related to a specific A-value, a , in R_j is to be retrieved. Again, as explained in derivations of time costs for implementation 9, it is assumed that in a previous operation a current B-value say b' related to a in R_j was accessed and now the next B-value,

b , related to a is to be retrieved. The dummy record associated with the pair $\langle a, b \rangle \in R_j$ is available at the time of the execution of the operation. As a member record in a set of the first type, this dummy record contains the address of the dummy record associated with $\langle a, b \rangle \in R_j$. Accessing the latter record costs $p_2(R_j)$ in expected page fault rate. The dummy record associated with the pair $\langle a, b \rangle \in R_j$ contains the address of the record in which b is stored. Accessing this record costs $p_3(R_j)$. Three locks must be accessed, one for data item B and the other two for the two data base sets at a total cost of $3p_4$. The total cost of implementation 21 for $R_j(A, B)$ in $O_\ell^k = \langle \text{FIND NEXT}, B, A, R_j, f_\ell^k \rangle$ is then given by:

$$TC(21, j, O_\ell^k) = f_\ell^k(p_2(R_j) + p_3(R_j) + 3p_4)$$

The last operation code to be considered is

$op = \text{FIND PREV.}$ In the case of an operation such as $O_\ell^k = \langle \text{FIND PREV.}, B, A, R_j(A, B), f_\ell^k \rangle$, the prior B -value, b , with respect to the current B -value, b' , (established in a previous operation) both related to a specific A -value, a , in R_j must be retrieved. The dummy record associated with $\langle a, b' \rangle \in R_j$ is the prior record (with respect to the record associated with $\langle a, b \rangle \in R_j$) in the chain of dummy member records of a data base set of the first type whose owner record contains a .

In order to retrieve the desired B -value, the dummy record associated with $\langle a, b \rangle$ must be retrieved. This

requires 1) the retrieval of the record (say A_1) that contains a using the proper owner pointer in the dummy record associated with the pair $\langle a, b \rangle$ and then 2) accessing the first member record in the set whose owner is A_1 using the member pointer in A_1 and then 3) retrieving all dummy records (in the chain) that precede the one associated with the pair $\langle a, b \rangle$ and finally 4) the owner record in a set of the second type (of which the dummy record associated with $\langle a, b' \rangle$ is a member) must be retrieved using the proper owner record pointer. The above record traversals cost $p_3(R_j)$, $p_3(R_j)$, $\frac{1}{2}(\bar{m}_j-1)p_2(R_j)$ and $p_3(R_j)$ respectively. Adding to the above the time cost of the four privacy decisions $4p_4$ gives the following formula for the time cost of implementation 21 for $R_j(A, B)$ in

$$O_\ell^k = \langle \text{FIND PREV.}, B, A, R_j, f_\ell^k \rangle.$$

$$TC(21, j, O_\ell^k) = f_\ell^k(3p_3(R_j) + \frac{1}{2}(\bar{m}_j-1)p_2(R_j) + 4p_4).$$

Following is a summary of formulas derived above for the time cost of implementation 21 for $R_j(A, B)$ in the ℓ -th operation of k -th run unit O_ℓ^k .

$$\begin{aligned} TC(21, j, O_\ell^k) &= f_\ell^k(p_1 + 2p_3(R_j) + 4p_4) \\ &\quad \text{if } op \in \{\text{FIND FIRST, STORE FIRST}\} \\ &= f_\ell^k(p_1 + 2p_3(R_j) + (m-1)p_2(R_j) + 4p_4) \\ &\quad \text{if } op \in \{\text{FIND LAST, STORE LAST}\} \\ &= f_\ell^k(p_1 + 2p_3(R_j) + \frac{1}{2}(m-1)p_2(R_j) + 4p_4) \\ &\quad \text{if } op \in \{\text{FIND ITH, FIND KEY,} \\ &\quad \quad \quad \text{STORE KEY, MODIFY, ERASE}\} \end{aligned}$$

$$\begin{aligned}
&= f_{\ell}^k(p_1 + (m-1)p_3(R_j) + (m-1)p_2(R_j) + 4p_4) \\
&\quad \text{if } op = \text{FIND ALL} \\
&= f_{\ell}^k(p_2(R_j) + p_3(R_j) + 3p_4) \\
&\quad \text{if } op = \text{FIND NEXT} \\
&= f_{\ell}^k(3p_3(R_j) + \frac{1}{2}(m-1)p_2(R_j) + 4p_4) \\
&\quad \text{if } op = \text{FIND PREV.} \\
&= 0 \quad \text{otherwise.}
\end{aligned}$$

where in the above $m = \bar{m}_j$ if $O_{\ell}^k = \langle op, B, A, R_j, f_{\ell}^k \rangle$ and $m = \bar{n}_j$ if $O_{\ell}^k = \langle op, A, B, R_j, f_{\ell}^k \rangle$.

Note that in the case of implementation 21 for R_j , $p_2(R_j)$ and $p_3(R_j)$ are computed using a different formula, compared to the formulas for all other implementations and that $p_2(R_j)$ and $p_2'(R_j)$ are equal in this case (see 3.4.1).

3.4.11 Time Costs of Implementations 22, 23 and 24

Implementation 22 for $R_j(A, B)$, as defined in 3.1.12, is the single linkage of B under A . The structure of the implementation is very much similar to that of implementation 17. The major difference between the two implementations for $R_j(A, B)$ is that in 17 a data base set should be defined to establish relationships in R_j and therefore only one-to-many relations can be implemented by 17. However, in the case of a linkage implementation this limitation is not necessary and in fact implementation 22 may be used to implement a many-to-many relationship. This difference, however, does not make the time cost of implementation 22 different from that of implementation 17 and

hence we have.

$$TC(22, j, O_{\ell}^k) = TC(17, j, O_{\ell}^k)$$

And similarly the following is true.

$$TC(23, j, O_{\ell}^k) = TC(18, j, O_{\ell}^k)$$

$$TC(24, j, O_{\ell}^k) = TC(17, j, O_{\ell}^k)$$

$$\text{if } O_{\ell}^k = \langle \text{op}, B, A, R_j(A, B), f_{\ell}^k \rangle$$

$$\text{and } \text{op} \in \text{SOP}$$

$$= TC(18, j, O_{\ell}^k)$$

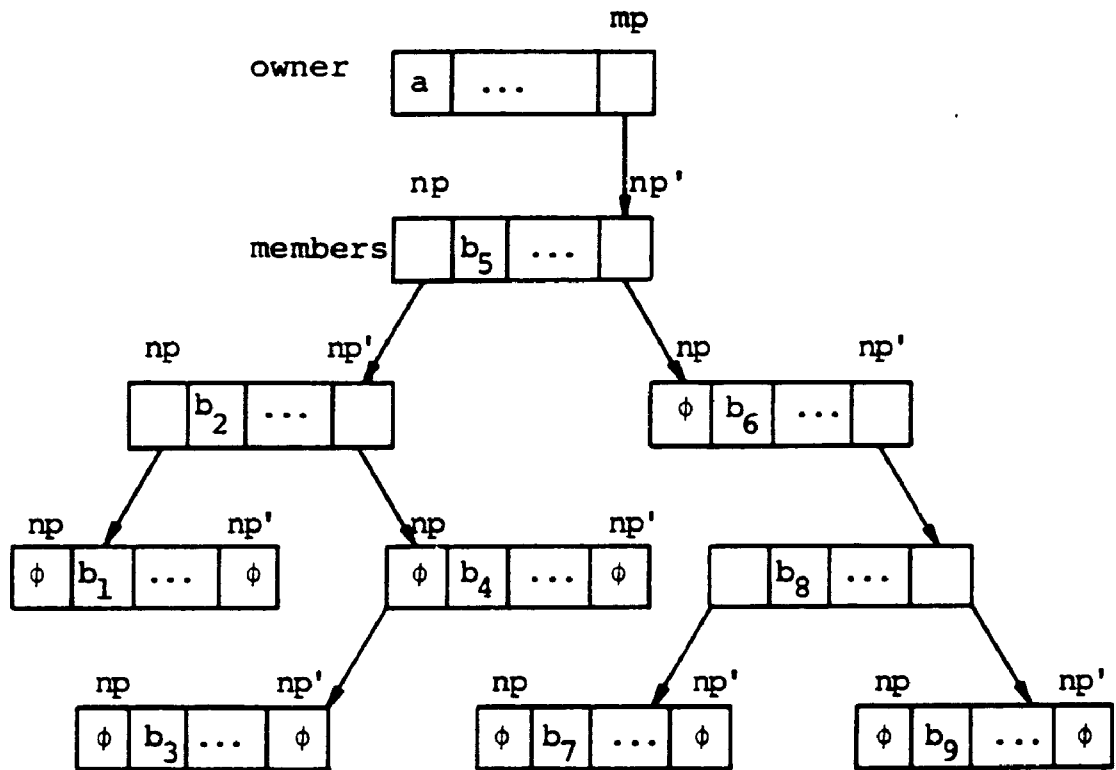
$$\text{if } O_{\ell}^k = \langle \text{op}, A, B, R_j(A, B), f_{\ell}^k \rangle$$

$$\text{and } \text{op} \in \text{SOP}$$

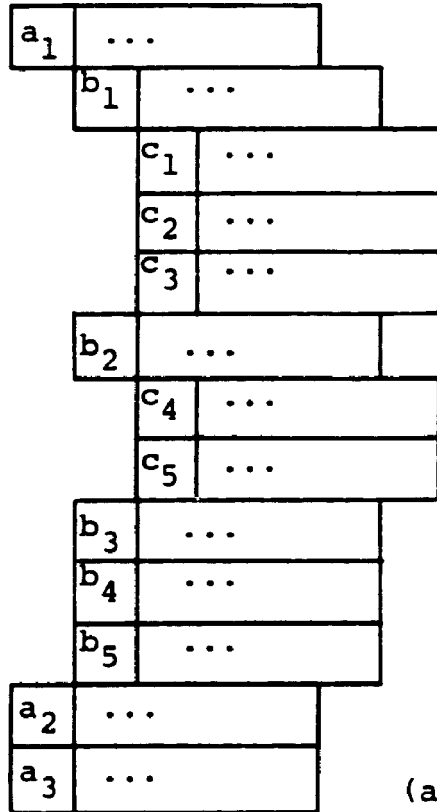
$$= 0 \quad \text{otherwise.}$$

The set of implementations defined and analyzed for storage and time costs in this chapter, is not considered to be exhaustive of all conceivable ways of implementing a relation. Especially in association implementations several other modes of implementation are possible which include simple or multi-level record arrays, Boolean array, packed Boolean array and binary tree. In a record array, association member records of a set are stored in physically sequential locations of the address space, whether or not each member is in turn the owner of some other set. In a multi-level record array association (Figure 3.18(a)), the owner record of a set is followed by its members. Each member in turn, if it is an owner of a set, will be followed by its own members, etc.

In the binary tree form (Figure 3.18(b)) the owner of



(b) Binary Tree Set Structure



(a) Multi-Level Record Array

Figure 3.18

each set points to a member and each member points to two other members or potentially to two subtrees of member records. There are several variations of this scheme. In the Boolean array mode the owner of each set contains (or has a pointer to) an array of bits, the size of which is at least equal to the count of records in the file of member records. Each bit position in the array corresponds to a particular record. A one in a bit position of the bit array of the owner record of a set specifies that the corresponding record is a member of the set. Again other variations of this scheme are possible with the packed Boolean array as an example.

Most of the above mentioned set implementations can be analyzed and incorporated into the relation implementation models of Section 3.1. However, increasing the number of implementation alternatives increases the computation time of the optimization program very rapidly. We have restricted the model to the previously defined 24 relation implementations because it covers a sufficiently large class of implementations used in data base systems.

Again on the subject of association implementations, it is possible that for sets with very large number of members, some sort of indexing would be desirable in certain types of user queries. In fact, any of the 13 association implementations (number 9 to 21) can be indexed or not. Also taking into account the fact that several forms of indexing may be made available, the set of possible relation

implementations would grow substantially in size which in turn can make the optimization program intractable.

The representation of individual data items is considered to be fixed (as specified by user input data) as opposed to being in control of the schema designer. Various transformations could be applied to data item representations which would affect storage/time costs of individual relation implementations and thereby affect the outcome of the optimization problem. These transformations have different effects depending upon the purpose they serve, which may be compaction/decompaction, encoding/decoding, etc. The privacy transformations in most practical cases require some extra space expenditure in addition to the time expenditure of actually performing the transformations. Their value, however, is in the data protection they provide and depends on some external parameters that are not modelled here.

On the other hand, the compaction/decompaction types of transformations are supposed to save storage space with some processing time overhead. In order to incorporate this into the model, some (may be one) alternative transformation options must be specified and analyzed so that costs and values could be determined. Storage savings can easily be absorbed by the previously mentioned storage cost functions, however, the incorporation of the processing times into our time cost functions would be a tedious and an inexact job.

Set types that we consider in this work may have only one member record type. This is not a strong restriction, because the same record type can be declared as owner in many different set types.

Furthermore, we have made the assumption that, within a set, sequential ordering of member records is determined by a key which is one of the data items within the record. The DBTG specifications allow the concatenation of several data item types to be specified as a sort key. Inclusion of this feature in our model would necessitate the knowledge of record structures beforehand which is a part of the design goal.

There are, of course, other factors that may have some effects on the cost function formulations of the preceeding sections. These factors include overflow mechanisms and strategies for reclaiming storage spaces made available by deletions. Simultaneous consideration of these features is beyond the scope of this research.

CHAPTER IV

DATA BASE DESIGN

In this chapter we discuss our data base design methodology. In Section 4.1 we use models of Chapters 2 and 3 to set up an optimization problem. This is done by first identifying, for each relation in the data base, a set of acceptable implementations through the application of constraint conditions developed in 3.2. Then storage and time costs for each of the acceptable implementations of a relation are determined using storage and time cost functions of Sections 3.3 and 3.4, respectively. Next in Section 4.2 we will develop the first phase of the optimization algorithm in which sequences of "undominated choices" are generated using the information about acceptability, storage costs and time costs of relation implementations. In Section 4.3 we describe the second phase of the optimization algorithm in which a discrete multi-stage decision process is formulated and solved through a step-wise generation of "undominated solutions." Solutions to the final stage of the decision process determine a series of optimal configurations from which we derive our optimal data base designs in Section 4.4.

4.1 Problem Set Up

The data base designer provides an application in the following form. A set SI of data items types, containing

NI elements, is provided. For each data item type, its name, size and cardinality must be given. Similarly, a set SR of data base relations is provided by the designer, that contains NR relations. For each relation. its name, cardinality, origin data item type, destination data item type and two multiplicity vectors $\langle m, \bar{m}, M \rangle$ and $\langle n, \bar{n}, N \rangle$ are given. Next a set SU of run units of cardinality NU is given. For each run unit a name and a number that gives the number of operations in the run unit must be specified along with the operations in the run unit. Each operation in turn must have an operation code, an operand data item type, a qualifier data item type, a relation and a frequency of execution. The relation specified in an operation must be a member of the set SR, and it must be defined over the two data items specified as operand and qualifier (in either order) in the operation. The last set of variables that the designer has to specify determines the values of storage space parameters: MSPG, PGSZ, PTRS, CNTRS, and PLOKS.

The above information, provided by the designer in accordance with the models developed in Chapter 2, is used as input to the data base design procedure. For detailed definition and example values of the above parameters, refer to Chapter 2.

The data base design procedure starts with the task of determining the type of each relation. In Chapter 2 we

defined eight types of relations (numbered 0 through 7) and depending on its multiplicity vectors, each relation is of one of the eight types. The knowledge of the type of each relation type simplifies the application of constraint conditions in order to determine the acceptability of implementations for the relation.

The next order of business is to find all acceptable implementation alternatives that can be used correctly to implement each individual relation in SR, from among the 24 possible implementation alternatives defined in Section 3.1. We developed, in Section 3.2, a set of constraint conditions for each implementation alternative that, when satisfied by the parameters specifying a relation, ensure the correctness of the choice of the implementation for that relation.

As discussed in Section 3.2, since the acceptability of some relation implementations depend on the acceptability of others, a certain order must be maintained in the application of constraint conditions. In particular, the acceptability of implementations 5, 6, 7 and 8 must be determined before all other implementations except implementations 1 and 2, because the constraint conditions for those implementations for a relation depend on whether or not implementations 5, 6, 7 and 8 are acceptable for the relation.

The design procedure, therefore, first applies the

constraint conditions of implementations 1 and 2 for all relations in SR and then proceeds with application of the constraint conditions pertaining to implementations 5, 6, 7 and 8 and finally, the constraint conditions of the remaining implementations (3, 4, 9, 10, ... , 24) are applied. It is evident that the inter-dependence of the constraint conditions may result in the exclusion of some otherwise acceptable configurations, but it is essential for the proper execution of our optimization algorithm, that the choice of an implementation for one relation be independent of choices made for other relations. Constraint conditions stated in Section 3.2 are, therefore, only sufficient conditions and they may reduce the size of the solution space in which we are looking for an optimal solution. In that sense, the solution space of our data base design method, although sufficiently large, is smaller than the class of all possible DBTG data base designs.

The information about acceptability of implementations for each relation is recorded by a set of variables $AC(MPL, j)$, where $MPL = 1, 2, \dots, 24$ designates an implementation number and $j = 1, 2, \dots, NR$ designates a relation number. If implementation numbered MPL is acceptable for relation R_j , then $AC(MPL, j) = 1$; otherwise, $AC(MPL, j) = 0$. The set of acceptable implementations for relation $R_j \in SR$ is denoted by $IMP(j)$ and defined as follows:

$$\text{IMP}(j) = \{\text{MPL} \mid \text{MPL} \in \{1, 2, \dots, 24\} \text{ and } \text{AC}(\text{MPL}, j) = 1\}$$

$$j = 1, 2, \dots, \text{NR}$$

Once $\text{IMP}(j)$ is known for each $R_j \in \text{SR}$, we can compute the storage costs of relation implementations. Storage costs are computed using the storage cost functions $\text{SC}(\text{MPL}, j)$, $\text{MPL} \in \{1, 2, \dots, 24\}$, $j = 1, 2, \dots, \text{NR}$ developed in Section 3.3. Out of the maximum storage bound of TMSB characters, only OMSB characters are available for optimization where:

$$\text{OMSB} = \text{TMSB} - \text{MMSB} \quad \text{and}$$

$$\text{MMSB} = \sum_{i=1}^{\text{NI}} \text{ITSZ}(i) * \text{ITCR}(i), \quad \text{where for } i = 1, 2, \dots, \text{NR}$$

$$\text{ITCR}(i) = |I_i| \text{ and } \text{ITSZ}(i) = \text{size of } I_i$$

The next step of the data base design procedure concerns the computation of time costs. Time costs functions for each relation implementation $\langle \text{MPL}, R \rangle$ in all possible kinds of operations were developed in Section 3.4. Operations were defined in Chapter 2 in the following form:

$O = \langle \text{op}, A, B, R, f \rangle$, where $\text{op} \in \text{SOP}$ is an operation code in the set of all possible operation codes

$$\text{SOP} = \{\text{FIND FIRST}, \text{FIND LAST}, \text{FIND ITH}, \text{FIND NEXT}, \text{FIND PREV.}, \text{FIND ALL}, \text{FIND KEY}, \text{STORE FIRST}, \text{STORE LAST}, \text{STORE KEY}, \text{MODIFY}, \text{ERASE}\}, R \in \text{SR} \text{ is a relation on } A \in \text{SI} \text{ and } B \in \text{SI} \text{ and } f \text{ is the operation's frequency. Depending on whether } A = \text{ORG}(R) \text{ and } B = \text{DST}(R) \text{ or } B = \text{ORG}(R) \text{ and}$$

$A = \text{DST}(R)$, implementation MPL for relation R has a different time cost function in operation O given above.

Since the operation code op in O can be anyone of the 12 operation codes in SOP and there is a total of 24 implementation alternatives, it appears that there is a large number, namely 576, of time cost functions to deal with. This is not true because a large number of these functions are similar in form and different only in parameters.

For example for $O = \langle \text{FIND ALL}, B, A, R_j, f \rangle$ and $\text{MPL} = 15$ the time cost function is

$$\text{TC}(\text{MPL}, j, O) = f(p_1 + p_3(R_j) + (\bar{m}_j - 1) p_2(R_j) + 3p_4)$$

and for the same operation O and $\text{MPL} = 16$ the time cost function is given by

$$\text{TC}(\text{MPL}, j, O) = f(p_1 + p_3(R_j) + (\bar{n}_j - 1) p_2(R_j) + 3p_4).$$

When all such similarities in time cost functions are considered, there are only 17 distinctly different functions that are used in time cost computations.

At this stage in the data base design procedure, we have, for each relation $R_j \in \text{SR}$, $j = 1, \dots, \text{NR}$ the following information:

- 1) A set of 24 variables $\text{AC}(\text{MPL}, j)$, $\text{MPL} = 1, 2, \dots, 24$ where $\text{AC}(\text{MPL}, j) = 1$ if implementation MPL is acceptable for R_j , $= 0$ otherwise.
- 2) A set $\text{IMP}(j)$ of implementation numbers MPL such that $\text{AC}(\text{MPL}, j) = 1$ iff $\text{MPL} \in \text{IMP}(j)$.
- 3) A storage cost $\text{SC}(\text{MPL}, j)$ associated with each

implementation $MPL \in IMP(j)$ for relation R_j .

4) A time cost $TC(MPL, j)$ associated with each implementation $MPL \in IMP(j)$ for relation R_j .

In Chapter 3 we defined a configuration C for SR to be a set of ordered pairs of relation implementations where there is one and only one element $\langle R_j, MPL_j \rangle \in C$ for each and every element $R_j \in SR$. In set theoretic terminology, a configuration C for a set of data base relations SR is a function from SR to the set of integers $\{1, 2, \dots, 24\}$. A configuration C for SR is called an acceptable configuration (for SR) if for all $\langle R_j, MPL_j \rangle \in C$, $AC(MPL_j, j) = 1$. For such configurations, we also defined storage and time costs, respectively, as follows:

$$\begin{aligned}
 CS(C) &= \sum_{j=1}^{NR} SC(MPL_j, j) \\
 CT(C) &= \sum_{j=1}^{NR} TC(MPL_j, j) \quad (\text{Note: } NR = |SR| = |C|)
 \end{aligned}$$

In the above two expressions $SC(MPL_j, j)$ and $TC(MPL_j, j)$ are the storage and time costs of relation implementation $\langle R_j, MPL_j \rangle \in C$, respectively.

The problem is, now, to find an acceptable configuration C^* for SR for some given storage limit $L \leq TMSB$, where C^* is defined as follows:

$$(4.1.1) \quad CS(C^*) \leq L - MMSB$$

$$(4.1.2) \quad CT(C^*) \leq CT(C) \text{ for all acceptable } C$$

configurations C for which $CS(C) \leq L - MMSB$.

If more than one configuration with different storage costs but equal time costs satisfy (4.1.1) and (4.1.2), then C^* is, by definition, the configuration with the lowest storage cost. If, however, more than one configuration with equal storage and time costs satisfy (4.1.1) and (4.1.2), then the choice of the optimal configuration among those configurations is arbitrary.

We call C^* an "optimal" configuration. The optimality of C^* is, however, within a sub-class of DBTG structures because of the interdependence of constraint conditions we discussed earlier in this section. It is also qualified by the observations we will make in Chapter 5 regarding errors in time cost computations.

The existence of an optimal configuration depends on two factors. The first factor is the existence of at least one acceptable implementation for each relation in SR and it is determined by the constraint conditions only. In terms of the above variables we must have:

$$|IMP(j)| \geq 1 \quad \text{for all } R_j \in SR \quad (4.1.3)$$

The second factor is the value of the storage limit L .

It can be shown that even when (4.1.3) is true, there exists a limit L_0 such that for any $L < L_0$ no optimal configuration exists. The fact that there exists a minimum L_0 on L is obvious because the set of acceptable configurations is a finite set and the cumulative storage

cost $CS(C)$ is a real function defined for the elements of that set. The value of L_0 can be derived as follows. Let MPL_j^- , $j = 1, 2, \dots, NR$ be the implementation with the smallest storage cost for R_j , i.e.

$$SC(MPL_j^-, j) = \min_{MPL_j \in IMP(j)} SC(MPL_j, j), \quad j = 1, 2, \dots, NR \quad (4.1.4)$$

The existence of MPL_j^- , $j = 1, 2, \dots, NR$ is guaranteed by (4.1.3). Then L_0 is given by the following:

$$L_0 = MMSB + \sum_{j=1}^{NR} SC(MPL_j^-, j)$$

If (4.1.3) holds and $L \geq L_0$, however, an optimal configuration C^* (i.e. an acceptable configuration that satisfies (4.1.1) and (4.1.2)) exists because the set of acceptable configurations is finite. C^* can be determined by the following method:

- 1) Generate all acceptable configurations for SR. There is at least one such configuration because (4.1.3) is assumed to be true. There are $\prod_{j=1}^{NR} |IMP(j)|$ such configurations.
- 2) Compute $CS(C)$ for all configurations C generated in 1).
- 3) Retain all configurations C for which $CS(C) \leq L - MMSB$ and discard all those for which $CS(C) > L - MMSB$. The retained set of configurations is not an empty set because we assumed $TMSB \geq L \geq L_0$ and every configuration in the set satisfies (4.1.1).

4) Compute $CT(C)$ for all C in the set of configurations retained in 3).

5) C^* is a configuration in the retained set that has minimum time cost (thereby (4.1.2) is satisfied).

The optimal configuration, however, is not necessarily unique because it is possible for two configurations C_1 and C_2 to have identical storage costs and identical time costs, i.e. $CS(C_1) = CS(C_2)$ and $CT(C_1) = CT(C_2)$. In our implementation of the optimization algorithm, the choice of the optimal configuration when two or more configurations satisfy the conditions of optimality is arbitrary.

The method described above for finding C^* is not practical because of the large number of configurations that must be generated in step 1). In our application example of Chapter 2, we have 26 relations in SR and the cardinality of the set $IMP(j)$ for $j = 1, 2, \dots, 26$ is tabulated in Table 4.1.1.

j	$ IMP(j) $	j	$ IMP(j) $
1	13	8	4
2	13	9	4
3	9	10	9
4	13	11	5
5	13	12	9
6	10	13	12
7	4	14	13

Table 4.1.1 Number of acceptable implementations for R_j

j	IMP(j)	j	IMP(j)
15	12	21	12
16	11	22	9
17	12	23	5
18	13	24	13
19	11	25	13
20	12	26	9

Table 4.1.1 (continued)

The number of acceptable configurations to be generated in step 1) of the above method is $\prod_{j=1}^{NR} |\text{IMP}(j)|$. For our example application, this number is over 2.32×10^{25} . Even if we could examine 10^9 configurations per second, it would take more than 7×10^8 years to examine that many configurations.

In order to get around this problem one can make use of the dynamic programming concepts in the manner discussed below. We first define a partial configuration $\text{PC}(\text{SSR})$ for the set of relations $\text{SSR} \subseteq \text{SR}$ as the following function:

$$\begin{aligned} \text{PC}(\text{SSR}) : \text{SSR} &\rightarrow \{1, 2, \dots, 24\} \\ \text{such that if } \langle R_i, \text{MPL}_i \rangle &\in \text{PC}(\text{SSR}) \\ \text{then } \text{AC}(\text{MPL}_i, i) &= 1. \end{aligned}$$

For each subset SSR of SR consisting of relations $R_{i_1}, R_{i_2}, \dots, R_{i_{|\text{SSR}|}}$, $\text{PC}(\text{SSR})$ determines a partial assignment of acceptable implementations for relations in SSR . Note that $\text{PC}(\text{SR})$ is an acceptable configuration for

SR as defined earlier in this section. We can similarly define cumulative storage and time costs for $PC(SSR)$, $SSR \subseteq SR$ as follows:

If $SSR = \{R_{i_1}, R_{i_2}, \dots, R_{i_j}\}$ where $j = |SSR|$ and

$PC(SSR) = \{ \langle R_{i_k}, MPL_{i_k} \rangle \mid k = 1, 2, \dots, j \}$, then

$$CS(PC(SSR)) = \sum_{k=1}^j SC(MPL_{i_k}, i_k) \quad \text{and}$$

$$CT(PC(SSR)) = \sum_{k=1}^j TC(MPL_{i_k}, i_k),$$

where for $k = 1, 2, \dots, j$, $SC(MPL_{i_k}, i_k)$ and $TC(MPL_{i_k}, i_k)$ are storage and time costs of implementation MPL_{i_k} for R_{i_k} , respectively.

In the special case where $SSR = \{R_{i_k} \mid k = 1, 2, \dots, j\}$ is the set of the first j relations in SR, namely if $i_k = k$ for $k = 1, 2, \dots, j$ we will refer to $PC(SSR)$ simply as $PC(j)$.

Let $SPC_j(S)$ be the set of all $PC(j)$ such that $CS(PC(j)) \leq S$. The optimal (or objective) value (partial) function $OVF_j(S)$ is defined by (4.1.5) below if $SPC_j(S)$ is non-empty. It is undefined otherwise.

$$OVF_j(S) = \min CT(PC(j)) \quad (4.1.5)$$

$$PC(j) \in SPC_j(S)$$

For each pair $\langle j, S \rangle$, $OVF_j(S)$ is called the optimal value function, or the optimal j -stage solution to state S . The partial configuration $PC^*(j) \in SPC_j(S)$ that satisfies (4.1.5) determines the partial assignment of imple-

mentations to relations $R_1, R_2, \dots, R_j$. The effect of this partial policy (j -stage solution) on later decisions is summarized in S . In other words, if a total of s bytes of storage space is required for the first j relation implementations, then $(L - \text{MMSB} - S)$ bytes remain for the implementation of $R_{j+1}, R_{j+2}, \dots, R_{NR}$. This is the entire information needed from the first j -stage decision to make later decisions and since S summarizes the entire past history in the decision process, we call it the state variable.

In the case of our problem, Bellman's principle of optimality [Bellman 1957] can be stated as follows:

If an optimal configuration

$C^* = \{ \langle R_1, \text{MPL}_1^* \rangle, \langle R_2, \text{MPL}_2^* \rangle, \dots, \langle R_{NR}, \text{MPL}_{NR}^* \rangle \}$ for SR has a partial storage cost of $S_1 = \text{CS}(\text{PC}^*(NR - 1))$ for the partial configuration $\text{PC}^*(NR - 1) = C^* - \{ \langle R_{NR}, \text{MPL}_{NR}^* \rangle \}$, then $\text{PC}^*(NR - 1)$ must be an optimal configuration for $SR - \{R_{NR}\}$ subject to a storage cost limit of S_1 . Or stated differently, suppose an optimal policy for all NR stages (stages are relations in this case) has a storage cost of S_1 bytes for the first $(NR - 1)$ stages. Then it must have made the optimal selection of implementations for relations $R_1, R_2, \dots, R_{NR - 1}$ subject to a storage constraint of S_1 . This principle can be seen easily by the following formal reasoning.

Suppose C^* is an optimal configuration that selects

implementation MPL_{NR}^* for R_{NR} ($\langle R_{NR}, MPL_{NR}^* \rangle \in C^*$). Suppose further that the partial configuration $PC^*(NR - 1) = C^* - \{\langle R_{NR}, MPL_{NR}^* \rangle\}$ is not optimal. Then an optimal partial configuration $PC^+(NR - 1) = \{\langle R_1, MPL_1^+ \rangle, \langle R_2, MPL_2^+ \rangle, \dots, \langle R_{NR-1}, MPL_{NR-1}^+ \rangle\}$ must exist whose time cost is less than the time cost of $PC^*(NR - 1)$. The configuration $C^+ = PC^+(NR - 1) \cup \{\langle R_{NR}, MPL_{NR}^* \rangle\}$ should then have a time cost that is less than $CT(C^*)$. This contradicts the assumption that C^* was an optimal configuration.

The above principle leads to the following algorithm:

Dynamic Programming Algorithm:

/* Initialization */

For $S = 1$ until $OMSB$ do;

$OVF_1(S) = \min TC(MPL_1, 1) ;$

$MPL_1 \in IMP(1) \ \& \ SC(MPL_1, 1) \leq S$

end;

/* */

/* Iteration */

For $j = 2$ until NR do ;

For $S = 1$ until $OMSB$ do ;

$OVF_j(S) = \min [OVF_{j-1}(S - SC(MPL_j, j)) + TC(MPL_j, j)] ;$

$MPL_j \in IMP(j)$

end;

end;

/* */

The final goal is to determine the optimal value function for $j = \text{NR}$, namely $\text{OVF}_{\text{NR}}(S)$. $\text{OVF}_{\text{NR}}(S)$ can determine the value of the time cost of the optimal configuration from which individual relation implementations (in the configuration) may be traced back (from NR to 1). There is clearly another step that must be added to the iteration part of the algorithm to keep track of the optimal implementation selected at each S and each j in order that it may be output as the solution. Note further that for a storage limit L , $L_0 \leq L \leq \text{TMSB}$ the optimal value function $\text{OVF}_{\text{NR}}(L - \text{MMSB})$ does not necessarily give the time cost of the optimal configuration C^* but it has to be derived from the following:

$$\text{CT}(C^*) = \min \text{OVF}(S)$$

$$S \leq L - \text{MMSB}$$

The algorithm as stated above, however, requires $|\text{IMP}(j)|$ subtractions, additions and comparisons at iteration j for computing $\text{OVF}_j(S)$ for each S . The amount of computation at iteration j is then proportional to $\text{OMSB} * |\text{IMP}(j)|$. Total computation for all NR iterations (initialization included) is then proportional to $\text{OMSB} * \sum_{j=1}^{\text{NR}} |\text{IMP}(j)|$.

The key difference between the complexities of the dynamic programming approach and enumeration is that in the former the complexity is proportional to $\sigma = \sum_{j=1}^{\text{NR}} |\text{IMP}(j)|$

and in the latter, it is proportional to $\pi = \prod_{j=1}^{NR} |\text{IMP}(j)|$ which is clearly a considerable difference. In our application example $\pi \approx 2.32 \times 10^{25}$ and $\sigma = 263$. Although the dynamic programming algorithm stated above achieves considerable savings in computation time as compared to enumeration, it still requires a relatively large number of steps to be considered. This is because at each iteration j , $\text{OVF}_j(S)$ must be computed for each S from 1 to OMSB and this introduces the factor OMSB in the expression for the complexity of the algorithm. We recall that OMSB is the amount of storage available for optimization given by $\text{OMSB} = \text{TMSB} - \text{MMSB}$. This value is on the order of a million (characters). One remedy for this problem is to reduce the number of distinct S values considered at each iteration to a given maximum of say 1000 points by scaling all storage costs and rounding-off the results to the nearest integer (see [Mitoma 1975]). That is to say, if OMSB is greater than 1000 characters (which almost always is the case), then reduce every storage cost, say S , to the closest integer to $(1000/\text{OMSB}) * S$. This is equivalent to measuring storage costs in blocks of $(\text{OMSB}/1000)$ characters rather than in blocks of one character and also by replacing OMSB with 1000 in the algorithm. The rescaling method has the side effect of introducing round-off errors whose effects on the outcome of the optimization are not exactly known.

In the following two sections we introduce an algorithm that solves our optimization problem in considerably fewer steps without the requirement of rescaling storage costs.

4.2 Undominated Choices

Definition 4.2.1

Let MPL be an acceptable implementation for relation R_j , i.e. $AC(MPL, j) = 1$. The storage and time costs of implementation MPL for relation R_j are, then, defined and known. If $S = SC(MPL, j)$ and $T = TC(MPL, j)$, then we shall call the triple $\langle MPL, S, T \rangle$ a choice for the relation R_j . $\square$

Definition 4.2.2

Let $\langle MPL, S, T \rangle$ and $\langle MPL', S', T' \rangle$ be two choices for R_j such that $S \leq S'$, then $\langle MPL, S, T \rangle$ is said to dominate $\langle MPL', S', T' \rangle$ if $T \leq T'$. If there are two or more choices for a relation such that their second and third coordinates are respectively equal, then all but one are arbitrarily selected as dominated. $\square$

The motivation for this definition of dominance becomes apparent by the following observation:

Observation 4.2.1

If C^* is the optimal configuration for SR with a storage limit of L , i.e. if C^* satisfies (4.1.1) and (4.1.2), then for no relation implementation pair $\langle R_j, MPL_j^* \rangle \in C^*$, there exists a choice $\langle MPL, S, T \rangle$ for

R_j , that dominates $\langle \text{MPL}_j^*, S^*, T^* \rangle$ and $\text{MPL} \neq \text{MPL}_j^*$ where $S^* = \text{SC}(\text{MPL}_j^*, j)$ and $T^* = \text{TC}(\text{MPL}_j^*, j)$.

Proof: If $\langle \text{MPL}, S, T \rangle$ dominates $\langle \text{MPL}_j^*, S^*, T^* \rangle$, then by definition we must have $S \leq S^*$ and $T \leq T^*$. Let C^+ be the configuration obtained from C^* by replacing $\langle R_j, \text{MPL}_j^* \rangle$ with $\langle R_j, \text{MPL} \rangle$, i.e. $C^+ = (C^* - \{\langle R_j, \text{MPL}_j^* \rangle\}) \cup \{\langle R_j, \text{MPL} \rangle\}$. If $T < T^*$ and $S \leq S^*$, then $\text{CT}(C^+) < \text{CT}(C^*)$ and $\text{CS}(C^+) \leq \text{CS}(C^*)$ which in turn implies that C^+ has a lower time cost than C^* ; a contradiction to the assumption that C^* was optimal. If $T = T^*$ and $S < S^*$, then C^+ and C^* have equal time costs but C^+ has a lower storage cost than S^* , which by definition, makes C^+ optimal, a contradiction again. Note that if $T = T^*$ and $S = S^*$, then C^+ and C^* have equal storage and time costs; they are both optimal. The argument still holds because the choice of the optimal configuration is, by definition, arbitrary. $\square$

It follows from Observation 4.2.1 that no implementation will ever be selected for any relation if the implementation's number is the first coordinate of a dominated choice for that relation. It is, therefore, desirable to exclude those implementations (for each relation) that are associated with dominated choices from consideration as potential implementation alternatives for the relation.

Definition 4.2.3

If there are two choices $\langle \text{MPL}, S, T \rangle$ and $\langle \text{MPL}', S', T' \rangle$ for relation R_j such that neither one

dominates the other, then the two choices are said to be a pair of undominated choices for R_j .

Observation 4.2.2

Let $q = \langle \text{MPL}, S, T \rangle$ and $q' = \langle \text{MPL}', S', T' \rangle$ be a pair of undominated choices for relation R . Then the following are true:

- i) $S \neq S'$ and $T \neq T'$
- ii) $S < S'$ iff $T > T'$
- iii) $S > S'$ iff $T < T'$

Proof: q and q' form a pair of undominated choices for relation R iff the following is true:

$$\neg (S \leq S' \wedge T \leq T') \wedge \neg (S' \leq S \wedge T' \leq T)$$

where $\wedge$, $\vee$ and $\neg$ are the usual logical AND, OR and NOT operations, respectively, with the conventional precedence relations. The above logical expression is equivalent to:

$$((S' < S) \wedge (T < T')) \vee ((S < S') \wedge (T' < T))$$

From this expression, the results i), ii) and iii) follow. □

Definition 4.2.4

A set of choices $\{\langle \text{MPL}_1, S_1, T_1 \rangle, \dots, \langle \text{MPL}_p, S_p, T_p \rangle\}$ is said to be an undominated set of choices for a data base relation R if every pair of elements of the set is a pair of undominated choices for R . □

Let Q be a set of undominated choices for relation R , where $Q = \{\langle \text{MPL}_i, S_i, T_i \rangle \mid i = 1, \dots, p\}$. Since by

definition for all $i \neq j \in \{1, \dots, p\}$, $\langle \text{MPL}_i, S_i, T_i \rangle$ and $\langle \text{MPL}_j, S_j, T_j \rangle$ form a pair of undominated choices, we know from part i) of Observation 4.2.2 that for $i \neq j \in \{1, 2, \dots, p\}$, $S_i \neq S_j$ and $T_i \neq T_j$. We shall, therefore, assume, with no loss of generality, that $S_1 < S_2 < \dots < S_p$, or equivalently, for $i = 1, 2, \dots, p-1$, $S_i < S_{i+1}$. From part ii) of Observation 4.2.2, it follows that for $i = 1, \dots, p-1$, $T_i > T_{i+1}$.

Theorem 4.2.1

Let $Q = \{q_i \mid q_i = \langle \text{MPL}_i, S_i, T_i \rangle, i = 1, \dots, p\}$ be a set of undominated choices for relation R with the property that $S_i < S_{i+1}$ (and thereby $T_i > T_{i+1}$) for all $i = 1, 2, \dots, p-1$. Let $q = \langle \text{MPL}, S, T \rangle$, $\text{MPL} \neq \text{MPL}_i$, $i = 1, 2, \dots, p$, be another choice for R such that $S > S_p$. Then q is pairwise undominated with q_i , $i = 1, 2, \dots, p$ iff q and q_p is a pair of undominated choices for R .

Proof: The "only if" part of the proof is trivial. In order to prove the "if" part, we first observe that q cannot dominate any $q_i \in Q$. This is because $S > S_p$ which implies $S > S_i$, $i = 1, \dots, p$. (See Definition 4.2.2). We now have to show that no $q_i \in Q$ dominates q to complete the proof. Suppose $q_j \in Q$ dominates q , for some $j \in \{1, 2, \dots, p-1\}$. Note that $j \neq p$ because q and q_p form a pair of undominated choices, by assumption.

Since $j < p$, we know that $S_j < S_p$, which in conjunction with $S_p < S$ implies $S > S_j$. $S > S_j$ in conjunction with the assumption that q_j dominates q implies $T \geq T_j$ (Definition 4.2.2). But again since $j < p$, we know that $T_j > T_p$, and hence, $T > T_p$. Referring back to Definition 4.2.2, it is clear that q_p must dominate q (because we arrived at $T_p < T$ and $S_p < S$), a contradiction. $\square$

A useful corollary of the above theorem leads us to an efficient algorithm for generating an ordered set of undominated choices for each relation in SR.

Corollary 4.2.1

Let $Q = \{q_i \mid q_i = \langle \text{MPL}_i, S_i, T_i \rangle, i = 1, 2, \dots, p-1\}$ be a set of undominated choices for relation R with the property that $S_i < S_{i+1}$ (and, therefore, $T_i > T_{i+1}$) for all $i = 1, 2, \dots, p-2$. Let $q_p = \langle \text{MPL}_p, S_p, T_p \rangle$ be another choice for R such that $S_p \geq S_{p-1}$. Let Q^+ be an undominated set of choices for R involving the maximum number of elements of $Q \cup \{q_p\}$, such that for all $q \in Q \cup \{q_p\} - Q^+$, q is dominated by some $q' \in Q^+$. Then:

- i) $Q^+ = Q \cup \{q_p\}$ iff $S_p > S_{p-1}$ and $T_p < T_{p-1}$
- ii) $Q^+ = Q \cup \{q_p\} - \{q_{p-1}\}$ if $S_p = S_{p-1}$ and $T_p < T_{p-1}$
- iii) $Q^+ = Q$ or $Q^+ = Q \cup \{q_p\} - \{q_{p-1}\}$ if $S_p = S_{p-1}$ and $T_p = T_{p-1}$
- iv) $Q^+ = Q$ otherwise.

Proof: i): The "if" part is a direct consequence of Theorem 4.2.1 because $(S_p > S_{p-1}$ and $T_p < T_{p-1})$ implies that q_p and q_{p-1} is an undominated pair. Maximality of Q^+ is clear because in this case $Q^+ = Q \cup \{q_p\}$. For the "only if" part, we observe that since Q^+ is undominated, hence q_{p-1} and q_p is an undominated pair, and since $S_p \geq S_{p-1}$ is assumed, it follows from Observation 4.2.2 that $S_p > S_{p-1}$ and $T_p < T_{p-1}$.

ii): Suppose $S_p = S_{p-1}$ and $T_p < T_{p-1}$. These two relations in conjunction with $S_{p-2} < S_{p-1}$ and $T_{p-1} < T_{p-2}$ (known from definition of Q) result in:

$$S_{p-2} < S_p \text{ and } T_{p-2} > T_p.$$

It follows from the latter two relations that q_p and q_{p-2} is an undominated pair of choices for R . Then by Theorem 4.2.1, $Q \cup \{q_p\} - \{q_{p-1}\}$ is an undominated set of choices for R . Furthermore, q_{p-1} is the only member of the set $Q \cup \{q_p\} - Q^+$ and it is dominated by $q_p \in Q^+$. The set of choices $Q \cup \{q_p\}$ is not undominated. At least one of its elements must be deleted so that the remaining elements can form an undominated set. Since Q^+ is obtained from $Q \cup \{q_p\}$ by deleting q_{p-1} , maximality of Q^+ is established.

iii): The proof of this part is trivial.

iv): Since $S_p \geq S_{p-1}$ is assumed, the only remaining cases are when either (4.2.1) or (4.2.2) is true in both of which cases q_{p-1} dominates q_p . Again

maximality of Q^+ is clear because q_p is the only member of the set $Q \cup \{q_p\} - Q^+$, it is dominated by $q_{p-1} \in Q$ and at least one element of $Q \cup \{q_p\}$ must be discarded so that the remaining elements can form an undominated set of choices for R. $\square$

$$S_p > S_{p-1} \quad \wedge \quad T_p \geq T_{p-1} \quad (4.2.1)$$

$$S_p = S_{p-1} \quad \wedge \quad T_p > T_{p-1} \quad (4.2.2)$$

The following algorithm follows immediately from the above corollary. The purpose of the algorithm is to generate (logically) a set of undominated choices ordered in the increasing sequence of storage costs, for each relation $R_j \in SR$.

Algorithm 4.2.1

For $j = 1$ until NR do;

1. Sort all $p' = |IMP(j)|$ choices for relation R_j in nondecreasing sequence of storage costs (2nd Coordinate). In case of equal storage costs, sort in nondecreasing sequence of time costs (3rd Coordinate). Let $Q' = \{q_1, q_2, \dots, q_{p'}\}$ be the ordered set of choices for R_j so obtained.
2. Initialize the (ordered) set Q_j of maximal undominated choices for R_j to the empty set. Initialize Best Time Cost BTC to infinity.
3. For $p = 1$ until p' do;

```

Let  $T_p$  be the time cost (3rd Coordinate) of
 $q_p \in Q'$ ;
If  $T_p < BTC$  then do;
 $Q_j = Q_j \cup \{q_p\}$ ; make  $q_p$  the last element
of the new  $Q_j$ ;
 $BTC = T_p$ ;
end;

end;

end;

```

The sorting of $p' = |\text{IMP}(j)|$ elements in step 1 has a time complexity proportional to p'^2 . The complexity of the loop in step 3 is proportional to p' . The algorithm then has a total time complexity proportional to $NR * f(p'')$ where $f(p'')$ is a polynomial of degree two in $p'' = \text{Avg } |\text{IMP}(j)|$ over j from 1 to NR .

Algorithm 4.2.1 constitutes the first phase of our optimization process. We recall from Section 4.1 that a necessary condition for the existence of an optimal configuration is the following: $|\text{IMP}(j)| \geq 1$ $\forall j \in \{1, 2, \dots, NR\}$. This condition guarantees that there exists at least one choice for every relation $R_j \in SR$. In general, there are more than one acceptable implementations (and thereby choices) for a relation. However, it may be the case that for some relation R_j , there exists a choice which dominates all other choices. In such a case the maximal set of undominated choices Q_j for

relation R_j contains only one element, say $q_j = \langle \text{MPL}_j, S_j, T_j \rangle$. It is clear that the optimal choice of an implementation for R_j in this case is MPL_j and that relation R_j may be excluded from consideration by the second phase of optimization. The storage and time costs of all such relations will be accumulated and denoted by BASC and BATC , respectively.

Thus, after the first phase of optimization is completed, we have a number $\text{MR} \leq \text{NR}$ of relations that constitute MR stages of a decision process to be solved in the second phase of optimization. Each stage k , $k = 1, \dots, \text{MR}$, has the following data:

$\text{RL}(k)$ is the index $j \in \{1, 2, \dots, \text{NR}\}$ of $R_j \in \text{SR}$ which is the relation associated with stage k .

$Q_{\text{RL}(k)} \stackrel{d}{=} Q^k$ is the maximal set of undominated choices for stage k , ordered in the increasing sequence of storage costs.

$\text{NCHS}(k)$ is the number of elements of $Q_{\text{RL}(k)}$; i.e. the maximal number of undominated choices for stage k .

$\text{FMP}(i, k)$ is the implementation number associated with (the first coordinate of) the i -th choice in $Q_{\text{RL}(k)}$ for $i = 1, 2, \dots, \text{NCHS}(k)$.

Clearly, the information content of $Q_{RL}(k)$,
 $k = 1, \dots, MR$ can be derived from $FMP(i, k)$,
 $i = 1, 2, \dots, NCHS(k)$, $k = 1, 2, \dots, MR$ and $RL(k)$
 using $SC(MPL, j)$ and $TC(MPL, j)$ tables derived earlier.
 For example the i -th choice of k -th stage is:

$$q_i^k = \langle FMP(i, k), SC(FMP(i, k), RL(k)), TC(FMP(i, k), RL(k)) \rangle$$

$$\text{or equivalently } q_i^k = \langle MPL_i^k, S_i^k, T_i^k \rangle$$

$$\begin{aligned} \text{where } MPL_i^k &= FMP(i, k) \\ S_i^k &= SC(MPL_i^k, j) \\ T_i^k &= TC(MPL_i^k, j) \\ j &= RL(k) \end{aligned}$$

The matrix FMP is generated by phase 1 and used as input
 to the second phase of optimization. We observe that since
 all relations for which $|Q_j| = 1$ are discarded by the
 second phase, $NCHS(k)$ is clearly greater than or equal to
 2. $BASC$ and $BATC$ are given by the following:

$$BASC = \sum_{j=1}^{NR} SC(MPL_j^1, j) * d_j,$$

$$BATC = \sum_{j=1}^{NR} TC(MPL_j^1, j) * d_j$$

$$\text{where } d_j = \begin{cases} 1 & \text{if } |Q_j| = 1 \\ 0 & \text{otherwise} \end{cases}$$

and MPL_j^1 is the first coordinate of the first choice
 in Q_j .

We can now identify two special configurations C_m and C_M
 which have the smallest and largest storage costs, re-
 spectively.

$$C_m = \{ \langle R_j, MPL_j \rangle \mid j = 1, 2, \dots, NR, R_j \in SR, \\ MPL_j = \text{first coordinate of the} \\ \text{first element of } Q_j \}$$

$$C_M = \{ \langle R_j, MPL_j \rangle \mid j = 1, 2, \dots, NR, R_j \in SR, \\ MPL_j = \text{first coordinate of the} \\ \text{last element of } Q_j \}.$$

C_M and C_m are also the configurations with the smallest and largest time costs, respectively. Increasing the storage limit L above $MMSB + CS(C_M)$ does not change the optimal solution, C_M . Also if $L < MMSB + CS(C_m)$, there will be no optimal solutions. Note that $L_0 = MMSB + CS(C_m)$ where L_0 was discussed in Section 4.1. The storage and time costs of C_M and C_m can be computed from the following:

$$CS(C_m) = BASC + \sum_{k=1}^{MR} SC(FMP(1, k), RL(k))$$

$$CT(C_m) = BATC + \sum_{k=1}^{MR} TC(FMP(1, k), RL(k))$$

$$CS(C_M) = BASC + \sum_{k=1}^{MR} SC(FMP(NCHS(k), k), RL(k))$$

$$CT(C_M) = BATC + \sum_{k=1}^{MR} TC(FMP(NCHS(k), k), RL(k))$$

Let FSR be the subset of the set of data base relations SR consisting of all relations R_j for which $|IMP(j)| = 1$, i.e. $FSR = \{R_j \mid R_j \in SR \text{ and } |IMP(j)| = 1\}$. The optimal partial configuration for FSR is denoted by FPC and it is given by

$$FPC = \{ \langle R_j, MPL_j \rangle \mid R_j \in FSR \text{ and } MPL_j \text{ is the first}$$

coordinate of the first element of Q_j }.

Clearly BASC and BATC are storage and time costs of FPC, respectively. The set FSR is the set of relations in SR for which the optimal implementation is fixed. The complement of this set with respect to SR, denoted VSR is given by:

$$VSR = SR - FSR = \{R_j \mid j = RL(k), k = 1, 2, \dots, MR\}.$$

For each storage limit value L , the optimal configuration C for SR is the union of two sets FPC, the optimal partial configuration for FSR, and VPC^* , the optimal partial configuration for VSR. Where VPC is defined as follows:

$$CS(VPC^*) \leq L - BASC - MMSB \quad (4.2.3)$$

$$CT(VPC^*) \leq CT(VPC) \quad \text{for all partial} \quad (4.2.4)$$

configuration VPC for VSR such
that $CS(VPC) \leq L - BASC - MMSB$

The task of the second phase of optimization is to find VPC^* , taking advantage of the fact (see Observation 4.2.1) that for every relation $R_{RL(k)}$, $k = 1, 2, \dots, MR$, only $NCHS(k)$ implementations are candidates for the optimal choice, namely those associated with the undominated choices in $Q_{RL(k)} = Q^k$.

4.3 Undominated Solutions

In this section we shall present the algorithm used in the second phase of the optimization. This algorithm takes the results of the first phase as input and develops, as output, a series of optimal configurations, one for each

storage limit range. Before discussing the algorithm, however, we shall define a few terms.

Definition 4.3.1

Let SSR_k be a subset of VSR defined as follows:

$$SSR_k = \{R_{RL(p)} \mid R_{RL(p)} \in VSR \text{ and } p = 1, 2, \dots, k\}, \\ k = 1, 2, \dots, MR$$

Let a partial configuration PC^k for SSR_k be defined as follows:

$$PC^k = \{ \langle R_{RL(p)}, MPL_p \rangle \mid R_{RL(p)} \in SSR_k, \\ MPL_p = FMP(i_p, p) \text{ for some} \\ 1 \leq i_p \leq NCHS(p), \\ p = 1, 2, \dots, k\}, \\ k = 1, 2, \dots, MR$$

If S^k, T^k are storage and time costs of PC^k , respectively, then we shall call the triple $\langle PC^k, S^k, T^k \rangle$ a k -stage solution (or a solution to stage k). $\square$

Note that in Definition 4.3.1, SSR_k is a set composed of the relations associated with the first k stages of the decision process and PC^k is a partial configuration for that set of relations composed of relation-implementations associated with undominated choices. It is clear that $SSR_{MR} = VSR$.

Definition 4.3.2

Let $z = \langle PC, S, T \rangle$ and $z' = \langle PC', S', T' \rangle$ be two k -stage solutions (for SSR_k) as defined in Definition 4.3.1, such that $S \leq S'$, then z is said to dominate z'

if $T \leq T'$. If there are two or more k -stage solutions (for SSR_k) such that their 2nd and 3rd coordinates are respectively equal, then all but one are arbitrarily selected as dominated. $\square$

This definition is similar to Definition 4.2.2 of dominance between two choices for a relation. The motivation for this definition is stated in the following observation:

Observation 4.3.1

Let VPC^* be the optimal partial configuration for VSR, i.e. VPC^* satisfies (4.2.3) and (4.2.4). Let PC^* be a subset of VPC^* composed of the first k elements of VPC^* . Then for no $k = 1, 2, \dots, MR$, there exists a k -stage solution $z = \langle PC, S, T \rangle$ that dominates $z^* = \langle PC^*, S^*, T^* \rangle$ and $PC \neq PC^*$, where $S^* = CS(PC^*)$ and $T^* = CT(PC^*)$.

Proof: Suppose z dominates z^* , then by definition $CS(PC) \leq CS(PC^*)$ and $CT(PC) \leq CT(PC^*)$. Let VPC^+ be the partial configuration obtained as follows:

$$VPC^+ = (VPC^* - PC^*) \cup PC$$

With an argument similar to the one presented in Observation 4.2.1, it can be shown that VPC^+ must have a smaller time cost (or smaller storage cost in case of equal time costs) than VPC^* . This is a contradiction to the assumption that VPC^* was optimal for VSR. $\square$

This observation justifies our desire to only consider solutions that are not dominated by other solutions.

In fact, it can be easily shown that, whatever algorithm is used to derive $(k + 1)$ st stage solutions from k -stage solutions $(k = 1, 2, \dots, MR - 1)$, dominated k -stage solutions "lead to" dominated $(k + 1)$ -stage solutions.

Definition 4.3.3

If there are two k -stage solutions z and z' such that neither one dominates the other, then z' and z are said to be a pair of undominated k -stage solutions. $\square$

The property shown for undominated choices of a relation in Observation 4.2.2 applies to undominated solutions and it is stated in the following:

Observation 4.3.2

If $z = \langle PC, S, T \rangle$ and $z' = \langle PC', S', T' \rangle$ are two undominated k -stage solutions where $PC \neq PC'$, then either $S < S'$ and $T > T'$ or $S' < S$ and $T' > T$. $\square$

Definition 4.3.4

A set of k -stage solutions is said to be an undominated set of k -stage solutions if every pair of elements in the set is a pair of undominated k -stage solutions. $\square$

As in the case of a set of undominated choices for a relation, the elements of a set of k -stage undominated solutions $z^k = \{z^k \mid z^k = \langle PC^k, S^k, T^k \rangle\}$ may be assumed, with no loss of generality, to be in the increasing order of storage costs and (at the same time) in the decreasing order of time costs. In other words, if $p = |z^k|$ and $z_i = \langle PC_i^k, S_i^k, T_i^k \rangle \in z^k$, then for all $i = 1, 2, \dots, p - 1$,

$$S_i^k < S_{i+1}^k \text{ and } T_i^k > T_{i+1}^k.$$

Theorem 4.3.1

Let $Z^k = \{z_i^k \mid z_i^k = \langle PC_i^k, S_i^k, T_i^k \rangle, i = 1, 2, \dots, p\}$

be a set of undominated k -stage solutions such that

$S_i^k < S_{i+1}^k$ for $i = 1, 2, \dots, p-1$. Let

$z^k = \langle PC^k, S^k, T^k \rangle$ be another k -stage solution ($PC^k \neq PC_i^k$, $i = 1, 2, \dots, p$) such that $S^k > S_p^k$. Then z^k is pairwise undominated with z_i^k , $i = 1, 2, \dots, p$ iff z^k and z_p^k is a pair of undominated k -stage solutions. $\square$

The proof of this theorem is analogous to the proof given for a set of undominated choices in Theorem 4.2.1.

The following corollary follows from Theorem 4.3.1:

Corollary 4.3.1

Let $Z^k = \{z_i^k \mid z_i^k = \langle PC_i^k, S_i^k, T_i^k \rangle, i = 1, 2, \dots, p-1\}$

be a set of undominated k -stage solutions with the property

that $S_i^k < S_{i+1}^k$ (and, therefore, $T_i^k > T_{i+1}^k$) for all

$i = 1, 2, \dots, p-2$. Let $z_p^k = \langle PC_p^k, S_p^k, T_p^k \rangle$ be another k -stage solution such that $S_p^k \geq S_p^k - 1$. Let $\tilde{Z}^k$ be an undominated set of k -stage solutions involving the maximum

number of elements of $Z^k \cup \{z_p^k\}$, such that for all

$z \in Z^k \cup \{z_p^k\} - \tilde{Z}^k$, z is dominated by some $z' \in \tilde{Z}^k$. Then:

- i) $\tilde{Z}^k = Z^k \cup \{z_p^k\}$ iff $S_p^k > S_p^k - 1$ and $T_p^k < T_{p-1}^k$
- ii) $\tilde{Z}^k = Z^k \cup \{z_p^k\} - \{z_{p-1}^k\}$ if $S_p^k = S_p^k - 1$ and $T_p^k < T_{p-1}^k$
- iii) $\tilde{Z}^k = Z^k$ or $\tilde{Z}^k = Z^k \cup \{z_p^k\} - \{z_{p-1}^k\}$ if $S_p^k = S_p^k - 1$ and $T_p^k = T_{p-1}^k$

$$S_i^k < S_{i+1}^k \text{ and } T_i^k > T_{i+1}^k.$$

Theorem 4.3.1

Let $Z^k = \{z_i^k \mid z_i^k = \langle PC_i^k, S_i^k, T_i^k \rangle, i = 1, 2, \dots, p\}$

be a set of undominated k -stage solutions such that

$S_i^k < S_{i+1}^k$ for $i = 1, 2, \dots, p-1$. Let

$z^k = \langle PC^k, S^k, T^k \rangle$ be another k -stage solution ($PC^k \neq PC_i^k$, $i = 1, 2, \dots, p$) such that $S^k > S_p^k$. Then z^k is pairwise undominated with z_i^k , $i = 1, 2, \dots, p$ iff z^k and z_p^k is a pair of undominated k -stage solutions. $\square$

The proof of this theorem is analogous to the proof given for a set of undominated choices in Theorem 4.2.1.

The following corollary follows from Theorem 4.3.1:

Corollary 4.3.1

Let $Z^k = \{z_i^k \mid z_i^k = \langle PC_i^k, S_i^k, T_i^k \rangle, i = 1, 2, \dots, p-1\}$

be a set of undominated k -stage solutions with the property

that $S_i^k < S_{i+1}^k$ (and, therefore, $T_i^k > T_{i+1}^k$) for all

$i = 1, 2, \dots, p-2$. Let $z_p^k = \langle PC_p^k, S_p^k, T_p^k \rangle$ be another k -stage solution such that $S_p^k \geq S_p^k - 1$. Let $\tilde{Z}^k$ be an undominated set of k -stage solutions involving the maximum

number of elements of $Z^k \cup \{z_p^k\}$, such that for all

$z \in Z^k \cup \{z_p^k\} - \tilde{Z}^k$, z is dominated by some $z' \in \tilde{Z}^k$. Then:

$$\text{i) } \tilde{Z}^k = Z^k \cup \{z_p^k\} \text{ iff } S_p^k > S_p^k - 1 \text{ and } T_p^k < T_p^k - 1$$

$$\text{ii) } \tilde{Z}^k = Z^k \cup \{z_p^k\} - \{z_p^k - 1\} \text{ if } S_p^k = S_p^k - 1 \text{ and } T_p^k < T_p^k - 1$$

$$\text{iii) } \tilde{Z}^k = Z^k \text{ or } \tilde{Z}^k = Z^k \cup \{z_p^k\} - \{z_p^k - 1\} \text{ if } S_p^k = S_p^k - 1 \text{ and } T_p^k = T_p^k - 1$$

iv) $\tilde{z}^k = z^k$, otherwise. □

As discussed earlier, we are interested in generating k -stage solutions for $k = 1, 2, \dots, MR$, but retaining only the maximal set of undominated solutions at each stage. If k -stage solutions are generated in the proper order, namely in increasing order of storage costs and increasing order of time costs in case of equal storage costs, then it is clear from the above corollary that only one comparison is necessary to decide whether to retain or discard a generated solution.

Let z^k denote the set of k -stage undominated solution, $z^k = \{z_i^k \mid z_i^k = \langle PC_i^k, SS_i^k, TT_i^k \rangle, i = 1, 2, \dots, NS(k)\}$ where $NS(k) = |z^k|$, and let Q^k be the set of undominated choices for stage k (relation $RL(k)$), $Q^k = \{q_p^k \mid q_p^k = \langle MPL_p^k, S_p^k, T_p^k \rangle, p = 1, 2, \dots, NCHS(k)\}$ for $k = 1, 2, \dots, MR$. In order to derive z^k , from Q^k , $k = 1, 2, \dots, MR$, we make use of the following two Observations.

Observation 4.3.3

Let z^k and Q^k be defined as above. The set z^1 of undominated solutions to stage 1 is directly derivable from $Q^1 = \{q_p^1 \mid q_p^1 = \langle MPL_p^1, S_p^1, T_p^1 \rangle, p = 1, 2, \dots, NCHS(1)\}$ in the following way:

$$NS(1) = NCHS(1)$$

$$\text{For } i = 1, \dots, NS(1), z_i^1 = \langle PC_i^1, S_i^1, T_i^1 \rangle$$

$$\text{where } PC_i^1 = \{\langle R_{RL(1)}, MPL_i^1 \rangle\}.$$

Proof: For each $1 \leq i \leq NS(1)$, z_i^1 satisfies the requirements of Definition 4.3.1 for $k = 1$, therefore, it is a solution to stage 1. The claim that Z^1 composed of all such z^1 , is the maximal set of undominated solutions to stage 1 follows directly from the assumption that Q^1 is the maximal set of undominated choices for stage 1 and that $SS_i^1 = S_i^1$ and $TT_i^1 = T_i^1$, $i = 1, 2, \dots, NCHS(1)$. $\square$

Observation 4.3.4

Let Z^k and Q^k be defined as above for $k = 1, 2, \dots, MR$. $z = \langle PC, SS, TT \rangle$ is a $(k + 1)$ -stage solution if for some $z_i = \langle PC_i, SS_i, TT_i \rangle$ in Z^k and some $q_p = \langle MPL_p, S_p, T_p \rangle$ in Q^{k+1} the following is true:

$$PC = PC_i \cup \{ \langle R_{RL(k+1)}, MPL_p \rangle \}$$

$$SS = SS_i + S_p$$

$$TT = TT_i + T_p,$$

$$1 \leq i \leq NS(k), 1 \leq p \leq NCHS(k + 1)$$

The proof of this observation is also clear from Definition 4.3.1. $\square$

The $(k + 1)$ -stage solutions obtained in this manner are not, however, necessarily undominated. For the purpose of simplicity, let $N = NS(k)$ and $C = NCHS(k + 1)$, then there are $N \cdot C$ solutions at stage $(k + 1)$. We have already established that if these solutions are considered in the proper order, then it can be decided whether or not a solution is dominated in only one step. The task of selection of $(k + 1)$ -stage solutions in the desired order

takes $(N \cdot C(N \cdot C - 1))/2$ steps. However, since elements of z^k are already in the desired order, all $(k + 1)$ -stage solutions generated by the addition of a given undominated choice for stage $(k + 1)$ to elements of z^k will be in the desired order. Taking advantage of this property, we can reduce the number of steps to $NC(C - 1)$. This is a considerable saving in time complexity as long as $N > 2 - \frac{1}{C}$, which is almost always the case (recall that $C \geq 2$).

The process of selection of $(k + 1)$ -stage solutions in the desired order using the above method is similar to that of merging, at stage k , $NCHS(k + 1)$ runs of length $NS(k)$. Of course, instead of retaining the whole merged sequence, only those items associated with undominated solutions will be retained. In that sense, more elaborate techniques developed for "multiway merging" may be used when necessary. For example, if $NCHS(k)$ is usually larger than 8, some work can be saved by using a "selection tree" (see [Knuth 1973]).

Algorithm 4.3.1 below generates all k -stage solutions for $k = 1, 2, \dots, MR$. Following is a description of some important variables used in the algorithm :

MNIT	is the maximum of $NCHS(k)$ over $k = 1, 2, \dots, MR$.
ITP(i)	$i = 1, 2, \dots, MNIT$ called the Input Tape Pointer, is an array of indices where for $1 \leq CH \leq NCHS(k + 1)$,

ITP(CH) contains, at stage k , the k -stage solution number to be considered with choice number CH of stage $k + 1$.

CH is the choice number, index into $FMP(CH, k + 1)$, $1 \leq CH \leq NCHS(k + 1)$.

NSC, NTC are the storage and time costs, respectively, of the $(k + 1)$ -stage solution composed of k -stage solution number ITP(CH) with the $(k + 1)$ -stage choice CH.

WINCH, WINSC, WINTC are choice number, storage and time cost, respectively, of the "winner" $(k + 1)$ -stage solution.

NS(k) $k = 1, 2, \dots, MR$, is the number of undominated solutions to stage k .

OPL(i, k) $k = 1, 2, \dots, MR$,
 $i = 1, 2, \dots, NS(k)$ is the implementation number (1st coordinate) of the optimal choice for stage k (optimal policy) associated with i -th undominated solution of stage k .

POP(i, k) $k = 1, 2, \dots, MR$,
 $i = 1, 2, \dots, NS(k)$ is called the Previous Optimal Policy and contains the index i' of the $(k - 1)$ -stage solution that when combined with

implementation $OPL(i, k)$ for $R_{RL}(k)$ produces the i -th solution to stage k .

$NSS(k')$ $k' = 1, 2$ number of undominated k and $(k + 1)$ -stage solutions, respectively.

$OVF(i, k')$ $k' = 1, 2, i = 1, 2, \dots, NSS(k')$ called the optimal (objective) value function; contains the time cost of the i -th undominated solution to stage k .

$OVS(i, k')$ defined similar to $OVF(i, k')$ for storage cost of undominated k -stage solutions.

Note that the two matrices OVF and OVS have only 2 columns ($k' = 1, 2$), rather than MR columns. This is because the storage and time costs of undominated solutions of stage k are only needed to produce those of stage $k + 1$ and, therefore, only two sets of pairs of storage and time costs should be maintained at each stage. Two variables INV and OUV contain column indices into the matrix for k - and $(k + 1)$ -stage solutions, respectively, where $INV, OUV \in \{1, 2\}$. The optimal policy at each stage (OPL matrix) should, however, be maintained for all MR stages. The POP matrix, while not absolutely necessary to maintain, facilitates to a great extent the process of backtracking overall optimal policies (Algorithm 4.3.2)

Algorithm 4.3.1

/* Generate one-stage solutions, using Observation

4.3.3 */

NSS(1) = NCHS(1);

INV = 1; OUTV = 2;

For CH = 1 until NSS(1) do;

OVS(CH, INV) = TC(FMP(CH, 1), RL(1));

OVS(CH, INV) = SC(FMP(CH, 1), RL(1));

OPL(CH, INV) = FMP(CH, 1);

end;

/* Now, at stage k, generate solutions for stage */

/* k + 1, using Observation 4.3.4 */

For k = 1 until MR - 1 do;

OVS(NSS(INV) + 1, INV) = INFINITY;

/* End Marker = Infinity */

NSS(OUTV) = 0; /* No (k + 1)-stage solution yet */

ITP = 1; /* Set all Input Tape Pointers to
1 */

WINNER = 1; /* WINNER = 1 if there exists a
winner choice, otherwise

WINNER = 0 */

While WINNER = 1 do;

WINNER = 0; /* No Winner yet */

WINTC, WINSC = INFINITY; /* Initialize winner's
storage and time
costs */

```

For CH = 1 until NCHS(k + 1) do;
    NSC = OVS(ITP(CH), INV) + SC(FMP(CH, k + 1),
                                RL(k + 1));

    /* This is the Next Storage Cost to
       be compared */
    IF NSC ≤ MSB then do;
        WINNER = 1; /* Indicate existence of a
                       winner */
        NTC = OVF(ITP(CH), INV) + TC(FMP(CH, k + 1),
                                      RL(k + 1));

        If NSC = WINSC then do;
            IF NTC < WINTC then Interchange;
            IF NTC < WINTC then Interchange;
        end;
        else If NSC < WINSC then Interchange;
        end;
    end;

    If WINNER = 1 then do;
        ITP(WINCH) = ITP(WINCH) + 1; /* Advance The
                                       Proper Tape */
        If NSS(OUV) = 0 then Add; /* Add 1st one */
        else If OVF(NSS(OUV), OUV) > WINTC then Add;
            /* Add others if undominated */
        end;
    end;

    TEMP = INV; INV = OUV; OUV = TEMP;

end;

```

□

In this algorithm Add and Interchange are as follows:

```
Interchange:  do;  WINCH = CH;
               WINTC = NTC;
               WINSK = NSC;
               end;

Add:          do;  NSS(OUV) = NSS(OUV) + 1;
               OVF(NSS(OUV), OUV) = WINTC;
               OVS(NSS(OUV), OUV) = WINSK;
               OPL(NSS(OUV), k + 1)
               = FMP(WINCH, k + 1);
               POP(NSS(OUV), k + 1)
               = ITP(WINCH) - 1;
               end;
```

The overall optimal policies can now be traced back using a simple algorithm such as Algorithm 4.3.2. It is clear that there are $NSS(INV) = NS(MR)$ optimal partial configuration $VPC^*(i)$, $i = 1, 2, \dots, NSS(INV)$ for VSR. The i -th such configuration is optimal for the following range of values for L :

$C_i^* = FPC \cup VPC^*(i)$ is the optimal configuration for all values of L such that:

$$OVS(i, INV) \leq L - MMSB - BASC < OVS(i + 1, INV).$$

Algorithm 4.3.2

```
For i = NSS(INV) until 1 step - 1 do;
    VPC*(i) =  $\phi$ ; /* Initialize to empty set */
    Index = i;
```

```

For k = MR until 1 step - 1 do;
    VPC*(i) = VPC*(i) U {<RRL(k), OPL(Index, k)>};
    Index = POP(Index, k);
end;
end;

```

□

The complexity of Algorithm 4.3.1 at stage k is proportional to $W(k) = NS(k) * [NCHS(k + 1)]^2$, $k = 1, 2, \dots, MR - 1$. This does not include the initial step of generating one-stage solutions which has a trivial time complexity proportional to $NCHS(1)$. The total complexity is then proportional to:

$$W = \sum_{k=1}^{MR-1} NS(k) * [NCHS(k + 1)]^2$$

The cardinality of the set of undominated solutions for stage k , $NS(k)$ ranges from $NS(k - 1) * NCHS(k)$ in the worst case to $\text{Max}(NS(k - 1), NCHS(k))$ in the best case. Usually, however, $NS(k)$ is on the order of $(NS(k - 1) + NCHS(k))$.

Note that even in the worst case where W is proportional to $\prod_{k=1}^{MR} NCHS(k)$, this number cannot be greater than $OMSB$ and in reality it is much smaller.

If we assume $NS(k)$ to be on the order of $NS(k - 1) + NCHS(k)$, then the following observation shows that W is minimized when stages are considered in the non-increasing order of $NCHS(k)$, i.e. if $NCHS(k) \geq NCHS(k + 1)$, $k = 1, 2, \dots, MR - 1$.

Observation 4.3.5

Let SP be a set of permutations:

$SP = \{P \mid P = \langle p_1, p_2, \dots, p_N \rangle\}$ is a permutation of $P^* = \langle 1, 2, \dots, N \rangle$. Let $a_1, a_2, \dots, a_N$ be real numbers such that for $1 \leq i \leq N$, $a_i \geq a_{i+1}$. Let $\tilde{P}$ be a sequence of numbers obtained from $P \in SP$ by replacing its i -th coordinate p_i with a_{p_i} : $\tilde{P} = \langle a_{p_1}, a_{p_2}, \dots, a_{p_N} \rangle$. If $W(\tilde{P}) \triangleq \sum_{i=1}^{N-1} (a_{p_1} + a_{p_2} + \dots + a_{p_i}) \cdot a_{p_{i+1}}^2$, then $W(\tilde{P}^*) \leq W(\tilde{P})$ for all $P \in SP$.

Proof: We first show that for any permutation P if two adjacent elements are interchanged in the wrong (correct) direction, then $W(\tilde{P})$ increases (decreases). More specifically we show that if

$P = \langle p_1, \dots, p_{i-1}, p_i, p_{i+1}, p_{i+2}, \dots, p_N \rangle$ and $P' = \langle p_1, \dots, p_{i-1}, p_{i+1}, p_i, p_{i+2}, \dots, p_N \rangle$ and $a_{p_i} \geq a_{p_{i+1}}$, then $W(\tilde{P}') \geq W(\tilde{P})$. In order to show this, we expand $W(\tilde{P})$ and $W(\tilde{P}')$ and then form the difference

$W(\tilde{P}') - W(\tilde{P})$ as follows:

$$\begin{aligned} W(\tilde{P}) &= a_{p_1} a_{p_2}^2 + (a_{p_1} + a_{p_2}) a_{p_3}^2 + \dots + (a_{p_1} + \dots + a_{p_{i-1}}) a_{p_i}^2 + \\ &\quad + (a_{p_1} + \dots + a_{p_{i-1}} + a_{p_i}) a_{p_{i+1}}^2 + \dots + (a_{p_1} + \dots + a_{p_{N-1}}) a_{p_N}^2 \\ W(\tilde{P}') &= a_{p_1} a_{p_2}^2 + (a_{p_1} + a_{p_2}) a_{p_3}^2 + \dots + (a_{p_1} + \dots + a_{p_{i-1}}) a_{p_{i+1}}^2 + \\ &\quad + (a_{p_1} + \dots + a_{p_{i-1}} + a_{p_{i+1}}) a_{p_i}^2 + \dots + (a_{p_1} + \dots + a_{p_{N-1}}) a_{p_N}^2 \\ W(\tilde{P}') - W(\tilde{P}) &= a_{p_{i+1}} a_{p_i}^2 - a_{p_i} a_{p_{i+1}}^2 = a_{p_i} a_{p_{i+1}} (a_{p_i} - a_{p_{i+1}}) \end{aligned}$$

This establishes the first part of the proof in both direc-

tions, namely that i) if $a_{p_i} \geq a_{p_{i+1}}$ (wrong direction), then $W(\tilde{P}') \geq W(\tilde{P})$; ii) if $a_{p_i} \leq a_{p_{i+1}}$ (correct direction), then $W(\tilde{P}') \leq W(\tilde{P})$; and iii) if $a_{p_i} = a_{p_{i+1}}$, then $W(\tilde{P}') = W(\tilde{P})$.

We now have to show that any permutation $P \in SP$ is reachable from P^* by a series of interchanges of adjacent elements in the wrong direction. Or equivalently from any permutation P in SP , we can arrive at P^* by a series of interchanges of adjacent elements in the correct direction. The latter phrasing of this claim is constructively proved by the application of the "Bubble Sort" algorithm to sort elements of P in the nonincreasing order of a_{p_i} . It is clear that i) all interchanges are between two adjacent elements and in the correct direction, and ii) that the final sorted sequence is $P^* = \langle 1, 2, \dots, N \rangle$, or reachable from it with a series of interchanges that do not change $W(\tilde{P})$. Note that $W(\tilde{P}') = W(\tilde{P})$ if $a_{p_{i+1}} = a_{p_i}$. $\square$

The amount of work saved by considering stages in the nonincreasing order of $NCHS(k)$ is nominal if $NCHS(k)$, $k = 1, \dots, MR$ are all on the same order of magnitude. It can, however, save a reasonable amount of work in cases such as the one illustrated by the following example:

$$W(\langle 20, 5, 2 \rangle) = 600 \text{ (min)}$$

$$W(\langle 20, 2, 5 \rangle) = 630$$

$$W(\langle 5, 20, 2 \rangle) = 2100$$

$$W(\langle 2, 5, 20 \rangle) = 2850 \text{ (max)}$$

We emphasize that the above property regarding the ordering of stages was shown for the case where

$NS(k + 1) = NS(k) + \text{CONSTANT} * NCHS(k + 1)$ which is an average type of behavior expected for these kinds of problems. It is possible to prove this property for more complex types of relationships between $NS(k)$ and $NCHS(k)$, $k = 1, 2, \dots, MR$, including the worst case situation where $NS(k + 1) = NS(k) * NCHS(k + 1)$. The difference $W(P') - W(P)$ in this case is

$C_1 C_2 \dots C_{i-1} [C_i C_{i+1} - (C_i + C_{i+1})] (C_i - C_{i+1})$. Note that the sign of $(W(P') - W(P))$ is the same as $(C_i - C_{i+1})$ because $C_k \geq 2$ for all $k = 1, 2, \dots, MR$.

The optimization program produces the solution to the optimization problem stated in Section 4.1, in two phases. Phase 1 uses Algorithm 4.2.1 of Section 4.2 to generate undominated choices for each relation in the set of database relations. The bulk of savings in processing time is achieved by phase 1. In the case of our application example, after deleting the dominated choices for each relation and discarding those relations for which one choice dominates all others, we are left with only 6 relations (and hence 6 stages in the second phase) with the following number of choices $NCHS(k)$ at each stage $k = 1, 2, \dots, 6$.

(Note: $MR = 6$)

k	NCHS(k)	k	NCHS(k)
1	3	4	3
2	2	5	2
3	4	6	3

Table 4.3.1 Maximal number of undominated choices for stage k

Note that for the above ordering of stages:

$W(<3, 2, 4, 3, 2, 3>) = 347$ which is about 15% greater

than $W(\tilde{P}^*) = 301$ for the best ordering of stages

$\tilde{P}^* = <4, 3, 3, 3, 2, 2>$. For the worst ordering of stages

$\tilde{P}' = <2, 2, 3, 3, 3, 4>$ the value of $W(\tilde{P}')$ is equal to

405 which is in turn about 34.5% greater than $W(\tilde{P}^*) = 301$.

In phase 2, the optimization program uses Algorithm 4.3.1 to find the maximal set of undominated solutions plus necessary information to derive the optimal partial configurations $VPC^*(i)$, $i = 1, 2, \dots, NS(MR)$, using Algorithm 4.3.2. At the end of this phase, the following information is available:

i) $NS(MR) = NSS(INV)$

ii) C_i^* $i = 1, 2, \dots, NS(MR)$, i -th optimal configuration

iii) $CS(C_i^*) = BASC + OVS(i, INV)$ storage cost of C_i^*

iv) $CT(C_i^*) = BATC + OVF(i, INV)$ time cost of C_i^*

where C_i^* is the optimal configuration for all storage limits L such that $CS(C_i^*) \leq L - MMSB < CS(C_{i+1}^*)$. The total storage requirement of C_i^* is $MMSB + CS(C_i^*)$, $i = 1, 2, \dots, NS(MR)$. C_i^* and $C_{NS(MR)}^*$, denoted C_m and C_M earlier in this section are the optimal configurations with the smallest and largest storage requirements, respectively. For $L \geq MMSB + CS(C_M)$, the optimal configuration is C_M and for $L < L_O = MMSB + CS(C_m)$, no optimal configuration exists.

4.4 Data Base Design

In this section we discuss the method of data base design given a configuration C for the set of data base relations SR . More specifically, we want to derive the data base design in terms of record and set declarations. We recall that a configuration C for SR is a set of relation implementation pairs, $\langle R_j, MPL_j \rangle$, $j = 1, 2, \dots, NR$, consisting of exactly one element for each relation in SR . Each MPL_j , $j = 1, 2, \dots, NR$ is a number from 1 to 24 designating an implementation number. As defined in Chapter 3, implementations 1, 2, 3, and 4 are duplications and implementations 5, 6, 7, and 8 are aggregations. Implementations 9 to 20 are associations. When used to implement a relation such as $R(A, B)$ where $A = ORG(R)$ and $B = DST(R)$, odd (even) numbered implementations duplicate, aggregate or associate (whichever is appropriate), $B(A)$ under $A(B)$. Implementation 21 for $R(A, B)$ is the dummy record association of A and B and the last three implementations (22, 23 and 24) for $R(A, B)$ are called single linkage of B under A , single linkage of A under B and double linkage of A and B , respectively.

The design of the data base takes place in three steps. In the first two steps intra-record relationships (duplications and aggregations) are considered and thereby all record types and their components are determined. In the third step, inter-record relationships (associations and

linkages) are considered to determine data base set declarations.

In step 1, we start by identifying all data items A in the set of data items SI that are not aggregated under some other data item type $B \in SI$. In other words, given a configuration C for SR such as

$C = \{ \langle R_1, MPL_1 \rangle, \dots, \langle R_{NR}, MPL_{NR} \rangle \}$ we are looking for all data items $I \in SI$ such that for each $j = 1, 2, \dots, NR$:

1) if $DST(R_j) = I$, then $\langle R_j, 5 \rangle \notin C$ and $\langle R_j, 7 \rangle \notin C$

2) if $DRG(R_j) = I$, then $\langle R_j, 6 \rangle \notin C$ and $\langle R_j, 8 \rangle \notin C$

One record type will be defined in the data base definition for each such data item type and the data item type will be called the "principal item" of the record type. Number of occurrences of the record type is equal to the cardinality of that data item type. The name of the record type will be formed by concatenating the string "REC" to the right of the first 3 characters of the name of the data item. For example consider configuration C', given below, whose data base design is given in Figure 5.8 in Chapter 5.

$C' = \{ \langle R_1, 5 \rangle, \langle R_2, 5 \rangle, \langle R_3, 9 \rangle, \langle R_4, 5 \rangle, \langle R_5, 5 \rangle, \langle R_6, 7 \rangle, \langle R_7, 5 \rangle, \langle R_8, 5 \rangle, \langle R_9, 5 \rangle, \langle R_{10}, 9 \rangle, \langle R_{11}, 3 \rangle, \langle R_{12}, 12 \rangle, \langle R_{13}, 5 \rangle, \langle R_{14}, 5 \rangle, \langle R_{15}, 5 \rangle, \langle R_{16}, 5 \rangle, \langle R_{17}, 5 \rangle, \langle R_{18}, 5 \rangle, \langle R_{19}, 5 \rangle, \langle R_{20}, 5 \rangle, \langle R_{21}, 5 \rangle, \langle R_{22}, 9 \rangle, \langle R_{23}, 21 \rangle, \langle R_{24}, 5 \rangle, \langle R_{25}, 5 \rangle, \langle R_{26}, 13 \rangle \}$

Name, origin data item type and destination data item type of each relation $R_1, \dots, R_{26}$ are given in Chapter 2. We

note that data item I_{11} called EMPNUM participates in relation R_{11} to R_{18} as origin and in 10, 22 and 23 as destination data item type. Given the set of relation implementations in C' , we observe that EMPNUM is not aggregated under any other data item type and, therefore, a record type with unique EMPNUM values in its occurrences will be defined and will be called EMPREC (see Figure 5.8). Similarly, data item types called ORGCOD, JOBCOD, OFRCOD and COUNUM are not aggregated under any other data item type, therefore, four more record types will be defined and called ORGREC, JOBREC, OFRREC and COUREC, respectively.

In the second step of data base definition, we define one component vector or repeating group, at level one of record definition, for each aggregation or duplication of some other data item type with the principal item of the record. A component vector will be defined for each duplication of some other data item under the principal item and each aggregation of some other data item if no other data item is aggregated or duplicated under the latter. In the case where another data item is aggregated or duplicated under the second data item (the one aggregated under the principal item), then a repeating group is defined and the second data item type becomes its principal item. Similarly, for any data item type duplicated or aggregated under the principal item of a repeating group, a vector or a repeating group at one level lower (higher

level number) than the original repeating group will be defined. This allows arbitrarily deep levels of nested repeating group definitions. In the case of configuration C' for SR, the vector called JOBCOD in EMPREC represents a duplication of data item JOBCOD under EMPNUM to implement relation R_{11} called JHSOFEMP (job history of employee) using implementation 3. The counter data item JHSCNT contains the length of the vector. As an example of a repeating group in the same configuration, we observe that ATHSAL is (variably) aggregated under JOBCOD to implement SALOFJOB; see pair $\langle R_6, 7 \rangle \in C'$. At the same time ATHMAX, ATHMIN and DDUCTN are aggregated under ATHSAL to implement MAXOFSAL, MINOFSAL and DDCOFSAL, respectively; see $\langle R_7, 5 \rangle$, $\langle R_8, 5 \rangle$ and $\langle R_9, 5 \rangle$ in C'. Therefore, a variably dimensioned repeating group called SALRPG is defined in JOBREC whose dimension is given by the counter data item SALRGC. Data item type ATHSAL is the principal item of the repeating group and ATHMAX, ATHMIN and DDUCTN are its components and DDUCTN is a vector of a fixed length. (see Figure 5.8). This completes the formation of record types.

In step three of data base definition, we consider all association and linkage implementation to derive all inter-record relationships in the data base. First for each $\langle R_j, MPL_j \rangle \in C$ such that MPL_j is an odd (even) number from 9 to 20, we define a data base set type. The name of the

data base set type is the same as the name of R_j . Its owner record type is the record type for which $ORG(R_j)$ ($DST(R_j)$) is the principal item. The member record type of the set type is the record type for which $DST(R_j)$ ($ORG(R_j)$) is the principal item. The following should be specified as OPTIONS in the definition of the set type:

CHAIN if $MPL_j = 9$ or 10

CHAIN WITH OWNER POINTERS if $MPL_j = 11$ or 12

CHAIN WITH PRIOR POINTERS if $MPL_j = 13$ or 14

CHAIN WITH OWNER AND PRIOR POINTERS if $MPL_j = 15$ or 16

POINTER ARRAY if $MPL_j = 17, 18$

POINTER ARRAY WITH OWNER POINTERS if $MPL_j = 19$ or 20 .

For example in the case of configuration C' given above since $\langle R_{12}, 12 \rangle \in C'$, a data base set type called JOBOFEMP is defined. The owner and member record types of the set type are JOBREC and EMPREC, respectively, and CHAIN WITH OWNER POINTERS has been specified. Recall that R_{12} , $ORG(R_{12})$ and $DST(R_{12})$ are called JOBOFEMP, ORGCOD and EMPNUM, respectively. Similarly, there are four other data base set types to be defined for R_3 , R_{10} , R_{22} and R_{26} (see Figure 5.8).

Next we define one (dummy) record type and two data base set types for each $\langle R_j, 21 \rangle \in C$. The dummy record type is the common member record type of both set types and $ORG(R_j)$ is the owner record type of one set type while $DST(R_j)$ is the owner of the other set type. The OPTION

specified for both set types is always CHAIN WITH OWNER POINTERS. Example of this is $\langle R_{23}, 21 \rangle \in C'$ where the dummy record type STULNK (student link) is defined (Figure 5.8) as the 6-th record type and two data base set types STUDNTOF and STUDENTS are defined as the 6-th and 7-th data base set types in the data base definition.

Finally for each $\langle R_j, MPL_j \rangle \in C$ such that $MPL_j = 22, 23$ or 24 , a fixed length vector of data-base keys (pointers) is defined in the record type for which $ORG(R_j)$ is the principal item if $MPL_j = 22$, in the record type for which $DST(R_j)$ is the principal item if $MPL_j = 23$, and in both if $MPL_j = 24$.

This completes the three steps of data base (definition) design from a given configuration. Several data base designs are given in Chapter 5 where each one is optimal, under certain conditions, for the application example of Chapter 2. In the following, we give the configurations corresponding to data base designs of Figures 5.1 to 5.6 of Chapter 5. In order to simplify the presentation, we will first give the common subset of all the six configurations as FPC. Then $C'_1, \dots, C'_6$ the configurations corresponding to Figures 5.1 to 5.6, respectively, are given by $FPC \cup VPC_1, \dots, FPC \cup VPC_6$, where $VPC_1, \dots, VPC_6$ and FPC are as follows:

$$FPC = \{ \langle R_1, 5 \rangle, \langle R_2, 5 \rangle, \langle R_4, 5 \rangle, \langle R_5, 5 \rangle, \langle R_6, 7 \rangle, \langle R_7, 5 \rangle, \\ \langle R_8, 5 \rangle, \langle R_9, 5 \rangle, \langle R_{11}, 3 \rangle, \langle R_{13}, 5 \rangle, \langle R_{14}, 5 \rangle,$$

$$\begin{aligned}
& \langle R_{15}, 5 \rangle, \langle R_{16}, 5 \rangle, \langle R_{17}, 5 \rangle, \langle R_{18}, 5 \rangle, \langle R_{19}, 5 \rangle, \\
& \langle R_{20}, 5 \rangle, \langle R_{21}, 5 \rangle, \langle R_{24}, 5 \rangle, \langle R_{25}, 5 \rangle \}, \\
VPC_1 = & \{ \langle R_3, 9 \rangle, \langle R_{10}, 9 \rangle, \langle R_{12}, 4 \rangle, \langle R_{22}, 3 \rangle, \langle R_{23}, 3 \rangle, \\
& \langle R_{26}, 2 \rangle \}, \\
VPC_2 = & \{ \langle R_3, 9 \rangle, \langle R_{10}, 9 \rangle, \langle R_{12}, 1 \rangle, \langle R_{22}, 3 \rangle, \langle R_{23}, 4 \rangle, \\
& \langle R_{26}, 2 \rangle \}, \\
VPC_3 = & \{ \langle R_3, 9 \rangle, \langle R_{10}, 9 \rangle, \langle R_{12}, 10 \rangle, \langle R_{22}, 9 \rangle, \langle R_{23}, 3 \rangle, \\
& \langle R_{26}, 9 \rangle \}, \\
VPC_4 = & \{ \langle R_3, 9 \rangle, \langle R_{10}, 9 \rangle, \langle R_{12}, 12 \rangle, \langle R_{22}, 9 \rangle, \langle R_{23}, 3 \rangle, \\
& \langle R_{26}, 13 \rangle \}, \\
VPC_5 = & \{ \langle R_3, 9 \rangle, \langle R_{10}, 9 \rangle, \langle R_{12}, 4 \rangle, \langle R_{22}, 3 \rangle, \langle R_{23}, 3 \rangle, \\
& \langle R_{26}, 13 \rangle \}, \\
VPC_6 = & \{ \langle R_3, 3 \rangle, \langle R_{10}, 3 \rangle, \langle R_{12}, 4 \rangle, \langle R_{22}, 3 \rangle, \langle R_{23}, 4 \rangle, \\
& \langle R_{26}, 2 \rangle \}.
\end{aligned}$$

CHAPTER V

RESULTS

5.1 Sensitivity of Time Cost Functions

We developed the time cost functions for relation implementations in Chapter 3. Time costs of relation implementations are evaluated, using these functions, and the resulting values are used as the respective second cost components of the relation implementations. The first cost component associated with implementing a relation by a specific implementation is the storage cost of the relation implementation. Formulas for computing storage costs were also discussed in Chapter 3. In Chapter 4 we used this two-component cost measure to design a data base for a given application.

Time costs, measured in the expected number of page faults, were expressed as functions of four page fault probabilities p_1 , p_2 , p_3 and p_4 , defined in Section 3.4. We repeat the expressions for these probabilities, for convenience, regarding relation $R_j(A,B) \in SR$, $j = 1, \dots, NR$.

$$\begin{aligned}
 p_1 &= 1 - \frac{PBSZ}{TMSB} \\
 p_2(R_j) &= 1 - \frac{PBSZ}{\beta'_j * ARL} && \text{if } PBSZ < \beta'_j * ARL \\
 &= 0 && \text{otherwise} \\
 p'_2(R_j) &= 1 - \frac{PBSZ}{\alpha'_j * ARL} && \text{if } PBSZ < \alpha'_j * ARL \\
 &= 0 && \text{otherwise} \\
 p_3(R_j) &= 1 - \frac{PBSZ}{(\alpha'_j + \beta'_j) * ARL} && \text{if } PBSZ < (\alpha'_j + \beta'_j) * ARL \\
 &= 0 && \text{otherwise}
 \end{aligned}$$

Where in the above, PBSZ is called the page buffer size given by:

$$\text{PBSZ} = \text{MSPG} * \text{PGSZ},$$

ARL is the Average Record Length, TMSB is the Maximum Storage Bound, and

$$\alpha'_j = |\text{ORG}(R_j)|, \quad \beta'_j = |\text{DST}(R_j)|.$$

The above expressions for p_2 , p'_2 and p_3 hold for all implementations except for implementation 21 for which the exact expressions were given in Section 3.4. For example in the expressions for $p_2(R_j)$, in the case of implementation 21, β'_j should be replaced by r_j which is the cardinality of R_j and ARL should be replaced by DRSZ which is the size of a dummy record.

As can be seen, in most cases p_2 and p_3 depend on the average record length ARL, a quantity which is not exactly known at the beginning of the optimization process. Time costs of relation implementations, defined in terms of these probabilities, are computed using an estimated ARL to derive the "optimal" configuration. We recall from Chapters 3 and 4 that a configuration C for a set of data base relations $SR = \{R_1, \dots, R_j, \dots, R_{NR}\}$ is a set of ordered pairs of relation implementations:

$$C = \{ \langle R_1, \text{MPL}_1 \rangle, \dots, \langle R_j, \text{MPL}_j \rangle, \dots, \langle R_{NR}, \text{MPL}_{NR} \rangle \}$$

where MPL_j , $j = 1, \dots, NR$ is an implementation number (an integer from 1 to 24) designating one of the 24 implementation alternatives defined in Chapter 3. An acceptable configuration C for SR is defined as follows:

$$C = \{ \langle R_j, MPL_j \rangle \mid j = 1, \dots, NR \text{ and } AC(MPL_j, j) = 1 \\ \text{for all } j \}.$$

The (cumulative) storage cost of an acceptable configuration C is the sum of storage costs of individual relation implementations in the configuration and is given by,

$$CS(C) = \sum_{j=1}^{NR} SC(MPL_j, j) .$$

The (cumulative) time cost of an acceptable configuration C is similarly defined as follows,

$$CT(C) = \sum_{j=1}^{NR} TC(MPL_j, j)$$

The "optimal" configuration C^* for a set SR of data base relations is an acceptable configuration such that

$$CT(C^*) \leq CT(C) \text{ and}$$

$$CS(C^*) \leq TMSB$$

for all acceptable configurations C for which $CS(C) \leq TMSB$.

In Chapter 4 we developed an efficient algorithm that finds C^* . The fact that an estimated ARL value was used to evaluate time cost functions has the following two implications.

- (1) Time cost values are approximate values because the ARL of the resulting configuration was not used to derive its own time cost.
- (2) Let $CT(C_i, \ell(C_j))$ denote the time cost of configuration C_i evaluated using the average record length of configuration C_j . When a configuration C_1 is selected, as optimal, by the optimization

algorithm of Chapter 4, using $ARL = \ell_0$,
 another configuration C_2 may exist for which
 $CT(C_2, \ell(C_2)) < CT(C_1, \ell(C_1))$ and
 $CT(C_2, \ell_0) > CT(C_1, \ell_0)$.

If case (2) occurs, configuration C_2 will be missed (will not be selected) by the optimization process because it looked inferior to C_1 when time costs were evaluated using $ARL = \ell_0$. In the following we show that the percentage variation of time costs, when ARL is varies, is small enough so that the chosen configuration (C_1) is within reasonable proximity of the optimal one (C_2) if the latter was missed.

The time cost of an implementation MPL for a relation R in a typical operation 0 is of the following form.

$$TC(MPL, R, 0) = f(\mu_1 p_1 + \mu_2 p_2 + \mu_3 p_3 + \mu_4 p_4)$$

In this expression p_2 and p_3 are dependent on the average record length ARL. We define the percentage variation of time costs, TC, for a variation $\Delta \ell$ in average record length ARL as follows.

$$\delta_\ell = \frac{\Delta TC}{TC}$$

Using the above expression for TC we get:

$$\delta_\ell = \frac{\Delta TC}{TC} = \frac{\sum_{i=1}^4 \mu_i \Delta p_i}{\sum_{i=1}^4 \mu_i p_i} \quad \begin{array}{l} \Delta p_i = \text{variation of } p_i \text{ for a } \Delta \ell \\ \text{variation in } \ell, i=1, \dots, 4 \end{array}$$

And since p_1 and p_4 are independent of ARL; $\Delta p_1 = \Delta p_4 = 0$
 and we have the following,

$$\delta_\ell < \frac{\mu_2 \Delta p_2 + \mu_3 \Delta p_3}{\mu_2 p_2 + \mu_3 p_3} \quad (5.1.1)$$

In the above expression $1 - \frac{k}{\gamma_i \ell}$ should be substituted for p_i , $i = 2, 3$, where k is the page buffer size PBSZ given by: $k = \text{PGSZ} = \text{MSPG} * \text{PGSZ}$, and ℓ is the average record length. For a specific relation $R_j(A, B)$, γ_2 and γ_3 are given by the following (see Chapter 3, Section 4).

$$\begin{aligned} \gamma_2 &= \beta'_j && \text{for } p_2(R_j) \\ \gamma_2 &= \alpha'_j && \text{for } p'_2(R_j), \text{ and} \\ \gamma_3 &= \alpha'_j + \beta'_j && \text{for } p_3(R_j). \end{aligned} \quad (5.1.2)$$

The right hand side of (5.1.1) is less than or equal to $\Delta p_2/p_2$ if $\Delta p_3/p_3 \leq \Delta p_2/p_2$. It is clear from (5.1.2) that $\Delta p_3/p_3$ is less than $\Delta p_2/p_2$ and hence the following is true:

$$\delta_\ell = \frac{\Delta \text{TC}}{\text{TC}} < \frac{\mu_2 \Delta p_2 + \mu_3 \Delta p_3}{\mu_2 p_2 + \mu_3 p_3} < \frac{\Delta p_2}{p_2}$$

Substituting $1 - \frac{k}{\gamma_2 \ell}$ for p_2 and $(- \frac{k}{\gamma_2} \frac{\Delta \ell}{\ell^2})$ for Δp_2 we conclude that $\delta_\ell < ((k/\gamma_2)/(\ell - k/\gamma_2)) \cdot \frac{\Delta \ell}{\ell}$. Using the average cardinality γ of all data items in SI for γ_2 we arrive at the following definition of the error estimate δ_ℓ .

$$\delta_\ell \triangleq \frac{k/\gamma}{\ell - k/\gamma} \cdot \frac{\Delta \ell}{\ell}$$

In a similar manner we estimate the error introduced in time cost evaluations by using TMSB in the computation of p_1 instead of the true storage requirement, S , of the optimal data base. The formula for p_1 , discussed in Section 3.4, is given below.

$$p_1 = 1 - \frac{\text{PBSZ}}{\text{TMSB}}$$

The percentage variation of time cost for a variation ΔS in storage requirement S is defined as follows.

$$\delta_s = \frac{\Delta TC}{TC} = \frac{\Delta p_1}{p_1}$$

Substituting $1 - \frac{k}{S}$ for p_1 where $k = \text{PBSZ} = \text{MSPG} * \text{PGSZ}$ and S is the storage requirement for the data base, we get the following.

$$\delta_s = \frac{k/S}{1-k/S} \cdot \frac{\Delta S}{S}$$

In order to estimate errors in the worst case we use the highest/lowest possible value for $\Delta \ell / \ell$ in the computation of δ_ℓ and, similarly, the highest/lowest possible value of $\Delta S / S$ in the computation of δ_s .

The optimization program of Chapter 4 was run for the example application introduced in Chapter 2 for various $\langle \text{MSPG}, \text{PGSZ} \rangle$ pairs. For each such pair the value of ARL was varied over a range of 50 to 600 characters. Values of δ_ℓ and δ_s were computed for each run and they are tabulated in Tables 5.1 to 5.8.

For the above runs the values of other parameters of the model were set as follows; size of an address occurrence (pointer size) PTRS = 4 characters, size of a counter occurrence CNTRS = 6 characters, size of a dummy record DRSZ = 4 * PTRS = 16 characters and TMSB = 1,600,000 characters.

As can be seen in the tables, eight different pairs $\langle \text{MSPG}, \text{PGSZ} \rangle$, of Main Storage Pages/Page Sizes were considered where MSPG = 1, 2, 4 and 5 pages and PGSZ = 2000

ARL	$\ell_{\max}$	$\ell_{\min}$	$\Delta\ell$	$\delta_{\ell} \%$	$\delta_s \%$
50	260.4	211.5	48.9	2.12	.55
100	260.6	211.5	48.9	2.12	.55
200	260.6	211.5	49.1	2.12	.55
400	260.6	211.5	49.1	2.12	.55
600	260.6	211.5	49.1	2.12	.55

Table 5.1 Time cost error estimates for
MSPG = 5, PGSZ = 4000

ARL	$\ell_{\max}$	$\ell_{\min}$	$\Delta\ell$	$\delta_{\ell} \%$	$\delta_s \%$
50	260.4	211.5	48.9	1.66	.44
100	260.4	211.5	48.9	1.66	.44
220	260.6	211.5	49.1	1.67	.44
600	260.6	211.5	49.1	1.67	.44

Table 5.2 Time cost error estimates for
MSPG = 4, PGSZ = 4000

ARL	$\ell_{\max}$	$\ell_{\min}$	$\Delta\ell$	$\delta_{\ell} \%$	$\delta_s \%$
50	260.4	211.5	48.9	.8	.33
100	260.6	211.5	49.1	.8	.33
220	260.6	211.5	49.1	.8	.33
600	260.6	211.5	49.1	.8	.33

Table 5.3 Time cost error estimates for
MSPG = 2, PGSZ = 4000

ARL	$\ell_{\max}$	$\ell_{\min}$	$\Delta\ell$	$\delta_{\ell} \%$	$\delta_s \%$
50	260.6	211.5	49.1	.4	.11
220	260.6	211.5	49.1	.4	.11
600	260.6	211.5	49.1	.4	.11

Table 5.4 Time cost error estimates for
MSPG = 1, PGSZ = 4000

ARL	$\ell_{\max}$	$\ell_{\min}$	$\Delta\ell$	$\delta_{\ell} \%$	$\delta_s \%$
50	260.5	211.5	49.0	1.02	.27
100	260.6	211.5	49.1	1.02	.27
220	260.6	211.5	49.1	1.02	.27
400	260.6	211.5	49.1	1.02	.27
600	260.6	211.5	49.1	1.02	.27

Table 5.5 Time cost error estimates for
MSPG = 5, PGSZ = 2000

ARL	$\ell_{\max}$	$\ell_{\min}$	$\Delta\ell$	$\delta_{\ell} \%$	$\delta_s \%$
50	260.5	211.5	49.0	.8	.33
100	260.6	211.5	49.1	.8	.33
220	260.6	211.5	49.1	.8	.33
400	260.6	211.5	49.1	.8	.33
600	260.6	211.5	49.1	.8	.33

Table 5.6 Time cost error estimates for
MSPG = 4, PGSZ = 2000

ARL	$\ell_{\max}$	$\ell_{\min}$	$\Delta\ell$	$\delta_{\ell} \%$	$\delta_s \%$
50	260.6	211.5	49.1	.4	.11
100	260.6	211.5	49.1	.4	.11
220	260.6	211.5	49.1	.4	.11
400	260.6	211.5	49.1	.4	.11
600	260.6	211.5	49.1	.4	.11

Table 5.7 Time cost error estimates for
MSPG = 2, PGSZ = 2000

ARL	$\ell_{\max}$	$\ell_{\min}$	$\Delta\ell$	$\delta_{\ell} \%$	$\delta_s \%$
50	260.6	211.5	49.1	.13	.054
100	260.6	211.5	49.1	.13	.054
220	260.6	211.5	49.1	.13	.054
400	260.6	211.5	49.1	.13	.054
600	260.6	211.5	49.1	.13	.054

Table 5.8 Time cost error estimates for
MSPG = 1, PGSZ = 2000

and 4000 characters. For each pair, the program was run for some ARL values such as 50, 100, 220, 400 and 600 characters. The average record lengths of all configurations obtained in each of those cases (for all possible storage limits) were on the range from $\ell_{\min}$ to $\ell_{\max}$ characters. The values of $\ell_{\min}$, $\ell_{\max}$ and $\Delta\ell = \ell_{\max} - \ell_{\min}$ are given in the Tables.

For the application example of Chapter 2 with 24 data items, 26 relations and 4 run units, the average cardinality of data items is: $\gamma = 27,070/24 \approx 1128$, and this value is always used for γ . As can be seen from Tables 5.1 to 5.8, δ_ℓ is always greater than δ_s . This is because the size PBSZ of the page buffer is a larger percentage of the size of an average file than a percentage of the total size of the data base. δ_ℓ varies from .13% for PBSZ = 2,000 characters (Table 5.8) to 2.12% for PBSZ = 20,000 characters (Table 5.1), while δ_s varies from .054% for PBSZ = 2,000 characters to .55% for PBSZ = 20,000 characters. The average record lengths of the resulting configurations vary between 211.5 and 260.6 characters, which accounts for a $\Delta\ell$ of about 49 characters.

Very large values of PBSZ result in high error estimates especially in δ_ℓ . However, the value of $\Delta\ell$ does not change very much. For example, for an extreme case of MSPG = 25 pages with PGSZ = 4,000 characters (PBSZ = 100,000 characters), δ_ℓ can be as high as 26.6% while $\Delta\ell$ is between 48.9 and 49.7 characters (Table 5.9).

ARL	$\ell_{\max}$	$\ell_{\min}$	$\Delta \ell$	$\delta_{\ell} \%$	$\delta_s \%$
50	230.0	180.4	49.7	26.6	2.94
220	260.4	211.5	48.9	16.69	2.94
600	260.4	211.5	48.9	16.69	2.94

Table 5.9 Time Cost Error Estimates for
MSPG = 25, PGSZ = 4,000

Results given in Tables 5.1 to 5.8 also show that the values of $\ell_{\max}$ and $\ell_{\min}$ have very small variations with respect to changes in ARL and PBSZ. The value of $\ell_{\max}$ in the above results is between 260.4 and 260.6 characters and $\ell_{\min}$ is 211.5 characters. The optimization program was run for all $\langle \text{MSPG}, \text{PGSZ} \rangle$ pairs, such that MSPG = 1, 2, 4, 5 and PGSZ = 2,000, 4,000, with ARL set to 211.5 and 260.6 characters for each pair. The results of these 16 runs indicate that the average record lengths of optimal configurations (for different storage limits) are still between 211.5 and 260.6 characters. The cumulative storage costs of the optimal configurations are between 1,207,050 and 1,485,350 characters. Storage limits set at values greater than 1,485,350 do not result in any new configurations, i.e. the configuration corresponding to a limit of 1,485,350 characters is optimal for any storage limit greater than or equal to 1,485,350 characters. For storage limit values less than 1,207,050 characters no optimal solution exists because there is not enough storage available to implement the data base for that particular application.

It is evident, from the definition of δ_l and δ_s , that these error estimates present the percentage of possible errors made in the evaluation of the time cost of relation implementations. The preceding results show that for reasonable PBSZ values (2,000 to 20,000 characters) these errors are on the order of a few percentage points. It is important, however, to observe that at the same time the cumulative time costs $CT(C_M)$ and $CT(C_m)$ of configurations C_M and C_m with highest and lowest time costs, respectively, are relatively very far apart. For example, when $MSPG = 1$, $PGSZ = 2,000$ and $ARL = 260.6$, CT_M and CT_m are equal to 135,673 and 52,553, respectively, (in expected page fault rate), for a difference of about 88.3% computed from $100 (CT_M - CT_m) / ((CT_M + CT_m) / 2)$.

Given the results stated above, concerning the approximations in time cost values, we know that the errors introduced as a result of using an estimated ARL value are reasonably small. Further we observed that the average record lengths of the outcoming data base designs stay within an almost fixed boundary. In our example the range (l_{min}, l_{max}) was (211.5, 260.6) which changed to (211.5, 260.5) or (211.5, 260.4) in some cases. We have also demonstrated that adopting an estimate for ARL within this boundary results in only a few percentage error in time costs.

The overall design method we use, then is (1) to execute the optimization program for a wide range of ARL

values, (2) compute the interval $(\ell_{\min}, \ell_{\max})$ in each case and (3) adopt an estimate for ARL in the intersection of all these intervals. For the data base designs discussed in the next section we use an average record length estimate of 211.5 characters.

5.2 Discussion of Resulting Data Base Designs

In this section we shall discuss several alternative data base designs for the application example introduced in Chapter 2. Each alternative is optimal under specific values for parameters of the model.

For example, in the case of the following values of parameters, the data bases shown in Figures 5.1 to 5.6 are obtained.

PTRS (size of a pointer)	4 characters
CNTRS (size of a counter)	6 characters
PGSZ (size of a page)	2,000 characters
PLOKS (size of a privacy lock)	20 characters
MSPG (main storage pages)	2 pages
ARL (Average Record Length)	211.5 characters
TMSB (Maximum Storage Bound)	1,600,000 characters

Figure 5.1 shows the data base description which is optimal for the above values of parameters and a storage limit of 1.35 million characters. The actual storage requirement of that data base is 1,348,350 characters and the expected page fault rate for the data base is 54,694.

The data base shown in Figure 5.1 contains five record

Record Descriptions:

NO.	NAME	LEVEL	ITEM	SIZE	COMMENT
1	ORGREC				/*Organization Record */
		1	ORGCOD	10	/*Organization Code */
		1	ORGNAM	30	/*Name of Organization */
		1	BUDGET	6	/*Budget of Organization */
2	JOBREC				/*Job Record */
		1	JOB COD	10	/*Job Code */
		1	JOB TTL	60	/*Job Title */
		1	ATHQNT	6	/*Authorized Quantity */
		1	SALRGC	6	/*Salary R.G. Count */
		1	SALRPG		/*Salary Repeating Group */
					OCCURS SALRGC TIMES
		2	ATHSAL	6	/*Authorized Salary */
		2	ATHMAX	6	/*Authorized Maximum */
		2	ATHMIN	6	/*Authorized Minimum */
		2	DDUCTN	15	/*Deduction */
					OCCURS 2 TIMES
		1	EMPCNT	6	
		1	EMPNUM	25	/*Employees on the Job */
					OCCURS EMPCNT TIMES
3	EMPREC				/*Employee Record */
		1	EMPNUM	25	/*Employee Number */
		1	EMPNAM	30	/*Employee Name */
		1	EMPADR	100	/*Employee Address */
		1	EMPBRT	8	/*Date of Birth */
		1	EMPMST	2	/*Marital Status */
		1	EMPLVL	2	/*Seniority Level */
		1	JHSYRS	2	/*Years with Organization */
		1	JHSCNT	6	/*Job History Count */
		1	JOB COD	10	/*Jobs Held with Org'n. */
					OCCURS JHSCNT TIMES

Figure 5.1 Data Base Description for a Storage Limit of
1.35 M Bytes, PGSZ = 2000 Bytes and MSPG = 2

NO.	NAME	LEVEL	ITEM	SIZE	COMMENT	
4	OFRREC				/*Offering Record	*/
		1	OFRCOD	25	/*Offering Code	*/
		1	OFRFMT	3	/*Format	*/
		1	OFRDAT	8	/*Date	*/
		1	OFRLOC	10	/*Location	*/
		1	TCHCNT	6	/*Count of Teachers	*/
		1	TCHNUM	25	/*Teachers	*/
					OCCURS TCHCNT TIMES	
		1	COUNUM	15	/*Course Number	*/
		1	STUCNT	6	/*Count of Students	*/
		1	STUNUM	25	/*Students	*/
					OCCURS STUCNT TIMES	
5	COUREC				/*Course Record	*/
		1	COUNUM	15	/*Course Number	*/
		1	COUTTL	60	/*Course Title	*/
		1	COUDSP	250	/*Course Description	*/

Set Descriptions:

NO.	NAME	OWNER	MEMBER	OPTIONS
1	JOBFORG	ORGREC	JOBREC	CHAIN
2	EMPOFORG	ORGREC	EMPREC	CHAIN

Figure 5.1 (Cont'd.)

types called ORGREC, JOBREC, EMPREC, OFRREC and COUREC. The first record type, ORGREC, is the "Organization Record" type that contains data item types ORGCOD, ORGNAM and BUDGET, thereby implementing data base relations NAMOFORG and BGTOFORG defined as first and second data base relations R_1 and R_2 in SR, respectively. The second record type, JOBREC, is a record type describing jobs. It contains data items JOBCOD, JOBTTL, ATHQNT, EMPCNT and EMPNUM, defined at level 1 (of record description) and a repeating group called SALRPG (salary repeating group) whose components ATHSAL, ATHMAX, ATHMIN and DDUCTN are defined at level 2. EMPNUM is a vector of variable length whose length is given by EMPCNT and for each JOBREC record EMPNUM contains employee numbers of employees holding that job. The presence of this vector in the JOBREC record types constitutes the implementation of the relation called JOBOFEMP by implementation 4 (variable duplication of EMPNUM under JOBCOD). The occurrences of EMPNUM values in these records are duplicate copies in contrast to the shared occurrences in instances of the EMPREC (employee record) record type. The latter record type consists of data item types EMPNUM, EMPNAM, EMPADR, EMPBRT, EPMST, EMPLVL, JHSYRS and JHSCNT and a vector of job codes called JOBCOD of variable length given by the value of JHSCNT. This vector records a list of codes of jobs the employee has held with the organization. The juxtaposition of the above data items to form the EMPREC record type implements the following data base

relations: NAMOFEMP, ADROFEMP, BRTOFEMP, MSTOFEMP, LVLOFEMP, YRSOFEMP and JHSOFEMP.

The fourth record type is called OFRREC (offerings record) and it consists of data item types OFRCOD, OFRFMT, OFRDAT, OFRLOC, COUNUM, TCHCNT and STUCNT and vectors TCHNUM and STUNUM of lengths given by the contents of TCHCNT and STUCNT, respectively. The first four data items contain the code, format, date and location of the offering, respectively. The fifth data item COUNUM gives the course number of the offering and the two vectors TCHNUM and STUNUM contain employee numbers of teachers and students of the offering, respectively.

The last record type COUREC (course record) contains data items COUNUM, COUTTL and COUDSP containing the number, title and description of the course respectively.

There are two data base set types defined in Figure 5.1 called JOBOFORG and EMPOFORG. The owner of both set types is ORGREC record type and members of the two sets are JOBREC and EMPREC, respectively. Sets of both types will be implemented by chains with next pointers (without owner or prior pointers). Each ORGREC occurrence is the owner of a data base set (of type EMPOFORG) whose member records are records (of type EMPREC) of employees working for the organization. Each ORGREC is, also, the owner of a data base set (of type JOBOFORG) whose member records are records (of type JOBREC) of jobs within the organization. Data base set types EMPOFORG and JOBOFORG represent the implementation

of relations EMPOFORG and JOBOFORG, respectively by implementation 9 (chain with next pointers association of EMPNUM and JOBCOD under ORGCOD, respectively).

This particular design for the data base requires 1,348,350 characters of storage and a small reduction in the storage limit results in another design becoming optimal, namely the one presented in Figure 5.5. The difference between the two designs is that in the second one (Figure 5.5) the relation called OFROFCOU is implemented by implementation 13 which is the chain with next and prior pointers association of OFRCOD under COUNUM which accounts for the set type OFROFCOU defined in Figure 5.5. Whereas in the previous design (Figure 5.1), the relation OFROFCOU is implemented by implementation 2 (fixed duplication of COUNUM under OFRCOD), which accounts for the presence of the data item COUNUM in the definition of OFRREC record type (Figure 5.1). The configuration associated with the data base description of Figure 5.5 has a cumulative storage cost of 1,348,250 characters and a cumulative time cost of 55,570 in expected page fault rate.

Larger reductions in the storage space limit result in different data base designs. For example, for a storage limit of 1.3 M characters the structure of Figure 5.2 becomes optimal, although at a higher time cost of 58,186 in expected page fault rate. The data base description of Figure 5.2 is different from that of Figure 5.1 in the following ways. Relation JOBOFEMP was implemented by

Record Descriptions:

NO.	NAME	LEVEL	ITEM	SIZE	COMMENT
1	ORGREC				/*Organization Record */
		1	ORGCOD	10	/*Organization Code */
		1	ORGNAM	30	/*Name of Organization */
		1	BUDGET	6	/*Budget of Organization */
2	JOBREC				/*Job Record */
		1	JOB COD	10	/*Job Code */
		1	JOB TTL	60	/*Job Title */
		1	ATHQNT	6	/*Authorized Quantity */
		1	SALRGC	6	/*Salary R.G. Count */
		1	SALRPG		/*Salary Repeating Group */
					OCCURS SALRGC TIMES
		2	ATHSAL	6	/*Authorized Salary */
		2	ATHMAX	6	/*Authorized Maximum */
		2	ATHMIN	6	/*Authorized Minimum */
		2	DDUCTN	15	/*Deductions */
					OCCURS 2 TIMES
3	EMPREC				/*Employee Record */
		1	EMPNUM	25	/*Employee Number */
		1	EMPNAM	30	/*Name of Employer */
		1	EMPADR	100	/*Address of Employee */
		1	EMPBRD	8	/*Date of Birth */
		1	EMPMST	2	/*Marital Status */
		1	EMPLVL	2	/*Seniority Level */
		1	JHSYRS	2	/*Years with Organization */
		1	JHSCNT	6	/*Job History Count */
		1	JOB COD	10	/*Jobs Held with Org'n. */
					OCCURS JHSCNT TIMES

Figure 5.2 Data Base Description for a Storage Limit of 1.3 M Bytes, PGSZ = 2000 Bytes and MSPG = 2

NO.	NAME	LEVEL	ITEM	SIZE	COMMENT	
3	EMPREC				/*Cont'd	*/
		1	JOB COD	10	/*Job of Employee	*/
		1	OFR CNT	6	/*Offering Count	*/
		1	OFR COD	25	/*Student in	*/
					OCCURS OFR CNT TIMES	
4	OFR REC				/*Offering Record	*/
		1	OFR COD	25	/*Offering Code	*/
		1	OFR FMT	3	/*Format	*/
		1	OFR DAT	8	/*Date	*/
		1	OFR LOC	10	/*Location	*/
		1	TCH CNT	6	/*Count of Teachers	*/
		1	TCH NUM	25	/*Teachers	*/
					OCCURS TCH CNT TIMES	
		1	COUNUM	15	/*Course Number	*/
5	COUREC				/*Course Record	*/
		1	COUNUM	15	/*Course Number	*/
		1	COUTTL	60	/*Course Title	*/
		1	COUDSP	250	/*Course Description	*/

Set Descriptions:

NO.	NAME	OWNER	MEMBER	OPTIONS
1	JOB OFORG	ORG REC	JOB REC	CHAIN
2	EMPOFORG	ORG REC	EMP REC	CHAIN

Figure 5.2 (Cont'd.)

Record Descriptions:

NO.	NAME	LEVEL	ITEM	SIZE	COMMENT
1	ORGREC				/*Organization Record */
		1	ORGCOD	10	/*Organization Code */
		1	ORGNAM	30	/*Name of Organization */
		1	BUDGET	6	/*Budget of Organization */
2	JOBREC				/*Job Record */
		1	JOB COD	10	/*Job Code */
		1	JOB TTL	60	/*Job Title */
		1	ATHQNT	6	/*Authorized Quantity */
		1	SALRGC	6	/*Salary R.G. Count */
		1	SALRBG		/*Salary Repeating Group */
					OCCURS SALRGC TIMES
		2	ATHSAL	6	/*Authorized Salary */
		2	ATHMAX	6	/*Authorized Maximum */
		2	ATHMIN	6	/*Authorized Minimum */
		2	DDUCTN	15	/*Deductions */
					OCCURS 2 TIMES
3	EMPREC				/*Employee Record */
		1	EMPNUM	25	/*Employee Number */
		1	EMPNAM	30	/*Name of Employee */
		1	EMPADR	100	/*Address of Employee */
		1	EMPBRT	8	/*Date of Birth */
		1	EMPMST	2	/*Marital Status */
		1	EMPLUL	2	/*Seniority Level */
		1	JHSYRS	2	/*Years with Organization */
		1	JHSCNT	6	/*Job History Count */
		1	JOB COD	10	/*Jobs Held with Org'n. */
					OCCURS JHSCNT TIMES

Figure 5.3 Data Base Description for a Storage Limit of 1,207,050 Bytes, PGSZ = 2000 Bytes and MSPG = 2

NO.	NAME	LEVEL	ITEM	SIZE	COMMENT
4	OFRREC				/*Offering Record */
		1	OFRCOD	25	/*Offering Code */
		1	OFRFMT	3	/*Format */
		1	OFRDAT	8	/*Date */
		1	OFRLOC	10	/*Location */
		1	STUCNT	6	/*Count of Students */
		1	STUNUM	25	/*Employee Nos. of Stdnts*/
OCCURS STUCNT TIMES					
5	COUREC				/*Course Record */
		1	COUNUM	15	/*Course Number */
		1	COUTTL	60	/*Course Title */
		1	COUDSP	250	/*Course Description */

Set Descriptions:

NO.	NAME	OWNER	MEMBER	OPTIONS
1	JOB OFORG	ORGREC	JOBREC	CHAIN
2	EMP OFORG	ORGREC	EMPREC	CHAIN
3	JOB OFEMP	JOBREC	EMPREC	CHAIN
4	TEACHERS	OFRREC	EMPREC	CHAIN
5	OFROFCOU	COUREC	OFRREC	CHAIN

Figure 5.3 (Cont'd.)

Record Descriptions:

NO.	NAME	LEVEL	ITEM	SIZE	COMMENT
1	ORGREC				/*Organization Record */
		1	ORGCOD	10	/*Organization Code */
		1	ORGNAM	30	/*Name of Organization */
		1	BUDGET	6	/*Budget of Organization */
2	JOBREC				/*Job Record */
		1	JOBCOD	10	/*Job Code */
		1	JOBTTL	60	/*Job Title */
		1	ATHQNT	6	/*Authorized Quantity */
		1	SALRGC	6	/*Salary R.G. Count */
		1	SALRPG		/*Salary Repeating Group */
					OCCURS SALRGC TIMES
		2	ATHSAL	6	/*Authorized Salary */
		2	ATHMAX	6	/*Authorized Maximum */
		2	ATHMIN	6	/*Authorized Minimum */
		2	DDUCTN	15	/*Deductions */
					OCCURS 2 TIMES
3	EMPREC				/*Employee Record */
		1	EMPNUM	25	/*Employee Number */
		1	EMPNAM	30	/*Name of Employee */
		1	EMPADR	100	/*Address of Employee */
		1	EMPBRT	8	/*Date of Birth */
		1	EMPMST	2	/*Marital Status */
		1	EMPLUL	2	/*Seniority Level */
		1	JHSYRS	2	/*Years with Organization*/
		1	JHSCNT	6	/*Job History Count */
		1	JOBCOD	10	/*Jobs Held with Org'n. */
					OCCURS JHSCNT TIMES

Figure 5.4 Data Base Description for a Storage Limit of 1,227,650 Bytes, PGSZ = 2000 Bytes and MSPG = 2

NO.	NAME	LEVEL	ITEM	SIZE	COMMENT	
4	OFRREC				/*Offering Record	*/
		1	OFRCOD	25	/*Offering Code	*/
		1	OFRFMT	3	/*Format	*/
		1	OFRDAT	8	/*Date	*/
		1	OFRLOC	10	/*Location	*/
		1	STUCNT	6	/*Count of Students	*/
		1	STUNUM	25	/*Students	*/
OCCURS STUCNT TIMES						
5	COUREC				/*Course Record	*/
		1	COUNUM	15	/*Course Number	*/
		1	COUTTTL	60	/*Course Title	*/
		1	COUDSP	250	/*Course Description	*/

Set Descriptions:

NO.	NAME	OWNER	MEMBER	OPTIONS
1	JOB OFORG	ORGREC	JOBREC	CHAIN
2	EMPOFORG	ORGREC	EMPREC	CHAIN
3	JOB OFEMP	JOBREC	EMPREC	CHAIN WITH OWNER POINTERS
4	TEACHERS	OFRREC	EMPREC	CHAIN
5	OFROFCOU	COUREC	OFRREC	CHAIN WITH PRIOR POINTERS

Figure 5.4 (Cont'd.)

Record Descriptions:

NO.	NAME	LEVEL	ITEM	SIZE	COMMENT
1	ORGREC				/*Organization Record */
		1	ORGCOD	10	/*Organization Code */
		1	ORGNAM	30	/*Name of Organization */
		1	BUDGET	6	/*Budget of Organization */
2	JOBREC				/*Job Record */
		1	JOB COD	10	/*Job Code */
		1	JOB TTL	60	/*Job Title */
		1	ATHQNT	6	/*Authorized Quantity */
		1	SALRGC	6	/*Salary R.G. Count */
		1	SALRPG		/*Salary Repeating Group */
					OCCURS SALRGC TIMES
		2	ATHSAL	6	/*Authorized Salary */
		2	ATHMAX	6	/*Authorized Maximum */
		2	ATHMIN	6	/*Authorized Minimum */
		2	DDUCTN	15	/*Deductions */
					OCCURS 2 TIMES
		1	EMPCNT	6	
		1	EMPNUM	25	/*Employees on the Job */
					OCCURS EMPCNT TIMES
3	EMPREC				/*Employee Record */
		1	EMPNUM	25	/*Employee Number */
		1	EMPNAM	30	/*Employee Name */
		1	EMPADR	100	/*Employee Address */
		1	EMPBRT	8	/*Date of Birth */
		1	EMPMST	2	/*Marital Status */
		1	EMPLVL	2	/*Seniority Level */
		1	JHSYRS	2	/*Years with Organization */
		1	JHSCNT	6	/*Job History Count */
		1	JOB COD	10	/*Jobs Held with Org'n. */
					OCCURS JHSCNT TIMES

Figure 5.5 Data Base Description for a Storage Limit of
 1,348,250 Bytes, PGSZ = 2000 Bytes and MSPG = 2

NO.	NAME	LEVEL	ITEM	SIZE	COMMENT	
4	OFRREC				/*Offering Record	*/
		1	OFRCOD	25	/*Offering Code	*/
		1	OFRFMT	3	/*Format	*/
		1	OFRDAT	8	/*Date	*/
		1	OFRLOC	10	/*Location	*/
		1	TCHCNT	6	/*Count of Teachers	*/
		1	TCHNUM	25	/*Teachers	*/
			OCCURS TCHCNT TIMES			
		1	STUCNT	6	/*Count of Students	*/
		1	STUNUM	25	/*Students	*/
			OCCURS STUCNT TIMES			
5	COUREC				/*Course Record	*/
		1	COUNUM	15	/*Course Number	*/
		1	COUTTL	60	/*Course Title	*/
		1	COUDSP	250	/*Course Description	*/

Set Descriptions:

NO.	NAME	OWNER	MEMBER	OPTIONS
1	JOBFOFORG	ORGREC	JOBREC	CHAIN
2	EMPOFOFORG	ORGREC	EMPREC	CHAIN
3	OFROFCOU	COURER	OFRREC	CHAIN WITH NEXT AND PRIOR POINTERS

Figure 5.5 (Cont'd.)

implementation 4 in Figure 5.1 and therefore we had a variable length vector of employee numbers EMPNUM (length given by EMPCNT in JOBREC) in the JOBREC record type, whereas in Figure 5.2 this relation is implemented by implementation 1 and therefore JOBREC record type does not contain EMPNUM vector and EMPCNT counter, instead a JOBCOD data item in the EMPREC record type that will contain the job code of the employee) is defined to implement JOBOFEMP relation. We recall that a) implementation 4 for relation JOBOFEMP is the variable duplication of EMPNUM under JOBCOD and b) implementation 1 for JOBOFEMP is the fixed duplication of JOBCOD under EMPNUM, where $EMPNUM = ORG(JOBOFEMP)$ and $JOBCOD = DST(JOBOFEMP)$.

The relation called STUDENTS is also implemented differently in the two designs. In the data base design described in Figure 5.1 this relation is implemented by implementation 3; the variable duplication of EMPNUM under OFRCOD, where $OFRCOD = ORG(STUDENTS)$ and $EMPNUM = DST(STUDENTS)$. Therefore in Figure 5.1 a variable length vector of employee numbers called STUNUM is included in the definition of OFRREC and the length of the vector is given by the value of a counter data item type called STUCNT also defined in OFRREC record type. On the other hand, relation STUDENTS is implemented by implementation 4, variable duplication of OFRCOD under EMPNUM, in the data base design described in Figure 5.2. As can be seen in Figure 5.2, a variable length vector of offering codes

called OFRCOD is included in the definition of EMPREC record type; the length of the vector is given by the contents of another data item type, called OFRCNT, also included in the definition of EMPREC record type. The elements of the vector OFRCOD in a particular employee's record (type EMPREC) contain offering codes of offerings in which that particular employee is registered as a student.

Further reductions in the storage space limit result in different data base designs with higher time costs. If the storage limit is reduced to 1,227,650 characters, the data base design described in Figure 5.4 becomes optimal and it has a cumulative time cost of 76,846 in expected page fault rate. This design includes five data base set type definitions. It is different from the data base design of Figure 5.1 in the following ways. The data base relation called JOBOFEMP is implemented, in Figure 5.4, by implementation 12 (called chain with next and owner pointers association of EMPNUM under JOBCOD, where $EMPNUM = ORG(JOBOFEMP)$ and $JOBCOD = DST(JOBOFEMP)$). That is why, in Figure 5.4, a data base set type called JOBOFEMP is defined (set type number 3) whose owner and member record types are JOBREC and EMPREC record types, respectively, and as can be seen in the Figure, the OWNER POINTER option is specified in the definition of the set type. Whereas in the data base design of Figure 5.1, relation JOBOFEMP was implemented by implementation 4 and thereby employee numbers of

employees holding a job were duplicated in a variable length vector in the JOBREC occurrence of that job. The implementations of the relation called TEACHERS is also different in the two data base designs. Employee numbers of teachers of a given offering (a specific offering code) were stored in the elements of a vector called TCHNUM in the OFRREC record of the offering, in the data base description of Figure 5.1, because TEACHERS relation was implemented by implementation 3. Whereas in the design shown in Figure 5.4, this relation is implemented by implementation 9 (chain with next pointers association of EMPNUM under OFRCOD). This is the reason why a data base set type called TEACHERS is defined in Figure 5.4. The owner and member record types of this set type are OFRREC and EMPREC record types, respectively. Each occurrence of the TEACHERS set type consists of exactly one record of OFRREC type, corresponding to some OFRCOD value, and zero or more record(s) of EMPREC type, corresponding to employees who teach that offering. Finally, the relation called OFROFCOU, implemented by implementation 2 in the data base design of Figure 5.1, is now implemented by implementation 13 (chain with next and prior pointers association of OFRCOD under COUNUM). We recall that $OFRCOD = DST(OFROFCOU)$ and $COUNUM = ORG(OFROFCOU)$. In the data base design of Figure 5.1, for each record of type OFRREC (corresponding to an offering code OFRCOD), the associated course number is duplicated in the data item COUNUM which is included in the

definition of OFRREC. However, in the data base design of Figure 5.4, since OFROFCOU is implemented by implementation 13, a data base set type is defined with owner and member record types declared to be COUREC and OFRREC, respectively (set type number 5 in Figure 5.4).

If we continue reducing the storage space limit, progressively more costly data base designs become optimal. However, if the limit is reduced to a value below 1,207,050 characters, then there are no acceptable solutions simply because the sum of storage costs of the least costly (in terms of storage) implementation for each relation is 1,207,050. The data base design which is optimal for a storage limit of 1,207,050 is described in Figure 5.3. Actually for the storage limit of 1,207,050 characters, this is the only acceptable solution and hence it is optimal. The data base design described in Figure 5.3 is different from the one in 5.4 (discussed above) in the set implementation technique option for set types 3 and 5. Set types number 3 and 5 in Figure 5.4 are defined with the OWNER POINTERS and PRIOR POINTERS options specified, respectively. Whereas in the design of Figure 5.3 these two data base sets (JOBOFEMP and OFROFCOU) use the simple chain with next pointers only, in the occurrences of the set types. The data base design of Figure 5.3 has an average record length of 211.5 characters.

Increasing the storage limit, on the other hand, also results in new data base designs becoming optimal. For

example, increasing the limit to 1,485,350 characters causes the data base described in Figure 5.6 to become optimal. We note that in this design of the data base no set types are defined. Increasing the storage space limit beyond 1,485,350 characters, however, does not change the optimal design because there is no acceptable configuration whose time cost is less than the time cost of the configuration associated with the data base design shown in Figure 5.6.

5.3 Storage/Time Trade-Off

In the preceding section we observed that for a given application—namely a given set of data items, data base relations, user activities and storage space parameters—a different data base design becomes optimal when the limit on storage space is varied over a certain range of values $S_{\min}$ and $S_{\max}$. For example, in the case of the example presented in Section 5.2, the range of storage limits was from 1,207,050 to 1,485,350 characters. There are certain storage limit values in this range where a new data base design becomes optimal. We will call these points on the storage space scale, breakpoints. The breakpoints on the storage space limit values correspond, one-to-one, to the sequence of undominated solutions of the final stage of optimization. We know from Chapter 4 that if this sequence is ordered in increasing order of storage costs, then it is also ordered in the strictly decreasing order of time costs.

Record Descriptions:

NO.	NAME	LEVEL	ITEM	SIZE	COMMENT
1	ORGREC				/*Organization Record */
		1	ORGCOD	10	/*Organization Code */
		1	ORGNAM	30	/*Name of Organization */
		1	BUDGET	6	/*Budget of Organization */
		1	JOBCNT	6	/*Count of Jobs */
		1	JOBCOD	10	/*Jobs in the Org'n. */
					OCCURS JOBCNT TIMES
		1	EMPCNT	6	/*Count of Employees */
		1	EMPNUM	25	/*Employees of the Org'n.*/
					OCCURS EMPCNT TIMES
2	JOBREC				/*Job Record */
		1	JOBCOD	10	/*Job Code */
		1	JOBTTT	60	/*Job Title */
		1	ATHQNT	6	/*Authorized Quantity */
		1	SALRGC	6	/*Salary R.G. Count */
		1	SALRPG		/*Salary Repeating Group */
					OCCURS SALRGC TIMES
		2	ATHSAL	6	/*Authorized Salary */
		2	ATHMAX	6	/*Authorized Maximum */
		2	ATHMIN	6	/*Authorized Minimum */
		2	DDUCTN	15	/*Deduction */
					OCCURS 2 TIMES
		1	EMPCNT	6	
		1	EMPNUM	25	/*Employees on the Job */
					OCCURS EMPCNT TIMES

Figure 5.6 Data Base Description for a Storage Limit of 1,485,350 (or more) Bytes, PGSZ = 2000 and MSPG = 2

NO.	NAME	LEVEL	ITEM	SIZE	COMMENT
3	EMPREC				/*Employee Record */
		1	EMPNUM	25	/*Employee Number */
		1	EMPNAM	30	/*Employee Name */
		1	EMPADR	100	/*Employee Address */
		1	EMPBRT	8	/*Date of Birth */
		1	EMPMST	2	/*Marital Status */
		1	EMPLVL	2	/*Seniority Level */
		1	JHSYRS	2	/*Years with Organization*/
		1	JHSCNT	6	/*Job History Count */
		1	JOBCOD	10	/*Jobs Held with Org'n. */
					OCCURS JHSCNT TIMES
4	OFRREC				/*Offering Record */
		1	OFRCOD	25	/*Offering Code */
		1	OFRFMT	3	/*Format */
		1	OFRDAT	8	/*Date */
		1	OFRLOC	10	/*Location */
		1	TCHCNT	6	/*Count of Teachers */
		1	TCHNUM	25	/*Teachers */
					OCCURS TCHCNT TIMES
		1	COUNUM	15	/*Course Number */
		1	STUCNT	6	/*Count of Students */
		1	STUNUM	25	/*Students */
					OCCURS STUCNT TIMES
5	COUREC				/*Course Record */
		1	COUNUM	15	/*Course Number */
		1	COUTTL	60	/*Course Title */
		1	COUDSP	250	/*Course Description */

Figure 5.6 (Cont'd.)

For each storage space limit on the range between two consecutive breakpoints $S_k < S_{k+1}$, including S_k and excluding S_{k+1} , the configuration (and, hence the data base design) associated with S_k is optimal, $k = 1, \dots, NSS$ where NSS is the number of solutions to the final stage of optimization and $S_{\min} = S_1 < S_2 < \dots < S_{NSS} = S_{\max}$.

We can illustrate the relationship between storage and time costs of optimal configurations in a diagram such as the one shown in Figure 5.7. In this diagram the vertical axis represents time costs in expected number of page faults and the horizontal axis represents storage costs in number of characters of storage. Each discontinuity point on the graph corresponds to a breakpoint on the storage space limit. There are a total of 45 breakpoints on the graph shown in Figure 5.7 on a range of storage costs from 1,207,050 to 1,485,350 characters. Time costs vary from 135,161 to 52,492 in expected number of page faults. It can be seen from the Figure that a sharp rise occurs in time costs when storage limit is reduced to below 1,227,050 characters. This rise of about 56,889 units in time cost when storage is reduced to below 1,227,050 characters, is due mainly to the difference in time costs of implementations 10 and 12 for relation JOBOFEMP. Implementation 10 for this relation has a time cost of 90,086 and implementation 12 has a time cost of 31,807 units. Implementation 12, we recall, is similar to implementation 10 except for the extra owner pointers that are present in implementation

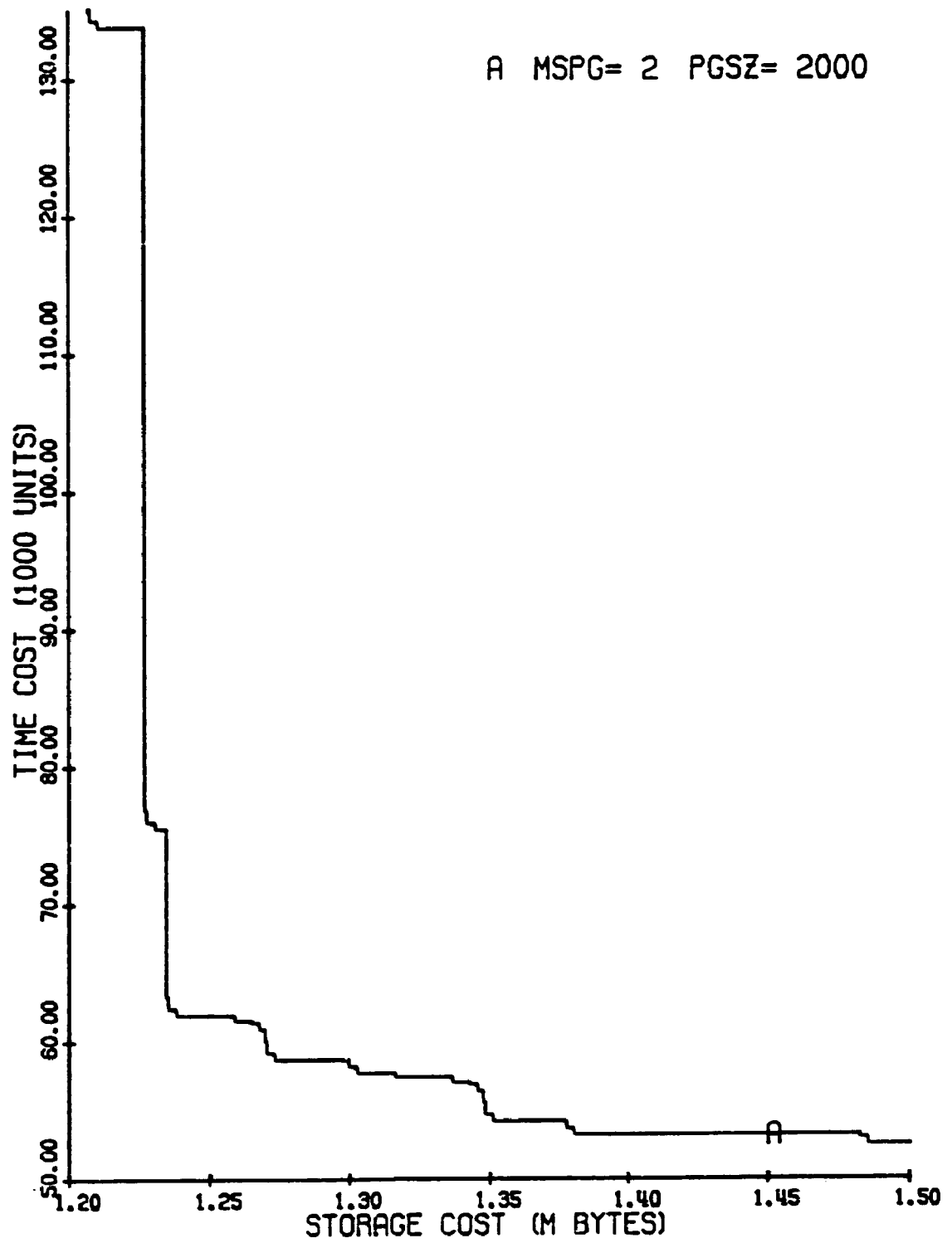


Figure 5.7 Storage/Time Trade-off Graph

12. This is an example of the case where a rather small storage overhead (20,000 characters) for the owner pointers in a specific set type can make a relatively large saving in time costs (58,279 units).

The storage/time trade-off graph exemplified in Figure 5.7 should be extended infinitely to the right along the horizontal line through the point on the graph corresponding to $S_{\max} = CS(C_M)$. This is because all storage limits greater than $S_{\max}$ will result in the same configuration C_M to become optimal and the (cumulative) time cost of this configuration $CT(C_M)$ is actually the lowest time cost achievable under any storage limit. Similarly since no acceptable configuration is attainable which has a storage cost less than $S_{\min} = CS(C_m)$, we associate a time cost of infinity with such configurations. This is demonstrated graphically by extending the storage/time trade-off graph upward on the vertical line through the point on the graph corresponding to $S_{\min}$.

The idea behind choosing a two-component cost function was to avoid the assessment of a relative measure of importance of time costs vs. storage costs which may not be known, at least at the time the data base is first designed. If, however, such information is available (possibly at a data base reorganization point) it can be used in conjunction with the storage/time trade-off graph to satisfy both objectives. In the simplest case, for example, one can assume some dollar cost a for a unit of time cost and b for

a unit of storage cost which amounts to a linear additive measure of cost such as $COST = aS + bT$ for S units of storage cost and T units of time cost. In this case points of equal $COST$ lie on straight lines whose slopes depend on $a:b$ ratio and are located farther from the origin for higher $COST$ s. The designer can, then, easily find the minimum $COST$ point on the trade-off graph by superimposing it with parallel lines of constant $COST$. The underlying assumption here is that the storage space limit is not a severe design constraint and the designer has some flexibility over the adjustment of this parameter in the model.

In the next section we will examine the effect of changing various parameters of the model (mainly storage related parameters) on the outcome of the optimization process. At this point, however, we present the optimal design of the data base for a different set of parameter values, which is structurally different from other data base designs in this section.

For the following values of parameters and a storage space limit of 1,194,000 characters the data base design of Figure 5.8 becomes optimal.

PTRS = 3 characters; DRSZ = $4 * PTRS = 12$ characters,
 MSPG = 2 pages, PGSZ = 2,000 characters,
 CNTRS = 6 characters
 PLOKS = 20 characters.

The difference between these values of parameters as compared to those for Figures 5.1 to 5.6 is in the value of

Record Descriptions:

NO.	NAME	LEVEL	ITEM	SIZE	COMMENT
1	ORGREC				/*Organization Record */
		1	ORGCOD	10	/*Organization Code */
		1	ORGNAM	30	/*Name of Organization */
		1	BUDGET	6	/*Budget of Organization */
2	JOBREC				/*Job Record */
		1	JOB COD	10	/*Job Code */
		1	JOB TTL	60	/*Job Title */
		1	ATHQNT	6	/*Authorized Quantity */
		1	SALRGC	6	/*Salary R.G. Count */
		1	SALRPG		/*Salary Repeating Group */
					OCCURS SALRGC TIMES
		2	ATHSAL	6	/*Authorized Salary */
		2	ATHMAX	6	/*Authorized Maximum */
		2	ATHMIN	6	/*Authorized Minimum */
		2	DDUCTN	15	/*Deductions */
					OCCURS 2 TIMES
3	EMPREC				/*Employee Record */
		1	EMPNUM	25	/*Employee Number */
		1	EMPNAM	30	/*Name of Employee */
		1	EMPADR	100	/*Address of Employee */
		1	EMP BRT	8	/*Date of Birth */
		1	EMPMST	2	/*Marital Status */
		1	EMPLVL	2	/*Seniority Level */
		1	JHSYRS	2	/*Years with Organization */
		1	JHSCNT	6	/*Job History Count */
		1	JOB COD	10	/*Jobs Held with Org'n. */
					OCCURS JHSCNT TIMES

Figure 5.8 Data Base Description for a Storage Limit of
1'194'000 Bytes, PGSZ = 2000 Bytes and MSPG = 2

NO.	NAME	LEVEL	ITEM	SIZE	COMMENT
4	OFRREC				/*Offering Record */
		1	OFRCOD	25	/*Offering Code */
		1	OFRFMT	3	/*Format */
		1	OFRDAT	8	/*Date */
		1	OFRLOC	10	/*Location */
5	COUREC				/*Course Record */
		1	COUNUM	15	/*Course Number */
		1	COUTTL	60	/*Course Title */
		1	COUDSP	250	/*Course Description */
6	STULNK				/*Student Link Record; This is a Dummy*/
					/*Record and Contains no Data */

Set Descriptions:

NO.	NAME	OWNER	MEMBER	OPTIONS
1	JOB OFORG	ORGREC	JOBREC	CHAIN
2	EMPOFORG	ORGREC	EMPREC	CHAIN
3	JOB OFEMP	JOBREC	EMPREC	CHAIN WITH OWNER POINTERS
4	TEACHERS	OFRREC	EMPREC	CHAIN
5	OFROFCOU	COUREC	OFRREC	CHAIN WITH PRIOR POINTERS
6	STUDNTOF	EMPREC	STULNK	CHAIN WITH OWNER POINTERS
7	STUDENTS	OFRREC	STULNK	CHAIN WITH OWNER POINTERS

Figure 5.8 (Cont'd.)

the pointer size PTRS which is reduced from 4 to 3 characters. This reduction in the size of an address occurrence makes implementation 21 less costly in terms of storage because dummy records of this implementation are totally made up of address occurrences. We recall that implementation 21 for a relation such as $R(A, B)$, defined in 3.1.11, is called the Dummy Record Association of A and B, and that this implementation can be used to implement $R(A, B)$ even if the relation is a many-to-many relation. One example of such a relation is the 23rd relation defined in the application described in Chapter 2 and it is called STUDENTS. The origin of the relation is OFRCOD and the destination of the relation is EMPNUM, and it relates employee numbers of employees registered as students to corresponding offerings. The relation is generally a many-to-many relation because a student may be registered in many offerings while at the same time many students are registered for a given offering. The data base design described in Figure 5.8 is to some extent similar to the one shown in Figure 5.4 and therefore we only mention their differences.

The source of difference between the two data base designs is in the implementation of STUDENTS relation. In the design described in Figure 5.4 this relation is implemented by variable duplication of EMPNUM under OFRCOD (implementation 3) whereas in the data base design of Figure 5.8 the STUDENTS relation is implemented by the dummy record association of EMPNUM and OFRCOD (implementation 21). This

necessitates, in the design shown in Figure 5.8, the definition of a new record type called STULNK (student link record). No data item types are defined as components of this record type and occurrences of this record type are composed only of pointers that establish their membership in certain data base sets. There are also two new data base set types defined in Figure 5.8, called STUDNTOF and STUDENTS. The STULNK record type is declared as the member record type in both set types and EMPREC and OFRREC record types are declared as owner record types of STUDNTOF and STUDENTS set types, respectively. Sets of these two types are to be implemented by chaining STULNK member records to each other and to the respective member records through next and owner pointers. Each employee record (type:EMPREC) is the owner of a (possibly empty, if the employee is not registered as a student in any offering) data base set of STUDNTOF type. Member records of this set are dummy records of STULNK type and each one of these member records also participates in exactly one data base set of STUDENTS type whose owner, in turn, is a record of type OFRREC. In this fashion each STULNK record corresponds to exactly one ordered pair $\langle a, b \rangle$ in the relation STUDENTS, where a and b are OFRCOD and EMPNUM values. This record is a member of exactly one set of STUDNTOF type whose owner contains b and it is also a member in exactly one set of STUDENTS type whose owner record contains a . Storage/time trade-off graph for MSPG = 2, PGSZ = 2,000 characters and PTRS = 3 charac-

ters is given in Figure 5.9.

5.4 Effects of Some Parameter Variations

In this section we will examine the effect of changing some of the parameters, especially in the storage spare model, on the outcoming designs and particularly on the storage/time trade-off graphs discussed in the previous section.

The effect of changing MSPG, the number of pages of data that may be in main storage at the same time, while keeping all other parameters fixed is shown in Figure 5.10. This Figure shows the storage/time graph for PGSZ = 2,000 characters, PTRS = 4 characters, CNTRS = 6 characters and MSPG = 1, 2, 4 and 5 pages. A smaller portion of the four graphs have been redrawn with different scales (the scale of vertical axis increased to 400%, and that of horizontal axis reduced to 80% of the original scales). This graph shows that for fixed PGSZ an increase in the number of main storage pages, MSPG, results in the overall reduction of time costs. The reduction is, however, not significant when MSPG is increased from 1 to 2 or from 4 to 5. The range of time costs in the four graphs for MSPG = 1, 2, 4 and 5 is 52,553 to 135,602, 52,492 to 135,161, 52,354 to 134,269 and 52,300 to 133,833, respectively.

The observation that for a fixed storage limit, the time cost of the optimal configuration decreases when MSPG is increased is not surprising because higher MSPG values

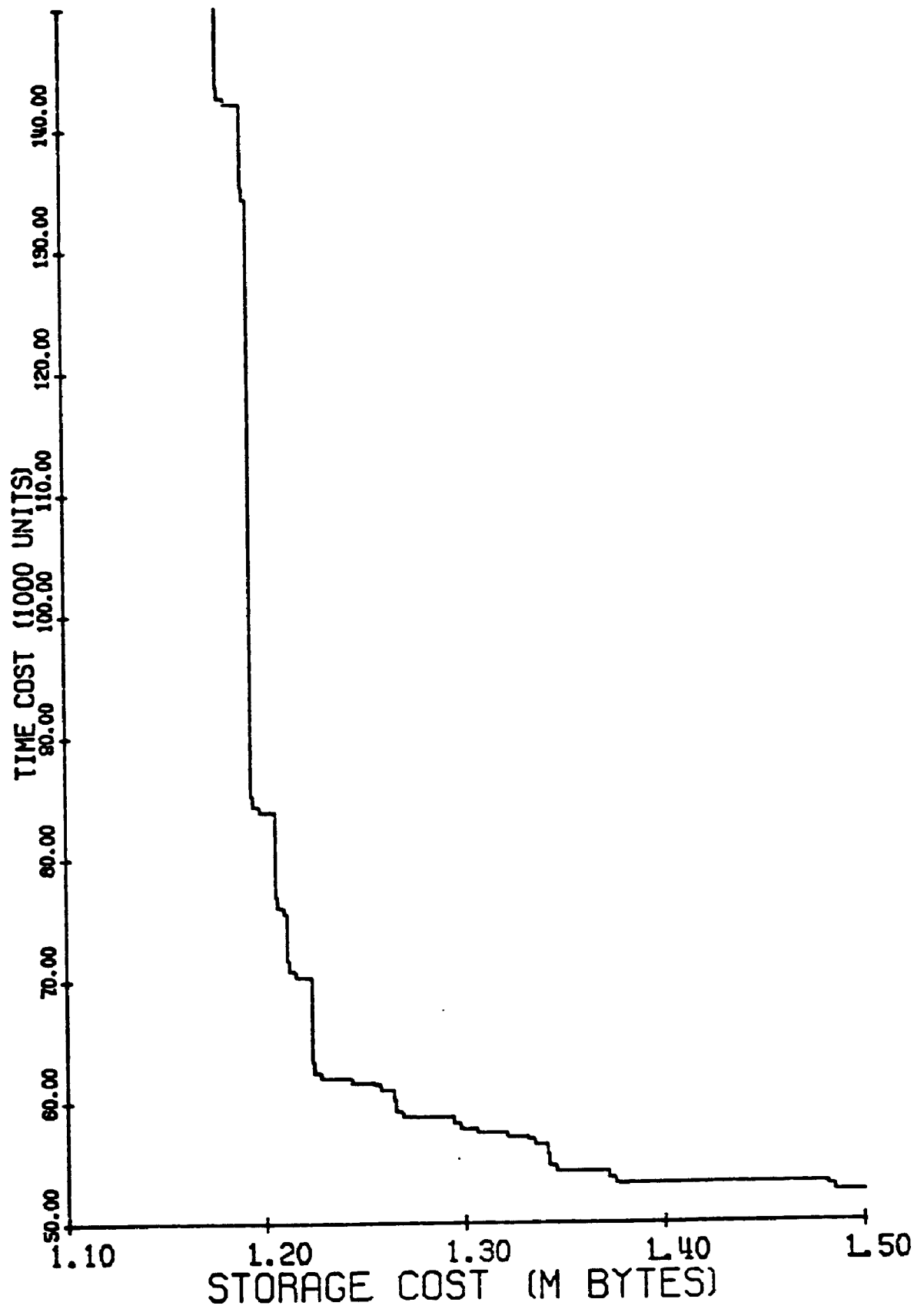


Figure 5.9 Trade-off Graph for PTRS = 3

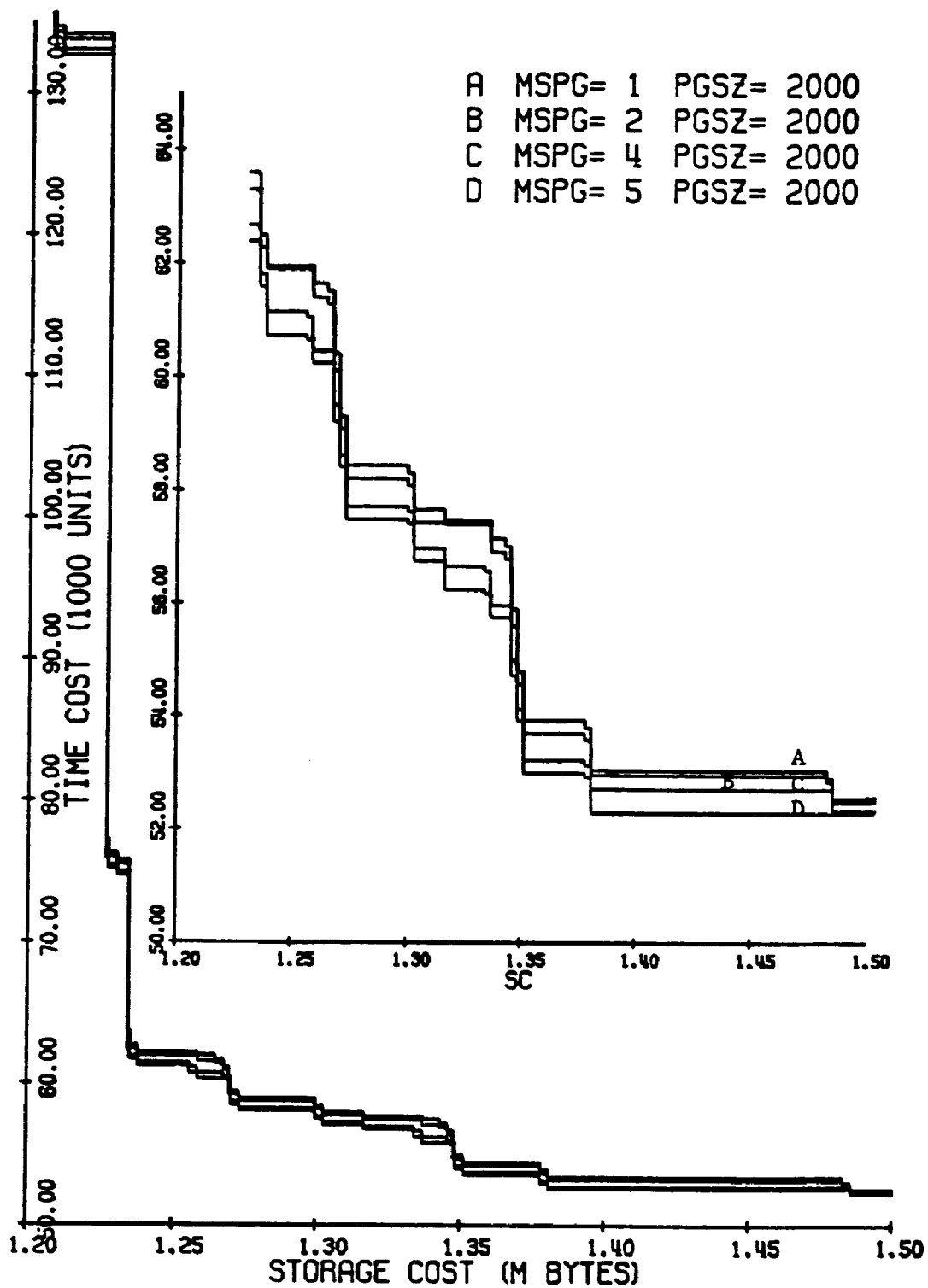


Figure 5.10 Trade-off Graphs for Different MSPG Values where PGSZ = 2000 Characters

account for larger page buffer sizes (for fixed PGSZ). Larger PGSZ values, in turn, make it possible to keep higher percentages of data in main storage, hence fewer page faults are expected to occur in those cases.

Figure 5.11 shows the storage/time graphs for PGSZ = 4,000 characters and MSPG = 1, 2, 4 and 5. Graphs shown in this Figure have the same characteristics as the graphs shown in Figure 5.10, namely that, for fixed PGSZ, increasing MSPG results in an overall decrease of time costs. Graphs of Figure 5.11 are relatively farther apart with respect to those shown in Figure 5.10. This is because page size values used in graphs of Fig. 5.11 are larger than those of Fig. 5.10. The conclusion here is that for higher page sizes, time cost values are more sensitive to changes in MSPG.

Individual configurations are usually identical for all eight graphs. We recall that each discontinuity point on a graph corresponds to a breakpoint in the storage cost axis which in turn corresponds to a specific configuration. The graphs of Figures 5.10 and 5.11 show that these discontinuities almost always occur at the same abscissae. Comparing Figures 5.10 and 5.11 one notices, also, that each graph for PGSZ = 4,000 falls below that for PGSZ = 2,000 for the same MSPG. In other words, for a fixed number of available pages MSPG, larger page sizes result in lower overall page faults, which is to be expected. This is more clearly illustrated in Figures 5.12

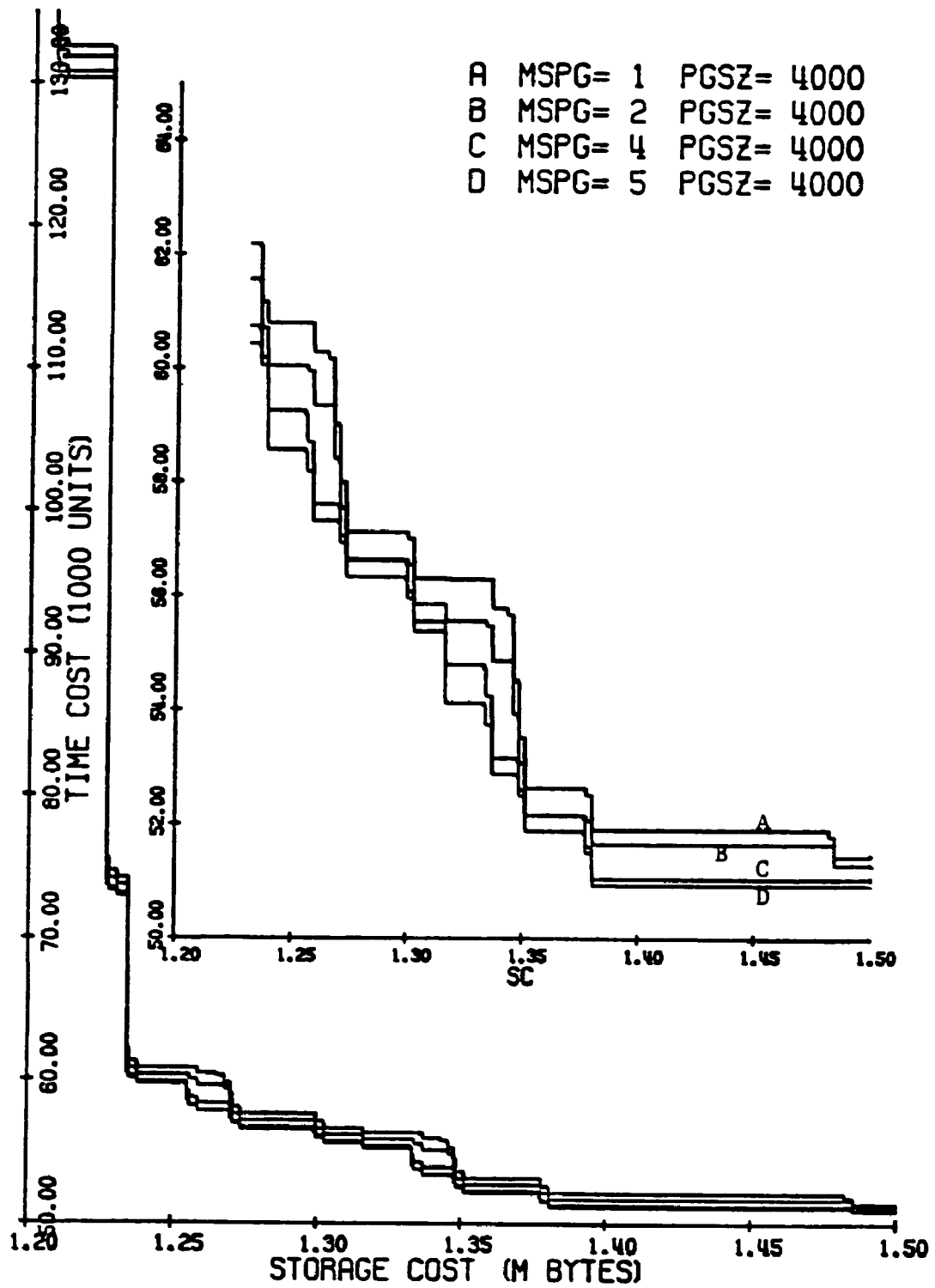


Figure 5.11 Trade-off Graphs for Different MSPG Values where PGSZ = 4000 Characters

through 5.15 where graphs of equal MSPG values but different PGSZ values are drawn together. It should be noted, however, that we are not taking into consideration the higher transport time for larger pages.

Another observation is that the product (rather than individual values) of PGSZ and MSPG controls the relative positioning of the storage/time graphs. This can be seen by comparing the graphs for (MSPG = 2, PGSZ = 2000) and (MSPG = 1, PGSZ = 4,000); or those for (MSPG = 4, PGSZ = 2,000) and (MSPG = 2, PGSZ = 4,000). Figure 5.16 illustrates this point. In Figure 5.16 the storage/time graphs are given for the three combinations (MSPG = 1, PGSZ = 2,000), (MSPG = 5, PGSZ = 2,000) and (MSPG = 5, PGSZ = 4,000). Note that the page buffer size PBSZ is equal to 2,000, 10,000 and 20,000 characters, respectively, for the above three combinations.

In order to investigate the effect of variations in the pointer size parameter PTRS, we hold all other parameter values, used for the storage/time trade-off graph of Figure 5.7, constant and generate the graph for PTRS values of 6, 8 and 16 characters. Note that we have shown the individual graphs of the two cases where PTRS = 3 and PTRS = 4, in Figures 5.7 and 5.9, respectively. Figure 5.17 shows all five graphs on the same set of coordinates.

We have already seen one of the effects of reducing the pointer size, namely that implementations which make extensive use of pointers become less expensive (in terms of

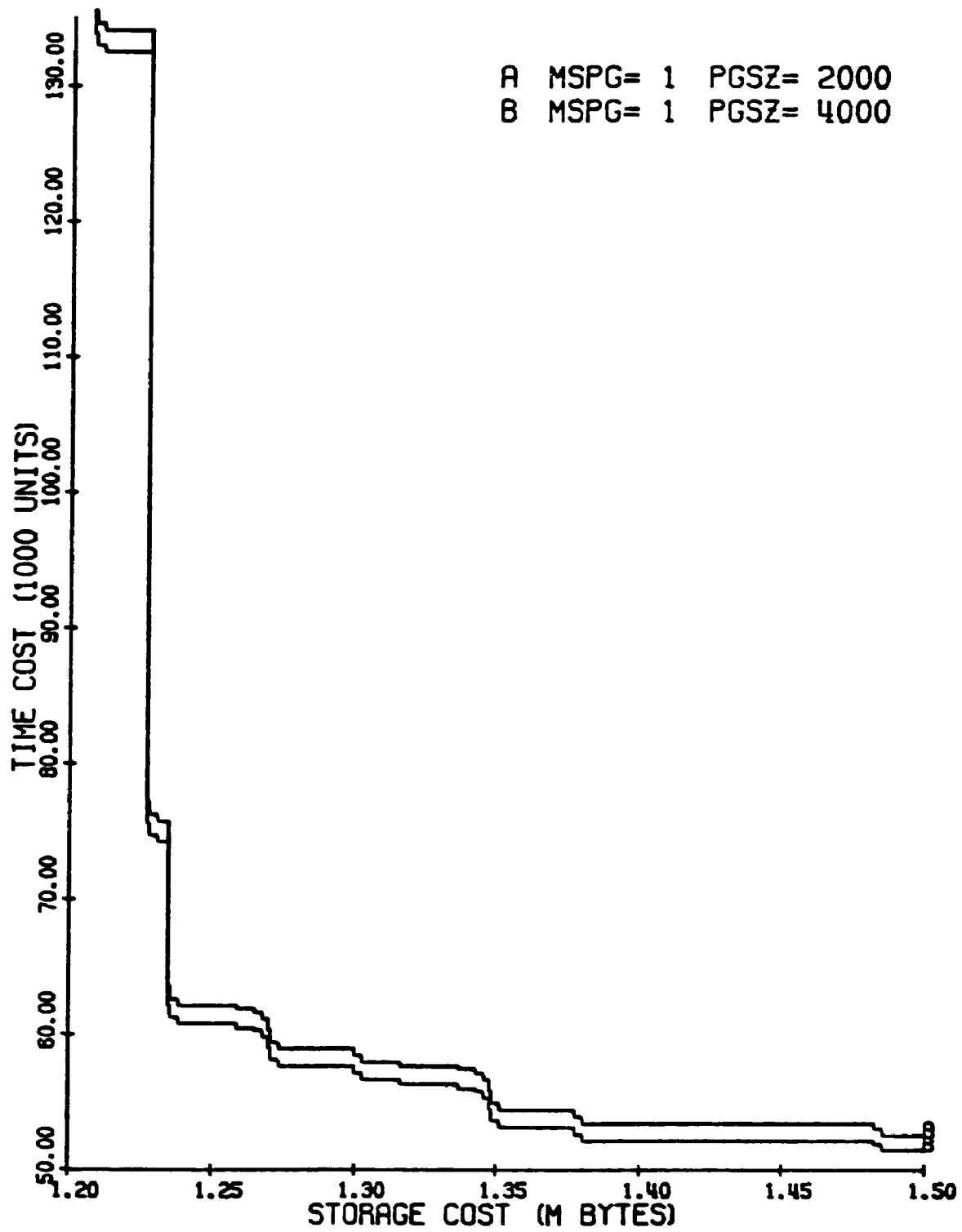


Figure 5.12 Trade-off Graphs for Different PGSZ Values
where MSPG = 1

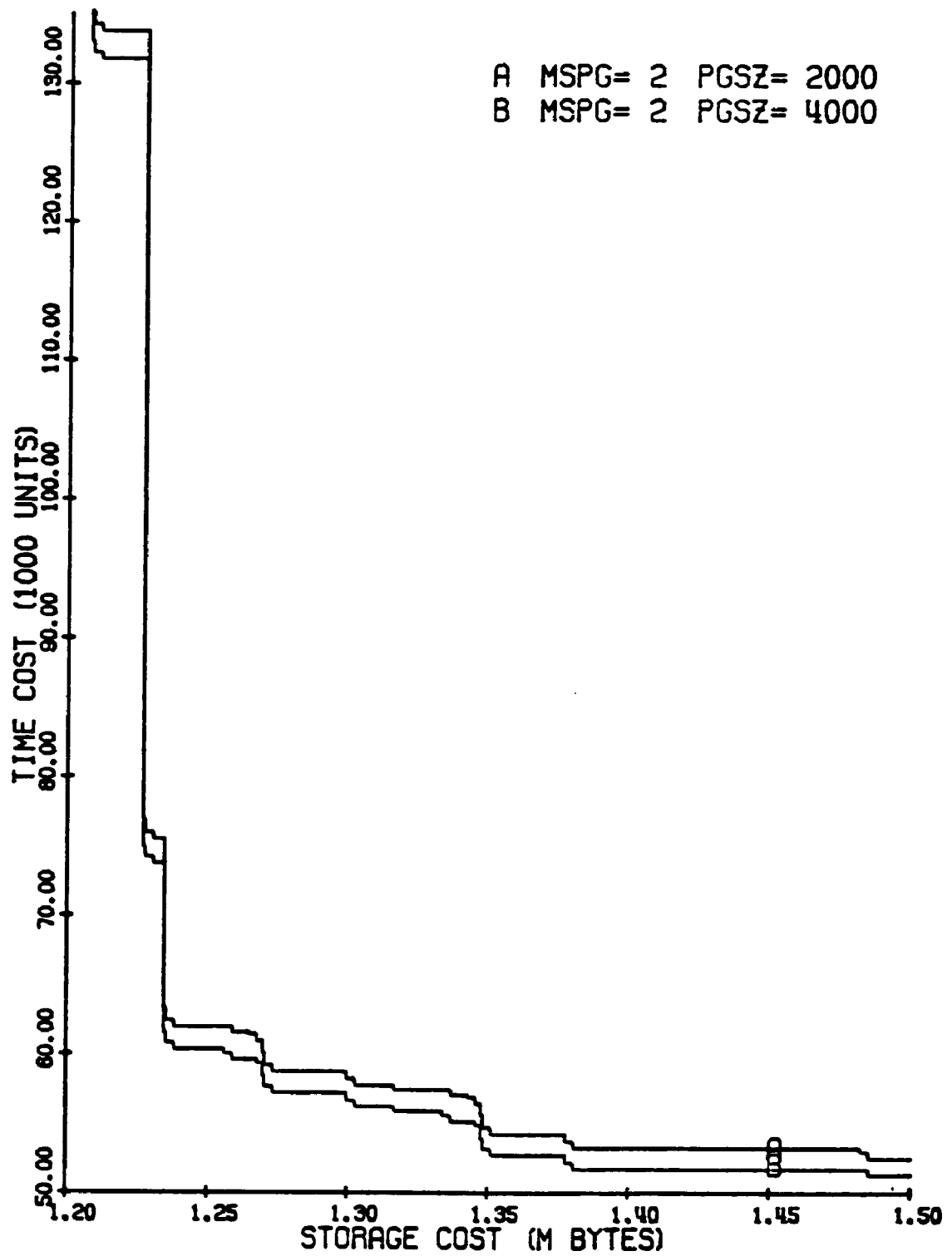


Figure 5.13 Trade-off Graphs for Different PGSZ Values
where MSPG = 2

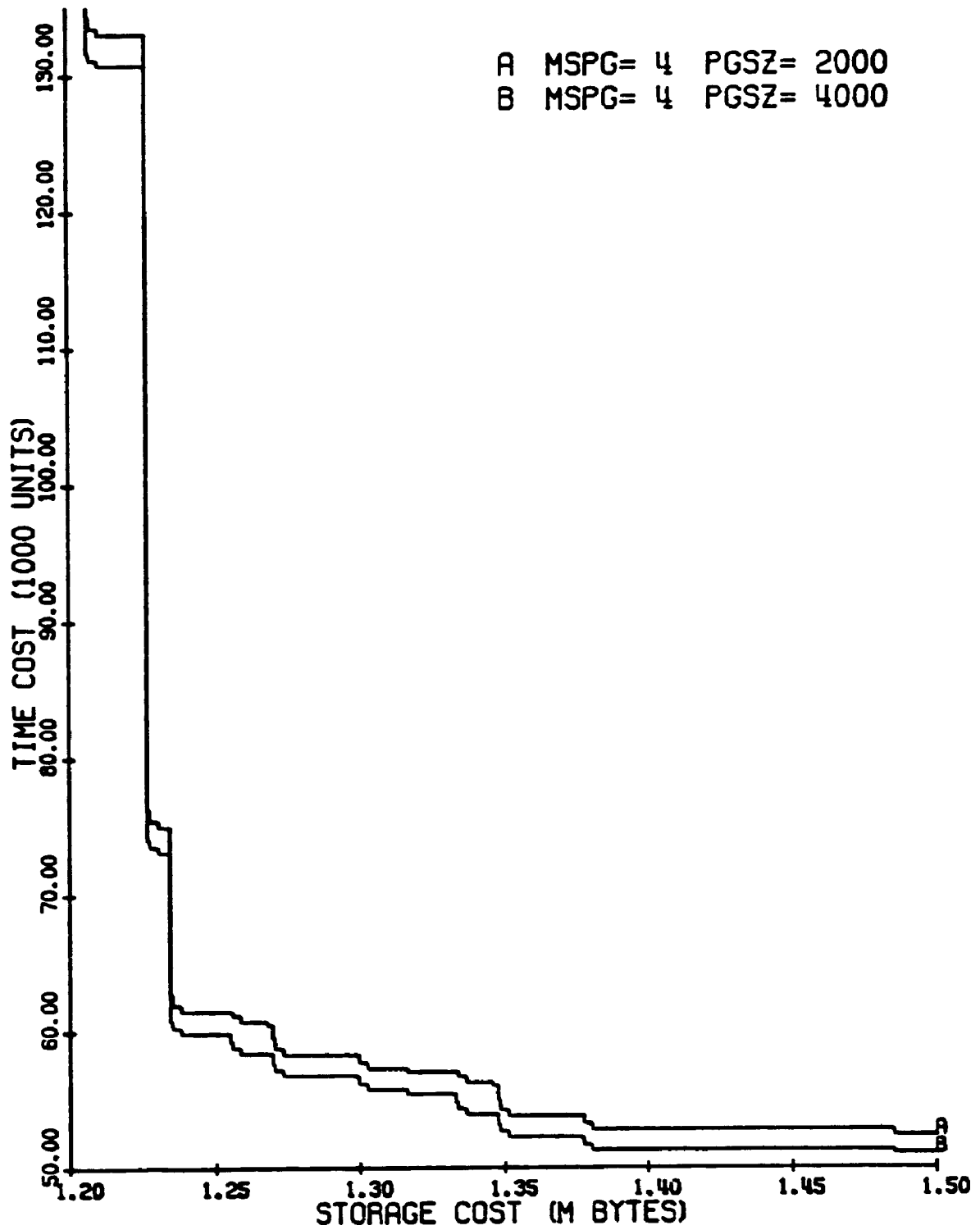


Figure 5.14 Trade-off Graphs for Different PGSZ Values where MSPG = 4

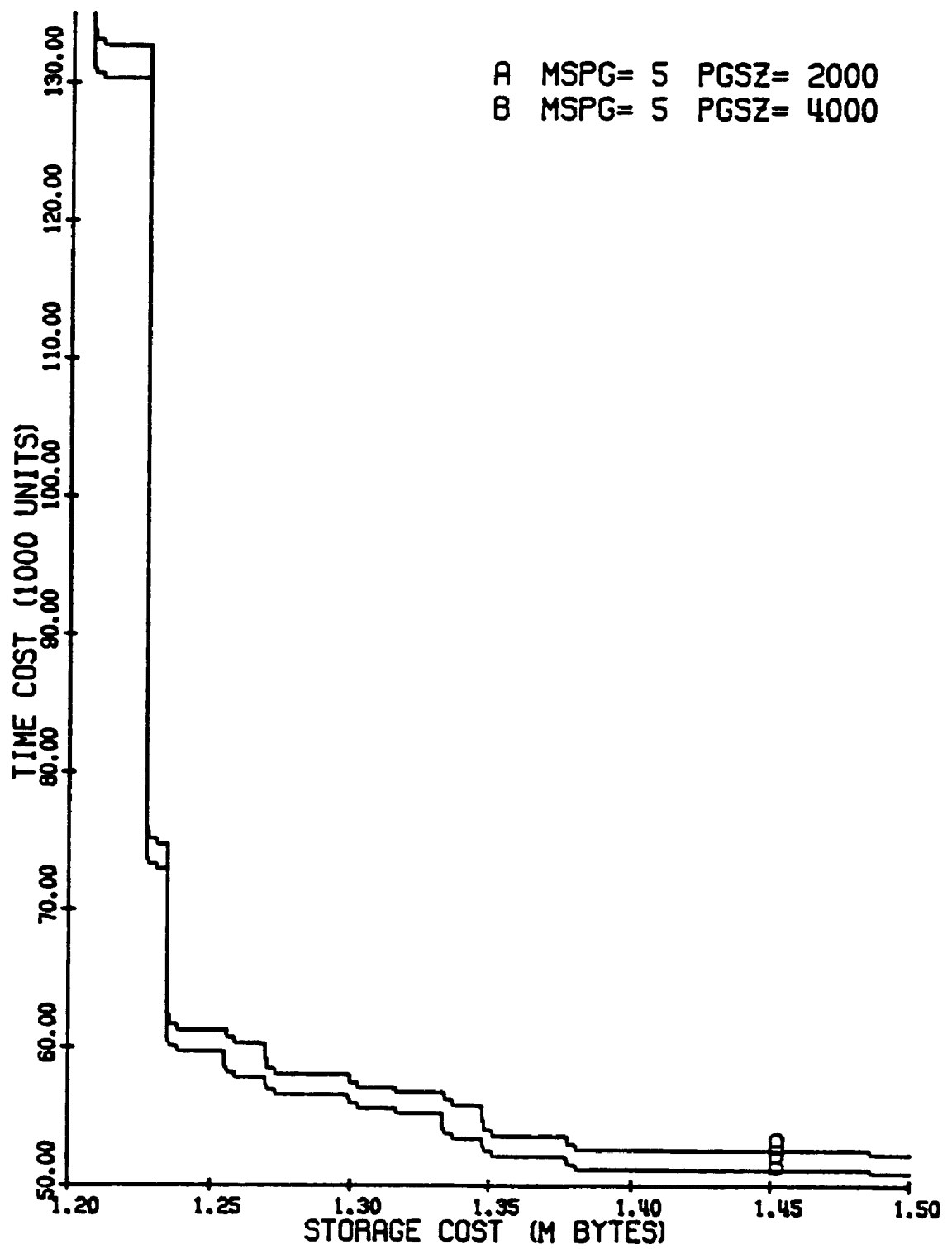


Figure 5.15 Trade-off Graphs for Different PGSZ Values where MSPG = 5

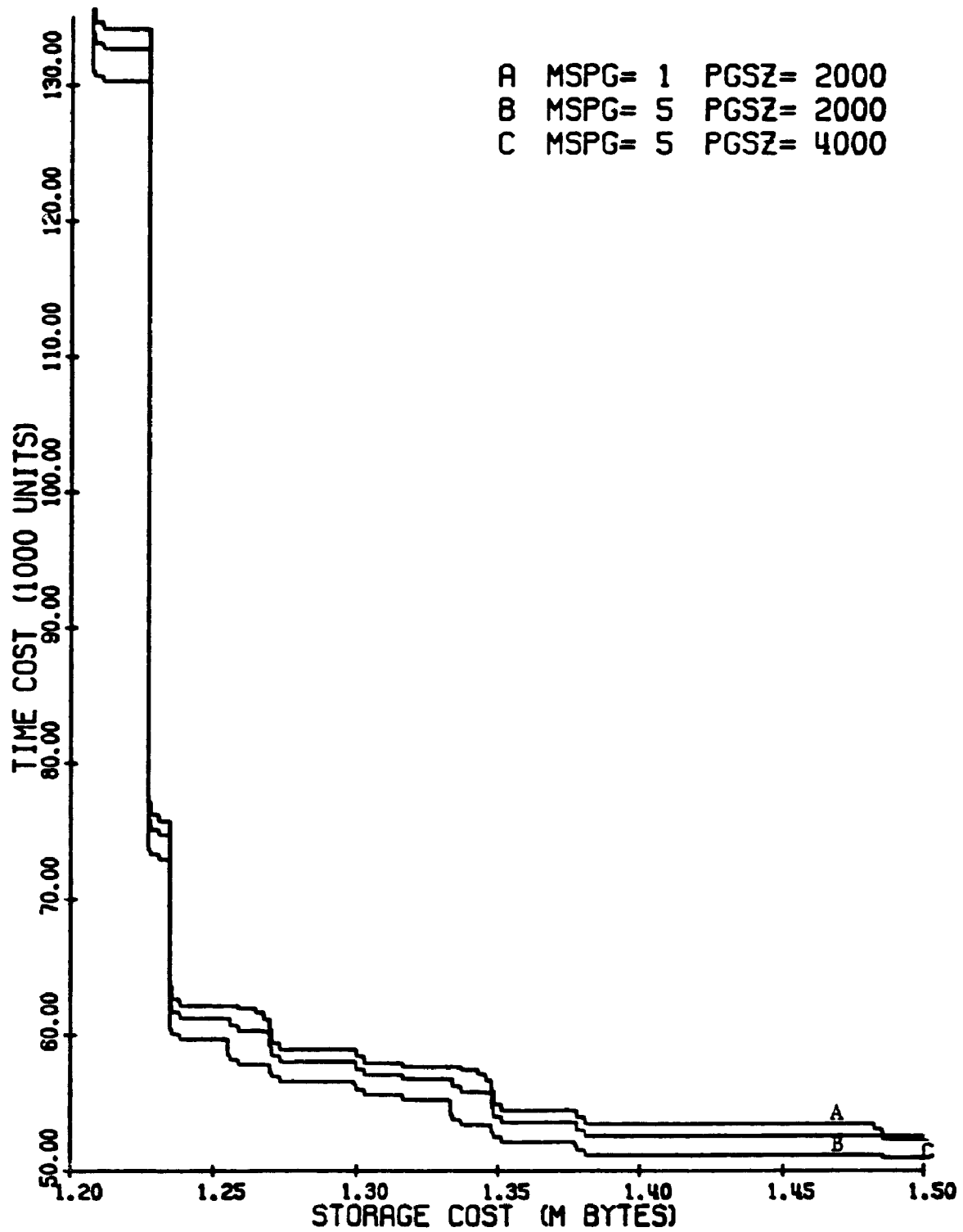


Figure 5.16 .Trade-off Graphs for Different MSPG and PGSZ Values

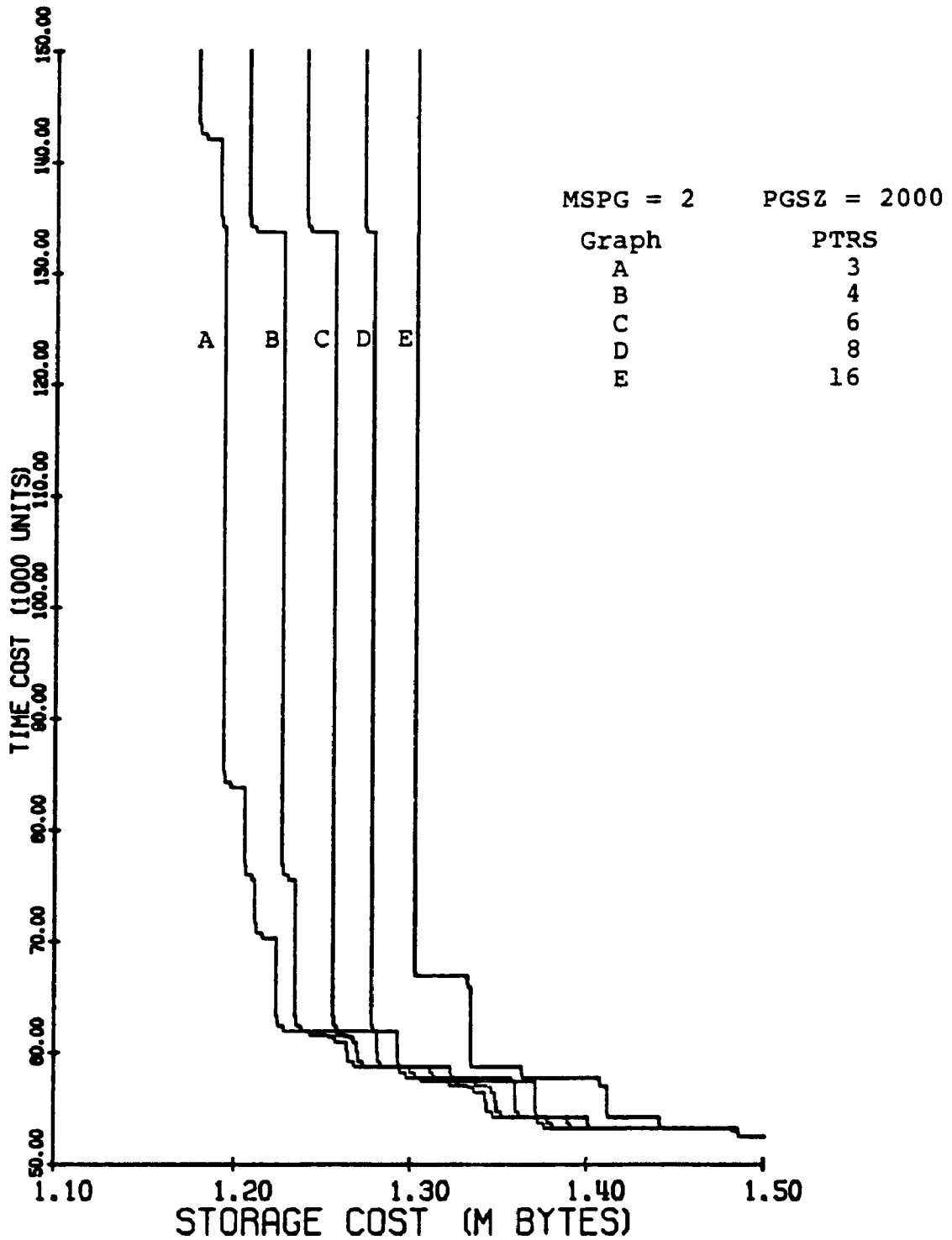


Figure 5.17 Trade-off Graphs for Different PTRS Values where MSPG and PGSZ are Fixed

storage cost) when the pointer size is reduced. For example, we observed in Section 5.3 that the reduction of PTRS from 4 to 3 characters resulted in the design of Figure 5.8 to become optimal for a storage limit of 1,194,000 characters. This design, as we discussed in Section 5.3, makes use of implementation 21 which is a relatively costly implementation in terms of storage space.

Figure 5.17 illustrates that increasing PTRS causes the storage/time graph to move to the right. This is because data base designs that use data base sets (thereby containing pointers) have higher storage costs for the same configurations. Of course, if a data base design is such that it does not include set type definitions, then its storage cost is not affected by changes in PTRS. This is why the lower right sides of all 5 graphs in Figure 5.17 coincide. The upper left sides of each one of these graphs, however, extend upwards along the storage cost of their respective smallest data base designs, which is also the lower bound on the storage limit for each graph. We observe that these lower bounds are different and actually for a fixed storage limit, an optimal configuration may not exist if PTRS is not small enough. For example, if $\text{PTRS} = 8$ characters then no optimal configuration exists if the storage limit is set below 1,272,450 characters, while for the same storage limit an optimal configuration exists for each of the smaller PTRS values, namely 6, 4 and 3.

The final observation about the pointer size variation

is that the number of optimal configurations decreases as PTRS increases. This is because, for higher PTRS values, the lower bound on storage limit increases while the upper bound (storage cost of the most costly optimal configuration) stays the same, and this in turn excludes some configurations (mainly those that include many set types) from the set of optimal configurations.

CHAPTER VI

CONCLUSIONS AND FURTHER RESEARCH

6.1 Conclusions

In this dissertation we have introduced a methodology that partially automates the design of the logical structure of a paged data base. This methodology is intended for use by a data base designer to automatically evaluate and compare a large number of logical structures that can model the information content of a given set of user requirements. This was accomplished by mapping a high level description of user requirements into a precisely formulated dynamic programming problem, whose solution can be easily translated into a set of record structures, record relationships and storage structures. Structural relationships among data elements were constrained to conform to the specifications of the CODASYL DBTG report [CODASYL 1971]. The performance objective used to evaluate alternative structures was the expected number of page fault occurrences for a given set of operations.

A model was presented in Chapter 2 which is capable of describing a specific data management application. Components of this model are data items, data base relations and storage space parameters. Data items and data base relations were characterized by their names, sizes and cardinalities rather than values of actual occurrences.

Storage space parameters include such things as storage space limit, memory page size, pointer size, etc. Expected frequencies of data retrieval and update were specified using a set of 12 data manipulation operations defined as a part of this model.

Various implementation alternatives for a given data base relation were described in Chapter 3. The selection of an implementation for a relation determines whether the component data items of the relation are associated in the data base by physical adjacency or some other technique using pointers. Constraint conditions were defined in Chapter 3 to govern the selection process such that security requirements of the application could be satisfied and also structures unconformable to DBTG specifications would be avoided. It was further observed that even after filtering the implementation alternatives for each individual relation, using the constraint conditions, there might still be several acceptable implementations for each relation. For any reasonable number of relations in an application, the number of possible configurations can be a very large number. In order to evaluate and select the "best" configuration, a two-component cost function was defined for all acceptable relation implementations in Chapter 3. The first cost component was defined as the storage cost and the second component was defined as the time cost of the relation implementation. Four categories of page faults were defined and some approxi-

mate formulas were given for the computation of a page fault in each category. Time costs were given in terms of these probabilities and various other parameters describing relations and implementations.

In Chapter 4 we formulated an optimization problem to select exactly one implementation for each relation, based on models described in previous chapters. Further investigation of the properties of our particular optimization problem led to an efficient algorithm based on the notions of "undominated choices" and "undominated solutions." Those properties of undominated choices (and solutions) that were utilized to improve the efficiency of the algorithm were investigated in a formal manner. Time complexities of the critical stages of the algorithm were also briefly analyzed.

In Chapter 5 we discussed various data base designs that resulted from applying our design methodology to an example application described in Chapter 2. Results of a sensitivity analysis for time cost functions with respect to certain parameters were reported in Chapter 5.

6.2 Further Research

Several areas of possible extensions were suggested in various sections of this dissertation, especially in Chapters 1 and 3. We will now summarize some of the areas where further research may prove to be fruitful.

Some DBTG data model concepts not included in our models can be added, with different degrees of difficulty. These

concepts were enumerated in Chapter 1 and include such things as the modelling of REALMs, INDEXing, SINGULAR SETs, CALL KEY, SEARCH KEY, SUB-SCHEMAS, LOCATION MODE, VIRTUAL data items, ENCODING/DECODING and set membership classes. Incorporation of some of these concepts requires some kind of file organization analyses for which successful results reported by Yao [1974] or Severance [1972] may be utilized.

As pointed out in Chapter 3, the set of implementation alternatives defined in that chapter is not considered to be exhaustive of all conceivable ways of implementing a data base relation. Some possible extensions to the implementation alternatives were suggested and briefly discussed at the end of Chapter 3.

The application of the optimization algorithm (of Chapter 4) to the example problem (of Chapter 2) showed that a significant saving could be achieved in processing time of the second phase of the algorithm by discarding "dominated choices" in the first phase. However, it is theoretically possible that the second phase of the algorithm could be faced with many stages for each of which several undominated choices exist. If at the same time cost components are such that the second phase of the algorithm behaves in the worst case, then the actual number of steps in its execution may become very large. In case such extreme conditions prove to be imminent, some modifications in the definitions of dominance between pairs of choices or solutions may be made

to improve the efficiency of the algorithm. Recall from Chapter 4 that $q = \langle \text{MPL}, S, T \rangle$ is a choice for a relation R if MPL is an acceptable implementation for R and S and T are storage and time costs, respectively, of MPL for R . By our definition of dominance, if $q_1 = \langle 9, 100, 100 \rangle$ and $q_2 = \langle 17, 99, 10'000 \rangle$ are two choices for a relation R , then $\{q_1, q_2\}$ is an undominated set of choices for R , although the time cost of q_2 is much higher than that of q_1 versus a very small advantage of q_2 over q_1 in storage costs. If some kind of relative importance of storage and time costs were assumed, then the notion of dominance could be generalized to, say, functional dominance where a set of functions (maybe two) would be utilized to decide whether or not two choices (or solutions) were undominated. Properties of this kind of dominance must also be investigated and employed in the corresponding optimization algorithm(s).

The effect of concurrency on data access interferences has not been treated in this research. Concurrent access of data bases by multiple users result in contention problems studied, for example, by Collmeyer [1971] and Shemer [1972]. Overhead expenditures required to avoid or resolve such contention problems may considerably affect our cost computations. Further research is necessary before such concepts can be incorporated into our models.

An important class of data base processing operations is the sequential (batched) access. This class of operations

often arises in data base processing for report generation, sorting, merging, etc. Our data manipulation operations can be extended so that sequential operations may be specified.

Time cost functions for such operations must be developed and their effects on other parts of the overall model must be analyzed so that the results may be incorporated into our data base design methodology.

Extensions to our models by the incorporation of more DBTG features were enumerated earlier in this section. We will now briefly discuss the degrees of difficulty in adding some of those features.

In order to accomodate the SINGULAR SET concept, it should be noted that for each data item in a record one such set may be defined. Once the set implementation technique for these sets is fixed then one can add their storage costs to the duplication and aggregation implementations. Time cost functions will be affected through P1 probabilities. Inclusion of this feature is reasonably easy except possibly in the cases where the data items involved are parts of vectors or repeating groups.

INDEXING and SEARCH KEY concepts are basically not difficult to include. It should, however, be noted that since each set may or may not have any number of possible indexing techniques, the total size of the problem may grow very fast.

Extensions concerning the representation of data items, such as VIRTUAL and ENCODING/DECODING concepts were discussed at the end of Chapter 3 (page 169).

Inclusion of the concept of LOCATION MODE (of records) would require some changes in our overall model. Basically, it requires the user requirements model to be changed particularly in the definitions of operations of all location modes are to be accounted for. It is clear that any changes in those operations will propagate to time cost formulations of Chapter 3.

REFERENCES

- [Bachman 1974] Bachman, C.W., "Implementation Techniques for Data Structure Sets", Data Base Management Systems edited by D.A. Jardine, Proc. of the SHARE Working Conference on DBMS's, Montreal, Canada, July 23-27, 1973, North-Holland, American Elsevier, 1974.
- [Beck 1976] Beck, Leland L., "An Approach to the Creation of Structured Data Processing Systems", Proceedings, 1976 SIGMOD, Page 179, International Conference on Management of Data, Washington, D.C., June 2-4, Edited by J.B. Rothnie.
- [Bellman 1957] Bellman, R.E., "Dynamic Programming", Princeton University Press, Princeton, New Jersey, 1957, (A Rand Corporation Research Study).
- [Bellman 1962] Bellman, R.E. and Dreyfus, S.E., "Applied Dynamic Programming", Princeton University Press.
- [Berg 1976] Berg, John L. (ed.), "Data Base Directions: The Next Steps", DATA BASE, Vol. 8, No. 2, Fall 1976 (A quarterly Newsletter of SIGBDP); SIGMOD RECORD, Vol. 8, No. 4, Nov. 1976, Proceedings of the Workshop of the NBS and ACM held at Fort Lauderdale, Fla., Oct. 29-31, 1975.
- [Bubenko 1976] Bubenko, J.A., "From Information Structures to DBTG-Data Structures", Proceedings of Conference on Data: Abstraction, Definition and Structure, March 1976, Salt Lake City, Utah, F.D.T., Bulletin of ACM-SIGMOD, V. 8, N. 2, 1976, pp. 73-85.
- [CODASYL 1971] CODASYL, "Data Base Task Group", Report, CODASYL, April 1971.
- [CODASYL 1973a] CODASYL, "CODASYL COBOL Data Base Facility Proposal", CODASYL, Data Base Language Task Group, March 1973.
- [CODASYL 1973b] CODASYL, "Journal of Development", CODASYL, Data Description Language Committee, June 1973.
- [Collmeyer 1971] Collmeyer, A.J., "Database Management in a Multi-Access Environment", Computer, Vol. 4, No. 6, Nov./Dec./1971, pp. 36-46.
- [Cullinane] Cullinane, "Integrated Data Base Management System, DDL and DML", Cullinane Corporation.
- [Date 1975] Date, C.J., "An Introduction to Database Systems", The Systems Programming Series, Addison Wesley, 1975.

- [DEC] DEC, "DBMS Programmer's Procedures Manual", Digital Equipment Corporation, Document DEC-10-APPMA-A-D.
- [Gerritsen 1975] Gerritsen, Rob, "A Preliminary System for the Design of DBTG Data Structures", CACM, October 1975, Vol. 18, Number 10, pp. 551-557.
- [GUIDE-SHARE 1970] GUIDE-SHARE, "Data Base Management System Requirements", Data Base Requirements Group, Guide-Share, Nov. 1970.
- [Honeywell] Honeywell, "Honeywell Series 600/6000 Integrated Data Store", Reference Manual, Document BR69.
- [Knuth 1968] Knuth, D.E., "The Art of Computer Programming", Vol. 1: "Fundamental Algorithms", Addison-Wesley, Reading, Mass., 1968.
- [Knuth 1973] Knuth, D.E., "The Art of Computer Programming", Volume 3: "Sorting and Searching", Addison-Wesley, Reading, Mass., 1973.
- [Mitoma 1975] Mitoma, M.F., "Optimal Data Base Schema Design", Ph.D. Thesis, University of Michigan.
- [Severance 1972] Severance, D.G., "Some Generalized Modeling Structures for Use in Design of File Organizations", Ph.D. Thesis, University of Michigan, 1972.
- [Shemer 1972] Shemer, J.E. and Collmeyer, A.J., "Database Sharing: A Study of Interference, Roadblock and Dead-lock", Proceedings of 1972 ACM-SIGFIDET Workshop, DATA Description Access and Control, Edited by A.L. Dean, Denver, Colorado, Nov. 29-30, Dec. 1/1972, pp. 147-162.
- [Taylor 1976] Taylor, R.W. and Frank, R.L., "CODASYL Data-Base Management Systems", ACM-Computing Surveys: Volume 8, Number 1, March 1976, pp. 67-103.
- [Tuel 1976] Tuel, W.G., Jr., "An Analysis of Buffer Paging in Virtual Storage Systems", IBM, J. Res. Develop, Vol. 20, No. 5, September 1976, p. 518.
- [UNIVAC 1974] UNIVAC, "Data Manipulation Language" UP-7992 and "Schema Definition" UP 7907, SPERRY UNIVAC 1100 Series Data Management System (DMS 1100).
- [Yao 1974] Yao, Shi Bing, "Evaluation and Optimization of File Organizations through Analytic Modelling", Ph.D. Thesis, University of Michigan, 1974.